Отчет по первому практическому заданию

Царькова Анастасия

Формулы для градиента и гессиана функции логистической регрессии:

$$f(x) = \frac{1}{m}\sum_{i=1}^{m}\ln(1+\exp(-b_i a_i^T x)) + \frac{\lambda}{2}\|x\|_2^2 = \frac{1}{m}\ln(1+\exp(-b*Ax)) + \frac{\lambda}{2}x^T x$$

$$\nabla f(x) = -\frac{1}{m}A^T\left(b*\left(\frac{1}{1-\exp(b*Ax)}\right)\right) + \lambda x = -\frac{1}{m}A^T\left(b*Expit(-b*Ax)\right) + \lambda x$$

$$\nabla^2 f(x) = \frac{1}{m}A^T Diag\left(\left(1-\left(\frac{1}{1-\exp(b*Ax)}\right)\right)*\left(\frac{1}{1-\exp(b*Ax)}\right)\right)A + \lambda I =$$

$$= \frac{1}{m}A^T Diag\left(1-Expit(b*Ax)\right)*\left(Expit(-b*Ax)\right)A + \lambda I$$

(Далее все комментарии к графикам над ними>)

Зависимость поведения метода от обусловленности функции.

В данном эксперименте предлагается проанализировать траекторию градиентного спуска в зависимости от:

1. Обусловленности функции
2. Выбора начальной точки
3. Стратегии выбора шага

Сравнивая графики с линиями уровня функции и траекториями методов.

Зададим объекты на которых будем сравнивать поведения методов:

1. Функции – двумерные квадратичные функции с d равным нулю(для простоты) $f(x) = x^T Ax + bx = x^T Ax$.

$$M_1 = \begin{pmatrix}1 & 0\\ 0 & 2\end{pmatrix}\ M_2 = \begin{pmatrix}1 & 0\\ 0 & 60\end{pmatrix}$$

2. Начальные точки

$$x_1 = \begin{pmatrix}1\\ 1\end{pmatrix}\ x_2 = \begin{pmatrix}20\\ 30\end{pmatrix}$$

3. Стратегии выбора шага: Armijo, Wolfe, Constant

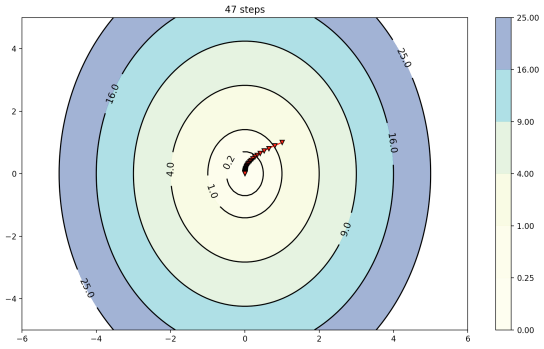
Результаты эксперимента:

- Constant

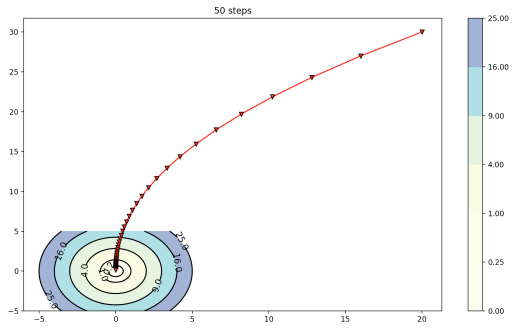
Видно что метод работает не очень хорошо. Делает слишком много шагов, но не слишком чувствителен к выбору начальной точки.

Более того крайне чувствителен к обусловленности функции, на M_2 метод возвращает $computational\_error$, поэтому я взяла вместо нее $M = \begin{pmatrix}1 & 0\\ 0 & 10\end{pmatrix}$ и все равно получила $iterations\_exceeded$, но полученные графики очень показательны. Кажется что метод вряд ли сойдется при увеличении числа итераций.

