Отчёт по лабораторной работе 1: Методы градиентного спуска и метод Ньютона

Бондарь Федор Николаевич

24.02.2024

В рамках данной лабораторной работы было предложено реализовать методы градиентного спуска и метод Ньютона, а также поэкспериментировать с параметрами данных алгоритмов, сравнить результаты на разных входных данных и оценить их работу на реальной задаче логистической регрессии.

Задания

Данный раздел несёт цель кратко описать процесс реализации заявленных методов и промежуточные результаты. Код доступен в приложенных файлах optimization.py и oracles.py.

Задание 2

По итогам выполнения задания 2 были реализованы метод градиентного спуска (функция gradient_descent в модуле optimization) и процедуру линейного поиска (метод line_search в классе LineSearchTool в модуле optimization). В ходе разработки алгоритма линейного поиска для поиска точки, удовлетворяющей сильным условиям Вульфа, была использована библиотечная функция scalar_search_wolfe2. Был учтён тот факт, что эта функция иногда не сходится: в таком случае запускается процедура дробления шага (бэктрекинг) для поиска точки, удовлетворяющей условию Армихо.

Задание 3

В данном задании было необходимо получить формулы для значения, градиента и гессиана функции логистической регрессии в векторном виде. Пусть задана логистическая регрессия с параметрами A (матрица признаков, размером $m \times n$), b (вектор лейблов, размером $m \times 1$) и r (коэффициент регуляризации, скаляр). Напишем формулу вычисления логистической регрессии в точке x (вектор в признаковом пространстве, размером $n \times 1$):

$$func(x) = \frac{1}{m} 1_m \times \ln(1 + e^{-b \odot (A \times x)}) + \frac{r}{2} ||x||_2^2,$$

где 1_m является единичным вектором размером $1 \times m$. В ответе получился скаляр, как и должно быть. Теперь напишем формулу для градиента:

$$grad(x) = \frac{1}{m} A^T \times (\frac{1}{1 + e^{A \times x}} - \frac{b+1}{2}) + r \cdot x.$$

В ответе получился вектор размера $n \times 1$, как и должно быть. Осталось написать формулу для гессиана в векторной форме:

$$hess(x) = \frac{1}{m} A^T \times diag(\frac{1}{1 + e^{b \odot (A \times x)}} \odot (1 - \frac{1}{1 + e^{b \odot (A \times x)}})) \times A + r \cdot I_n,$$

где diag означает взятие диагональных элементов матрицы и получение из них вектора, I_n означает единичную матрицу размера $n \times n$. В ответе получилась матрица размера $n \times n$, как и должно быть.

Задание 4

В данном задании был реализован оракул для логистической регрессии по формулам, описанным выше. В реализации были учтены замечания относительно векторизации всех вычислений, а также их оптимальности.

Задание 5

В данном задании был реализован подсчет разностных производных (функции grad_finite_diff и hess_finite_diff в модуле oracles). Для проверки методов была сгенерирована модельная выборка, после чего было проведено сравнение величин, которые выдают функции grad и hess с вероятностными аппроксимациями (код доступен в приложенном файле experiments.ipynb). В результате сравнений было выяснено, что значения градиента и гессиана правильно приближаются с заданной точностью, значит, функции были разработаны верно.

Задание 6

В данном задании был реализован метод Ньютона. При разработке было учтено ограничение на гессиан (симметричная положительно определенная матрица), поэтому при решении систем уравнений было использовано разложение Холецкого.

Эксперименты

В данном разделе описаны эксперименты по исследованию траектории градиентного спуска, зависимости быстроты сходимости от числа обусловленности и размерности, стратегии выбора длины шага в градиентном спуске и методе Ньютона. Код доступен в приложенном файле experiments.ipynb.

Эксперимент 1: Траектория градиентного спуска на квадратичной функции

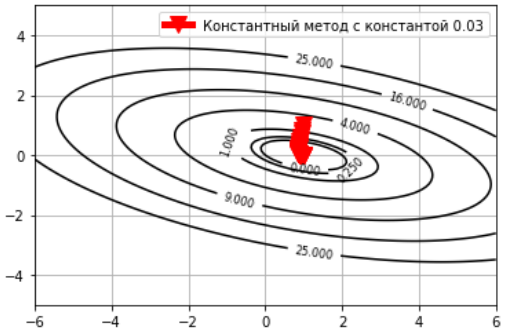
Объявим оракул с параметрами:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 5 \end{pmatrix}, \; b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

