МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ
Институт системной и программной инженерии
и информационных технологий (Институт СПИНТех)

Лабораторный практикум по курсу "Интеллектуальные системы" (09/23 – 01/24)

Лабораторная работа 1
Линейная регрессия как задача контролируемого (индуктивного) обучения.

На этом занятии компьютерного практикума вы изучите *линейную регрессию* и получите представление, о том, как данная процедура применяется для обработки данных. Во многих приложениях нейронные сети реализуют регрессионные вычисления (статистический метод исследования влияния одной или нескольких независимых переменных x_1, x_2, ... , x_p на зависимую переменную *y)* или решение задач классификации. При этом, в частности, линейная регрессия (англ. *Linear regression*) представляет собой регрессионную модель зависимости одной (объясняемой, зависимой) переменной *y* от другой или нескольких других переменных (факторов, регрессоров, независимых переменных) *x* с линейной функцией зависимости.

Прежде чем приступить, собственно к программированию, настоятельно рекомендуется ознакомиться с материалом лекций, а также с дополнительными материалами, имеющими отношение к задаче градиентного спуска и к области минимизации функционалов.

Файлы, включенные в задание:

- *lab_intelligent_systems_liner_regressopn.ipynb* - ноутбук, реализующий пошаговое выполнение задания 1 части задания по разделу «Линейная регрессия с одной переменной» (простая линейная регрессия) и выполнение 2-й части задания по разделу «Многомерная линейная регрессия»;
- *service_center.csv, ex1data1.txt* - набор данных для простой линейной регрессии;
- *cost_apartments.csv, ex1data3.txt* - набор данных для многомерной линейной регрессии;

Вам необходимо будет реализовать следующие функции:

- *plotData* - функция, отображающая набор данных;
- *computeCost* - функция, с помощью которой производится вычисление функциистоимости (целевой функции);
- *gradientDescent* - функция, реализующая метод градиентного спуска;
- *computeCostMulti* - функция, с помощью которой производится вычисление функции стоимости (целевой функции) для многомерной линейной регрессии;
- *gradientDescentMulti* - функция, реализующая метод градиентного спуска для случая нескольких переменны

- f*eatureNormalize* - функция, с помощью которой производится нормализация признаков;
- *normalEqn* - функция, реализующая вычисления по методу наименьших квадратов.

В этой лабораторной работе Вы будете использовать ноутбук lab_intelligent_systems_liner_regressopn.ipynb. Необходимо дописать функции по инструкциям упражнения.

1. Линейна регрессия с одной переменной

В этой части упражнения Вы реализуете линейную регрессию с одной переменной для прогнозирования прибыли сервисного центра по обслуживанию, например, изделий бытовой электроники. Допустим, рассматриваются кандидатуры нескольких городов для такого сервисного центра. Сеть подобных сервисных центров уже функционирует и у Вас есть данные по прибыли в зависимости от числа жителей города (или от количества изделий, находящихся на обслуживании в городе). Вы бы хотели использовать эти данные с тем, чтобы понять, в каком городе открыть дополнительный сервисный центр или спрогнозировать прибыль в зависимости от числа жителей. Файл *service_center.csv* содержит набор данных для задачи с линейной регрессией. Первый столбец – число жителей города (количество проданных изделий бытовой электроники), второй – прибыль центра в этом городе. Отрицательные значения прибыли означают убыток.

1.1. Отображение данных на экране

До начала выполнения любого задания было бы полезным представить данные в графическом виде. Для визуализации набора данных (в дальнейшем, при решении аналогичной задачи с помощью нейронной сети, этот набор данных представляет собой обучающие данные) можно использовать точечную диаграмму, т. к. имеется только два свойства – прибыль и население города (количество аппаратов). Многие задачи из реальной жизни имеют большее количество свойств и параметров, так что построить их на двумерном графике, конечно же, не удастся.