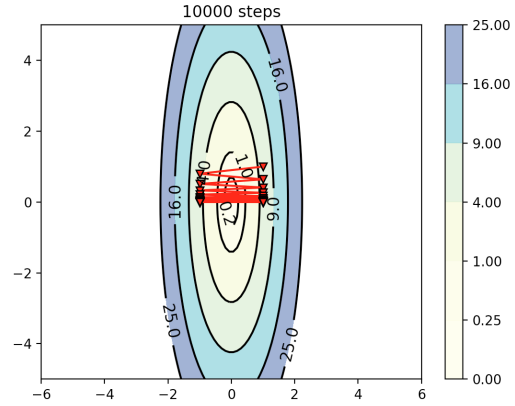
Данный метод очевидно является плохим, потому что если выбрать "плохой"шаг то можно не сойтись никогда.



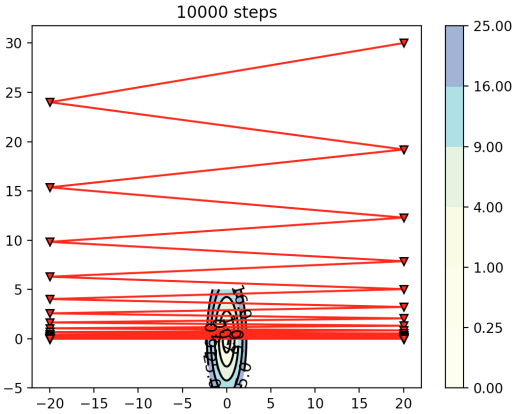
a)$M_1,\ x_1$



b)$M_1,\ x_2$



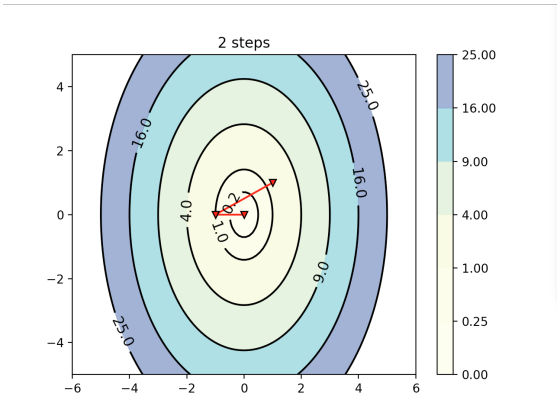
a)$M,\ x_1$



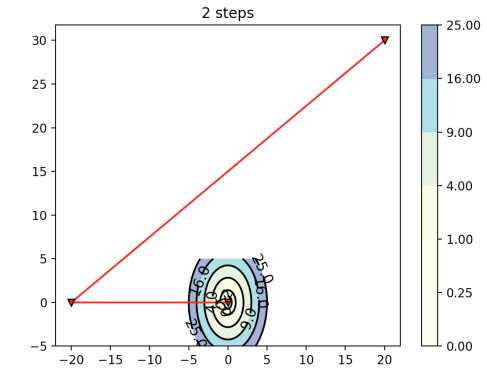
b)$M,\ x_2$

• Armijo

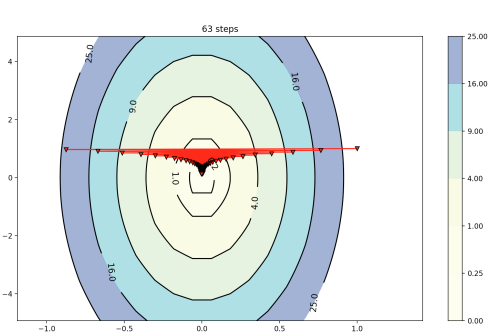
Данные метод явно лучше предыдущего. Видно что метод очень чувствителен к обусловленности функции. Так же есть небольшая зависимость о выбора начальной точки.