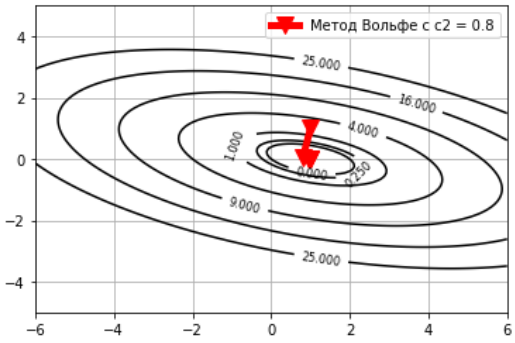
Начальная точка алгоритма - $\binom{1}{1}$. Посмотрим последовательно на траектории методов: константный, Вольфе и Армихо:

```
Число шагов:  104
Минимум:  [0.97947038 0.00484641]
```

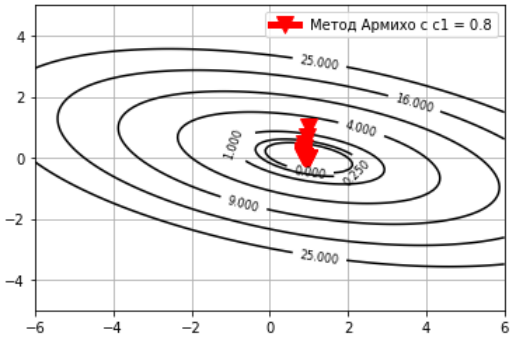
```
C:\Users\bonda\optimization-methods\lab1\plot_traje
'linewidth'
  CS = plt.contour(X, Y, Z, levels=levels, colors='
```




```
Число шагов:  7
Минимум:  [0.98897059 0.00367647]
```

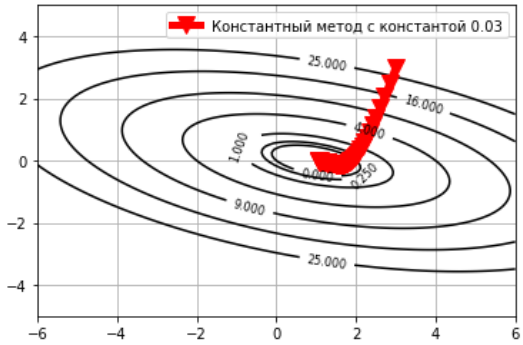


```
Число шагов:  50
Минимум:  [0.97965873 0.00480193]
```

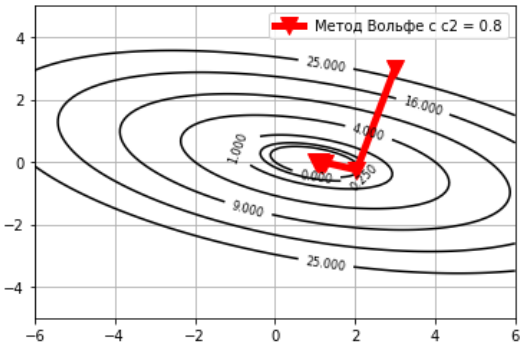


Теперь возьмём другую начальную точку - $\binom{3}{3}$ - и повторим наблюдения:

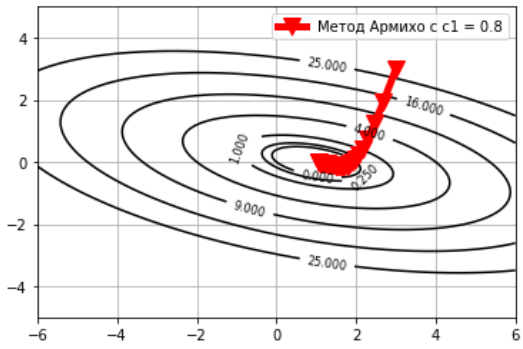
```
Число шагов:  124
Минимум:  [ 1.07065755 -0.01667998]
```




```
Число шагов:  9
Минимум:  [ 1.04366304 -0.00396937]
```



```
Число шагов:  60
Минимум:  [ 1.06824394 -0.01611021]
```



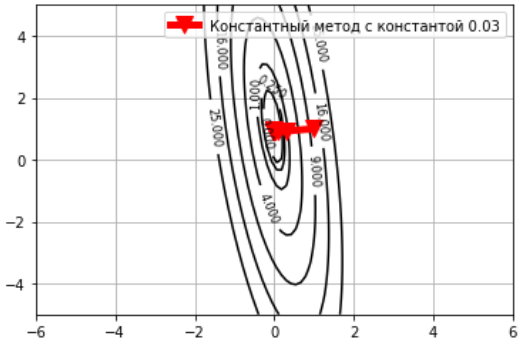
После двух этапов наблюдений можно сделать вывод, что работа алгоритма спуска зависит от выбора начальной точки: при выборе более дальней от минимума точки алгоритм сходился дольше, что логично. При этом видно, что метод Вольфе среди всех стратегий выбора шагов сходится быстрее в несколько раз независимо от выбора начальной точки.