Ваша задача – построить график, завершив код в функции *plotData*. Результат должен выглядеть так, как показано на Рис. 1

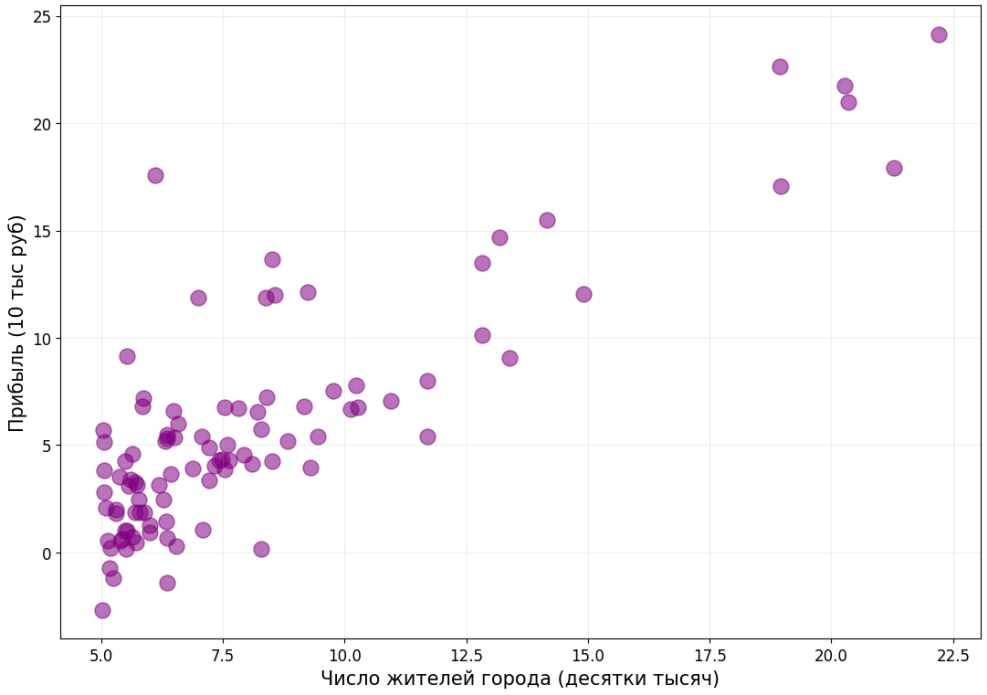


Рис. 1. Точечный график обучающих данных

1.2. Метод градиентного спуска для одной переменной

В этой части производится настройка параметров линейной регрессии θ на основе метода градиентного спуска.

1.3. Вычисления

Задача линейной регрессии – минимизировать функцию стоимости (затрат)

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2,$$

в котором уравнение гипотезы $h_\theta(x)$ представляет собой линейную модель. Напомним, что параметры модели, это значения θ_j. Именно их необходимо подобрать с целью минимизации стоимости *J(θ)*. Один из способов это сделать - использовать алгоритм наискорейшего спуска, где каждая итерация обновляет

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta x^i - y^i)x_j^i$$

(одновременно обновляет *θj* для всех *j).*

С каждым шагом градиентного спуска параметры θ_j становятся ближе к оптимальным значениям, при которых достигается наименьшая величина стоимости *J(θ)*.

Указание: В формуле выше каждый пример хранится в строке матрицы X. С целью учёта дополнительного коэффициента θ_0, к X добавлен единичный столбец, что позволяет рассматривать θ_0 как дополнительный параметр. С учётом этого, формула, описывающая применение градиентного спуска, после вычисления частных производных функции стоимости $J(\theta_0, \theta_1)$ относительно θ_0 и θ, представленная в покомпонентной форме имеет вид:

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}),$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\,x^{(i)}.$$

1.4. Выполнение

В ноутбуке данные для линейной регрессии уже определены. Добавим дополнительную размерность с целью учёта θ_0. Кроме того, установим начальные параметры, равные нулю, а также скорость обучения равную 0.01.

```
X = np.stack([np.ones(m), df['population']], axis=1) # Добавляем единичный столбец к X
theta = np.zeros((2, 1)) # Инициализируем начальные значения
iterations = 2000 # Количество итераций
alpha = 0.01 #Скорость обучения
```

1.5. Вычисление стоимости J(θ)

Осуществляя вычисления на основе метода градиентного спуска в процессе минимизации функции стоимости *J* (*θ*), полезно отслеживать сходимость вычислительного процесса.

В этой части задания предстоит написать функцию стоимости *J(θ)*, при помощи которойможно отслеживать сходимость уже реализованного процесса спуска.