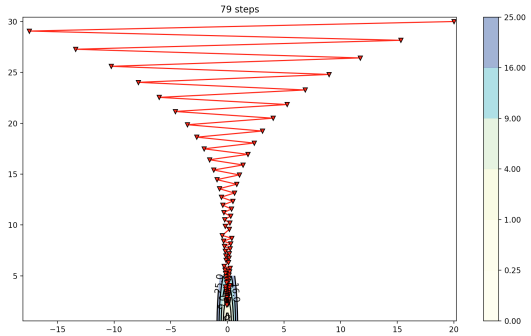
a)$M_1,\ x_1$



b)$M_1,\ x_2$



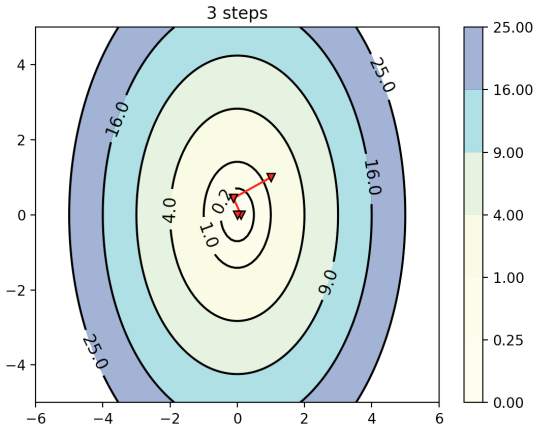
a)$M_2,\ x_1$



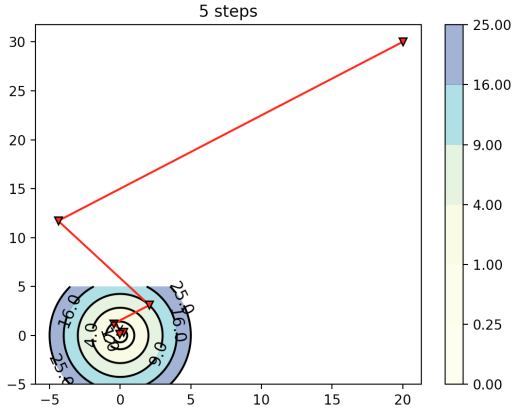
b)$M_2,\ x_2$

• Wolfe

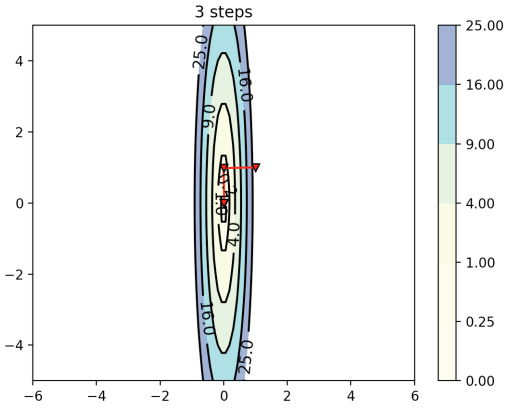
Данный метод показал себя лучше чем все предыдущие. Отсутствует зависимость скорости сходимости от обусловленности функции и (практически) от начальной точки.