Теперь попробуем взять оракул с функций, у которой число обусловленности выше:

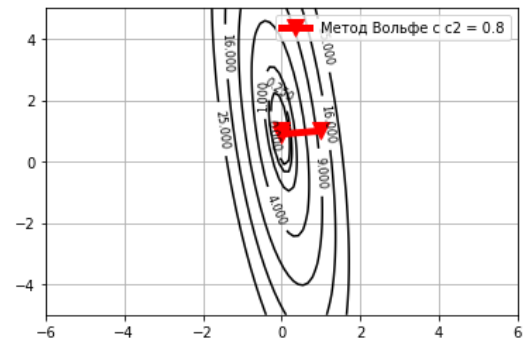
$$A = \begin{pmatrix} 23 & 2.5 \\ 2.5 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Начальная точка алгоритма - $\binom{1}{1}$. Теперь повторим эксперимент:

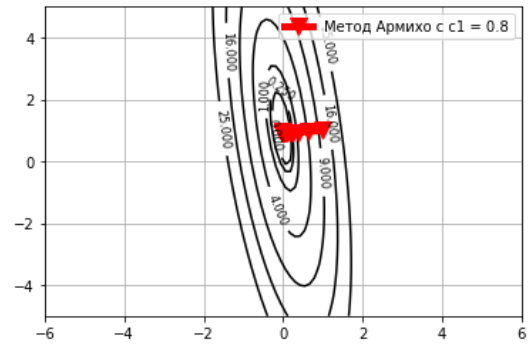
```
Число шагов:  29
Минимум:  [-0.01841136  0.97267674]
```



```
Число шагов:  3
Минимум:  [-0.0234007   1.02365066]
```




```
Число шагов:  54
Минимум:  [-0.01827573  0.9714679 ]
```



Видно, что увеличение числа обусловленности повлияло только на константный метод, он стал работать быстрее.

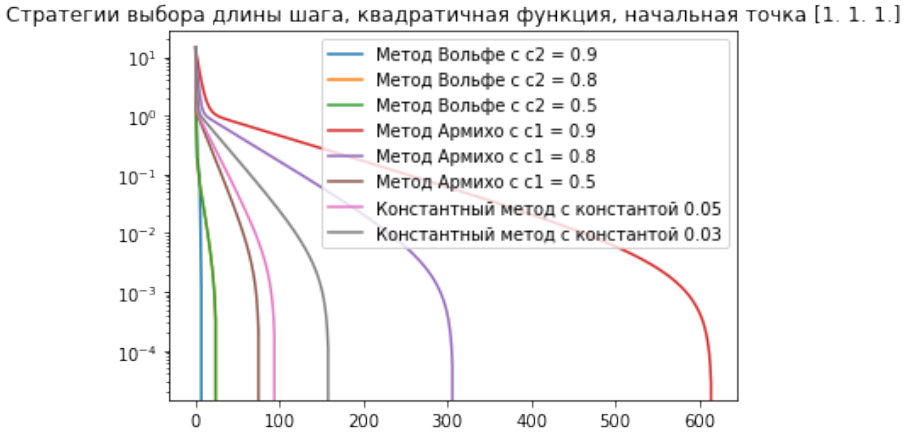
Эксперимент 2: Зависимость числа итераций градиентного спуска от числа обусловленности и размерности пространства

Сгенерируем данные и запустим градиентный спуск с методом Вольфе, который показал себя лучше всего в прошлом эксперименте. В качестве значений размерности n возьмём 5, 10, 15, а в качестве чисел обусловленности 10, 20, 30, 40. Построим график по результатам:

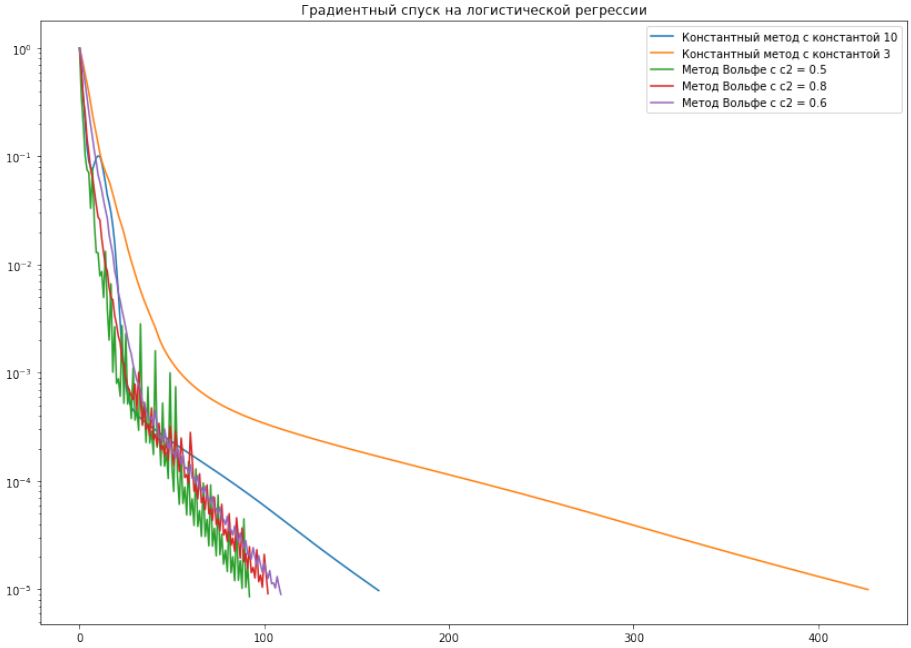
По графику видно, что, в среднем, с увеличением числа обусловленности увеличивается количество итераций (зависимость, возможно, нелинейная). С увеличением размерности квадратичной задачи количество итераций также увеличивается, при этом повышается количество исходов, когда градиентный спуск не сошелся.

Эксперимент 4: Стратегия выбора длины шага в градиентном спуске

Сгенерируем случайную квадратичную функцию, сравним стратегии выбора длины шага с начальной точкой из единиц:



При поиске минимума квадратичной функции градиентным спуском метод Вольфе работает быстрее всех. Исходя из сравнения начальных точек, можно сказать, что для точек, близких к минимуму, алгоритм сходится быстрее, что логично. Теперь проведём эксперимент с регрессией:

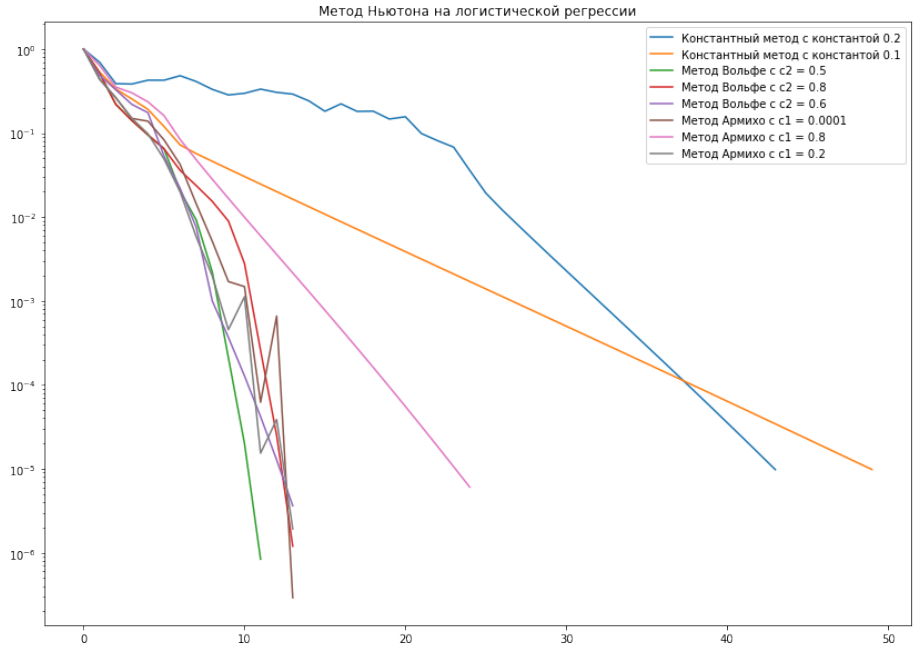


Метод Вольфе снова оказался быстрее всех, хотя если подобрать хорошую константу, то константный метод может догнать по скорости сходимости. Метод Армихо в десятки раз медленнее сходится, как и до этого.

По итогам эксперимента можно сказать, что лучше использовать метод Вольфе, поскольку он выигрывает за счёт более быстрой сходимости.

Эксперимент 5: Стратегия выбора длины шага в методе Ньютона

Повторим эксперимент с регрессией, но теперь будем использовать метод Ньютона, а не градиентный спуск:



Здесь метод Вольфе показывает примерно одинаковые результаты с методом Армихо, а константу надо подбирать, чтобы она с ними сравнялась. Ещё было отмечено, что в методе Ньютона сходимость в 10 раз быстрее.

По итогам эксперимента можно сказать, что лучше использовать метод Вольфе, поскольку он выигрывает за счёт более быстрой сходимости.