Следующее задание – завершить код в функции computeCost, содержащий функцию *J(θ)*.

Указание: Переменные X и y - не скалярные величины, а матрицы, чьими строками являются примеры из обучающей выборки (набора данных).

Закончив программировать функцию стоимости *J(θ),* запустите скрипт, который выполнит функцию *computeCost* используя θ с начальными значениями, равными нулю. В результате вычислений на экране появится стоимость примерно равная 32.07.

2.1.1 Метод градиентного спуска

Далее завершите код градиентного спуска в функции *gradientDescent.m*. Необходимый цикл был заранее подготовлен и Вам остается лишь дополнить корректировку θ для каждой итерации.

В процессе программирования учтите, что стоимость *J(θ)* зависит от параметра θ, а не от *X* и *y*. Т.е. необходимо оптимизировать *J(θ)* изменяя значения вектора θ, а не *X* и *y*. Хороший способ проверить правильность работы градиентного спуска – *посмотреть* на значения *J(θ)* и проверить их уменьшение с каждой итерацией. Начальный код *gradientDescent.m* вызывает функцию *computeCost* на каждой итерации и выводит на экран значение стоимости. При безошибочном функционировании кода и функции *computeCost*, значения функции стоимости *J(θ)* не возвращают и вычислительный процесс должен сходиться к устойчивой величине в конце вычислений. Как это показано на рисунке 2

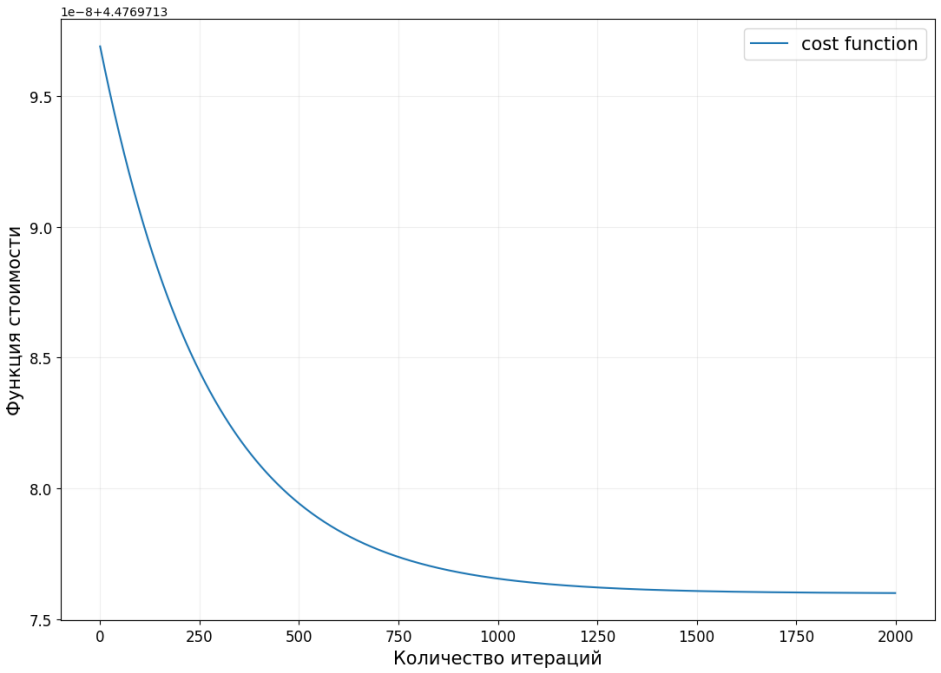


Рис. 2. График изменения значения функции стоимости

После завершения ноутбук использует ваши конечные параметры для построения линейного тренда. Результат должен быть схож с рисунком

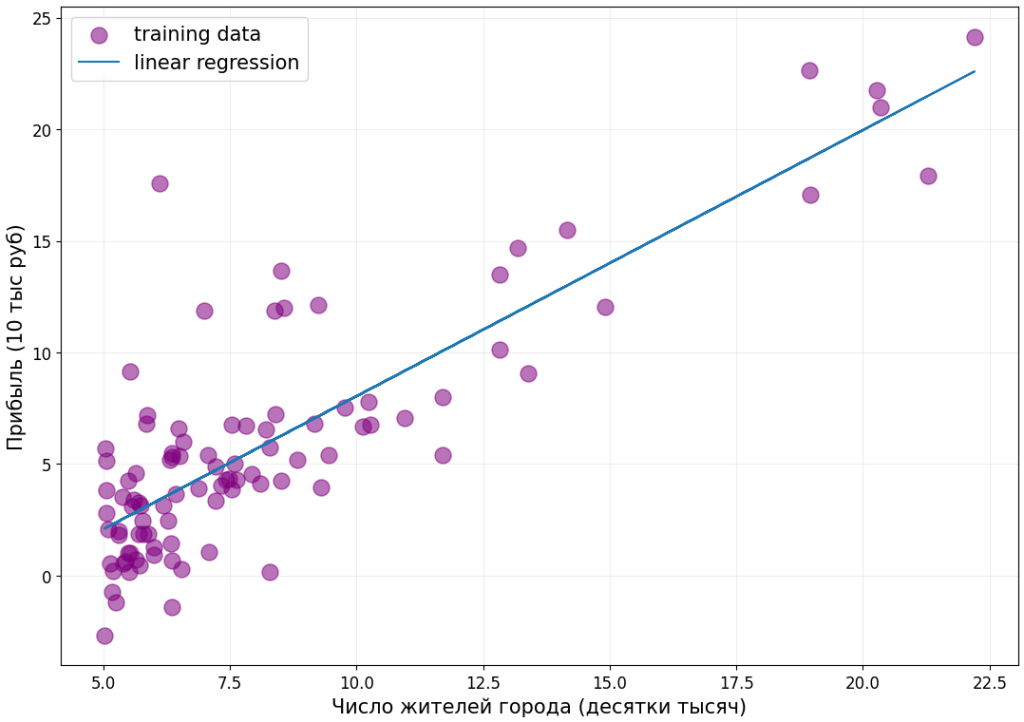


Рис.4. Представление данных в виде тренда линейной регрессии

Оптимальные величины θ могут быть использованы для прогнозирования прибыли в городе с числом жителей (проданных изделий бытовой электроники) 35,000 и 70,000. Обратите внимание на способ, которым следующие строки кода в ноутбуке используют перемножение матриц, вместо явного сложения или использования цикла для вычисления вашего прогноза. Это пример векторизации (преобразования данных в векторную форму).

```
predict1 = np.array([1, 3.5]).dot(theta)
predict2 = np.array([1, 7]).dot(theta)
```

2.2 Отладка

Указание: несколько полезных советов для осуществления градиентного спуска:

- Некоторые ошибки возникают из-за несоответствия размеров матриц для операций сложения или умножения.

2.3 Наглядное представление стоимости J(θ)

Для лучшего понимания поведения целевой функции (функции стоимости) *J(θ)* постройте двумерный график в зависимости от значений θ_0 и θ_1. Вам не потребуется для этого писать новый код, но предстоит разобраться, как функционирует уже написанная программа.

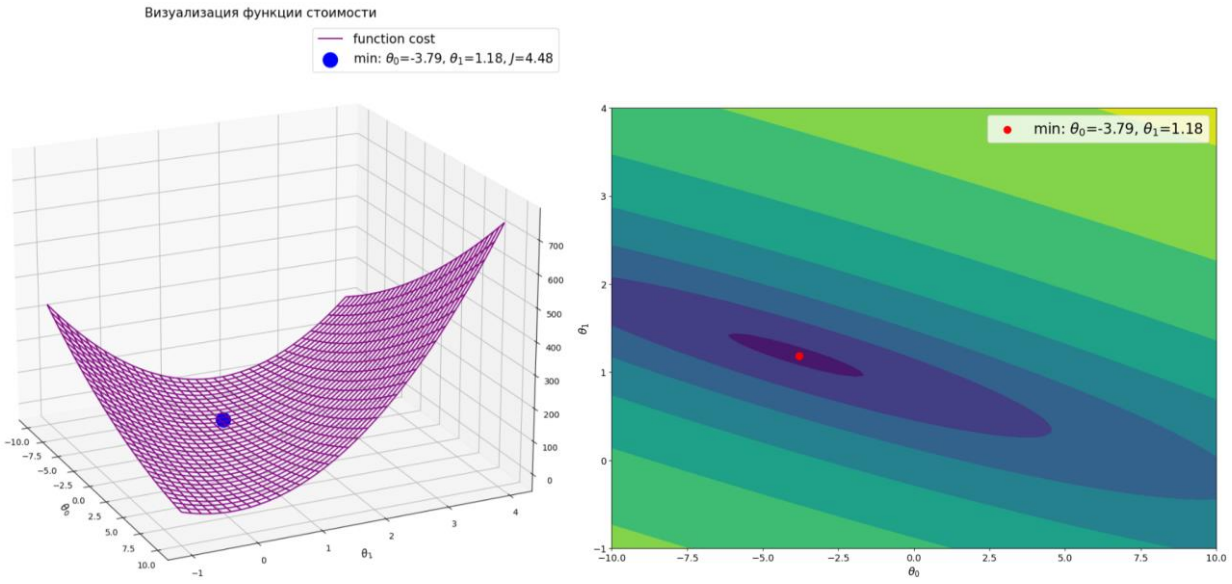
Следующий шаг вычислить набор значений *J(θ)* с использованием написанной вами ранее функции *computeCost*.

```
# Инициализация J_vals to a matrix of 0's
J_vals = np.zeros((len(theta0_vals), len(theta1_vals)))

# Заполнение J_vals
for i in range(len(theta0_vals)):
    for j in range(len(theta1_vals)):
        t = [theta0_vals[i], theta1_vals[j]]
        J_vals[i,j] = computeCost(X, y, t)
```

После выполнения этих строк кода Вы будете располагать двумерным массивом значений *J(θ)*.Затем ноутбук использует эти значения для построения поверхности и контурного графика *J(θ)* с помощью функций *plot_wireframe* и *contourf*. Графики будут выглядеть примерно так, как показано

на рисунке. 5:



а) поверхность б) контурный график с обозначенным минимумом

Рис. 5. Функция стоимости $J(\theta)$

Цель данного графического представления - показать влияние θ_0 и θ_1 на $J(\theta)$. Целевая функция (функция стоимости) $J(\theta)$ приняла форму чаши или цилиндра и имеет глобальный минимум (что будет отчетливее видно на контурном графике, нежели на поверхности). Однако, если прологарифмировать функцию стоимости, то глобальный минимум будет виден ещё нагляднее, как это показано на рисунке 6. Этот минимум и есть оптимальная точка θ_0 и θ_1 к которой градиентный спуск с каждым шагом всё ближе и ближе.

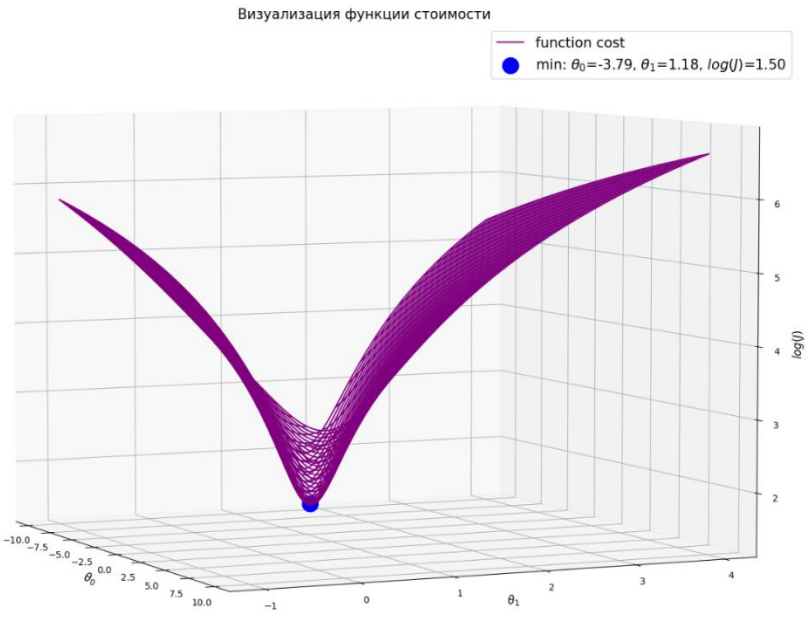


Рис. 6. Прологарифмированная функция стоимости $J(\theta)$

3. Линейная регрессия для нескольких переменных *

После успешного выполнения предыдущих заданий Вы лучше владеете методом линейной регрессии и можете им пользоваться для решения практических задач. Это, в свою очередь,позволит в дальнейшем понять процедуру предсказания или интерполяции данных, которая реализуется на основе нейронно-сетевого подхода.

Здесь Вы осуществите метод линейной регрессии для нескольких переменных, например, для

оценки стоимости, приобретаемой квартиры, состоящей из нескольких комнат. Один из способов сделать это – накопить информацию о недавних подобных сделках и сконструировать ценовую модель.

Файл cost_apartments.csv содержит набор недавних цен на квартиры. Первый столбец – площадь в м^2, второй – количество комнат, третий - цены.

3.1 Нормализация свойств

Ноутбук сначала загрузит и отобразит выборочные данные. Обратите внимание на то, что численные значения площади квартиры, выраженной в м^2, во много раз больше численных значений, соответствующих количеству комнат в квартире. В таких случаях, когда свойства отличаются значительно (в том числе, на несколько порядков), сначала проводят их нормализацию, что существенно ускоряет процесс схождения градиентного спуска.

Ваша задача завершить функцию *featureNormalize* с тем, чтобы:

- вычесть среднее значение каждого свойства из набора данных;
- затем поделить полученные значения на соответствующие стандартные отклонения.

≪Стандартное отклонение≫ это измерение (*max* - *min*) диапазона значений для каждого конкретного свойства (большинство отклонений располагается в пределах ± 2 единиц от среднего). В python стандартное отклонение вычисляет функция *std с помощью numpy*. Например, внутри *featureNormalize* величина data['squera'] содержит все значения обучающих данных - площадей квартир, так что команда np.std(data['squera']) вычисляет стандартное отклонение данного признака. Во время вызова *featureNormalize* первый столбец с единичными значениями x0 = 1, упомянутый выше, пока не был добавлен к X. Предстоит сделать эти приготовления для всех свойств так, чтобы составленная Вами программа работала с массивами данных любых размеров (любым количеством свойств/примеров). Заметим, что каждому свойству соответствует один столбец матрицы X.

Указание: при нормализации свойств важно сохранять данные, используемые для нормализации – *средние значения* и *стандартные отклонения*. После обучения параметров модели мы сможем получить новые данные – рассчитать цены на интересующие нас квартиры. Эти новые значения **x** (площадь квартиры, количество комнат и т.д.) следует снова нормализовать, используя рассчитанные ранее средние значения и стандартные отклонения.

3.2 Метод градиентного спуска для нескольких переменных

Ранее Вы уже применяли метод градиентного спуска в процессе решения задачи линейной регрессии для одной переменной. Единственным отличием теперь будет введение дополнительного свойства (признака) в матрицу X. Функция гипотезы и итеративное правило обновления функции градиентного спуска останутся без изменений.

Теперь необходимо завершить функции вычисления стоимости и градиентного спуска для линейной регрессии с несколькими переменными в функциях *computeCostMulti* и *gradientDescentMulti*. Вы также можете использовать функции из предыдущих заданий, если они написаны в векторизованном виде (т.е. поддерживают многомерные входные данные). Убедитесь, что ваше решение поддерживает любое число свойств и векторизовано!

Указание: в случае с несколькими переменными функция стоимости можно также записать в следующем векторизованном виде:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

где

$$X = \begin{bmatrix} — \left(x^{(1)}\right)^T — \\ — \left(x^{(2)}\right)^T — \\ \vdots \\ — \left(x^{(m)}\right)^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

Векторизованная форма более эффективна в вычислительном плане. Если Вы хорошо разбираетесь матричными операциями, то Вам будет несложно доказать, что обе формы записи эквивалентны.

3.3. Выбор скорости обучения

В этой части задания необходимо выяснить скорость обучения для используемых данных, в которой результат сходится быстрее. Вы можете изменять скорость обучения.

В ноутбуке необходимо запустить функцию *gradientDescent* около 50 раз при соответствующей скорости обучения. Ваша функция должна также возвращать массив с данными *J(θ)* в вектор *J*. После последней итерации программа построит график значений *J(θ)* в зависимости от числа итераций. Если выбранные величины скорости обучения совпали с представленными в руководстве (см. ниже), то зависимость расположится ближе к рисунку 7.

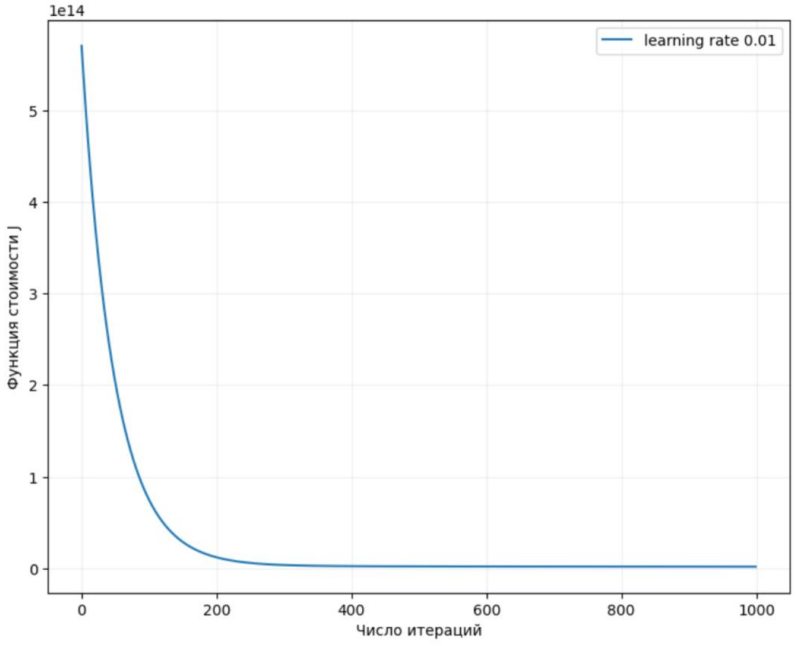


Рис. 7. Сходимость метода градиентного спуска с соответствующим диапазоном скоростей обучения

В противном случае, если значения *J(θ)* увеличиваются или расходятся (в бесконечность), – подберите другой диапазон скорости обучения и попробуйте снова. Рекомендуется попробовать значения скорости обучения α с уменьшением, скажем, в 3 раза (т.е. 0.3, 0.1, 0.03, 0.01 и т.д.).

Указание: Если величина α очень большая, то *J(θ)* может разойтись, что приведёт к чрезмерно большим значениям для вычислений. В этом случае python вернёт ≪NaN≫, что ознает ≪not a number≫ (не число). Это частая ситуация при наличии каких-либо операторов с $-\infty$ или $+\infty$.

Указание: для сравнения влияния скорости обучения на сходимость можно построить *J(θ)* для нескольких α на одном графике. Обратите внимание на кривые сходимости для разных α. С малым α скорость схождения до оптимального значения занимает больше времени относительно случая с большим α. Кроме того, с большим α система может не сойтись вовсе или разойтись!

Используя оптимальную скорость обучения из найденных, запустите скрипт из ноутбука с тем, чтобы найти окончательные параметры θ. Далее, используя полученные θ, спрогнозируйте стоимость трехкомнатной квартиры площадью 60 м^2. Вы примените это значение далее для проверки задания по *аналитическим уравнениям (системы нормальных уравнений)*. Не забудьте нормализовать свойства перед выполнением!

3.3 Система нормальных уравнений. МНК-решение

Замкнутая форма для линейной регрессии записывается как:

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

Использование этой формулы не требует нормализации свойств. Точный результат достигается без использования итераций: нет цикла как в градиентном спуске. Завершите код в *normalEqn* используя приведённую формулу, для вычисления θ. Здесь не требуется нормализация, однако это не значит, что не нужно добавить единичный столбец (учёт θ_0) к X.

Используйте этот метод для оценки стоимости трехкомнатной квартиры площадью 60 м^2, методом наименьших квадратов. Можете использовать предыдущее задание с использованием метода градиентного спуска для проверки (должно получиться сходное число).