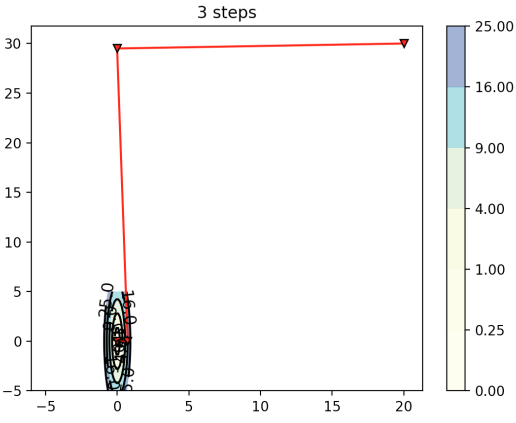
a)$M_1,\ x_1$



b)$M_1,\ x_2$



a)$M_2,\ x_1$



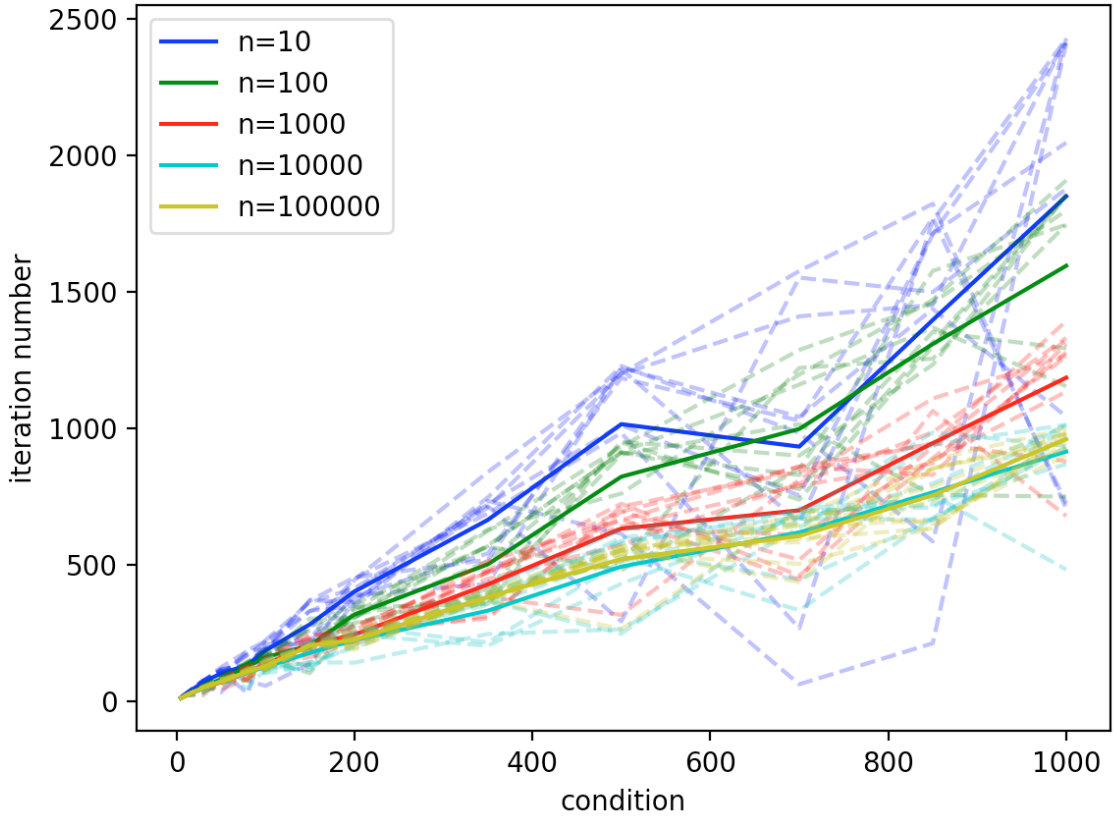
b)$M_2,\ x_2$

Зависимость числа итераций градиентного спуска от числа обусловленности и размерности пространства.

Для проведения данного эксперимента случайным образом были сгенерированы матрицы с различной обусловленностью из пространства различной размерности. На полученных данных был запущен градиентный спуск. По результатам работы градиентного спуска был построен график, по вертикальной оси отложено количество итераций совершенных алгоритмом, а по горизонтальной оси обусловленность задачи.

Из полученного графика видно что при увеличении обусловленности задачи растет количество итераций. При увеличении размерность пространства количество итераций уменьшается, причем чем больше n тем медленнее это уменьшение.

Так же можно заметить что чем больше n тем меньше разброс пунктирных линий то есть количество итераций растет стабильней.



Сравнение методов градиентного спуска и Ньютона на реальной задаче логистической регрессии.

Стоимость шага

На каждой итерации метода ньютона надо решить $\nabla^2 f(x_k)d_k = -\nabla f(x_k)$, данная операция при использовании разложения Холецкого занимает $O(n^3)$ и вычисление гессиана занимает $O(n^2 * k)$ получаем $O(n^2 * k + n^3)$ Для граиентного спуска $O(n * m)$.

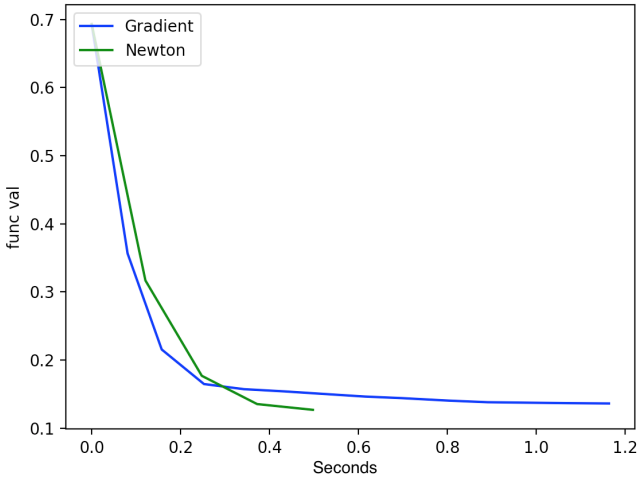
Память

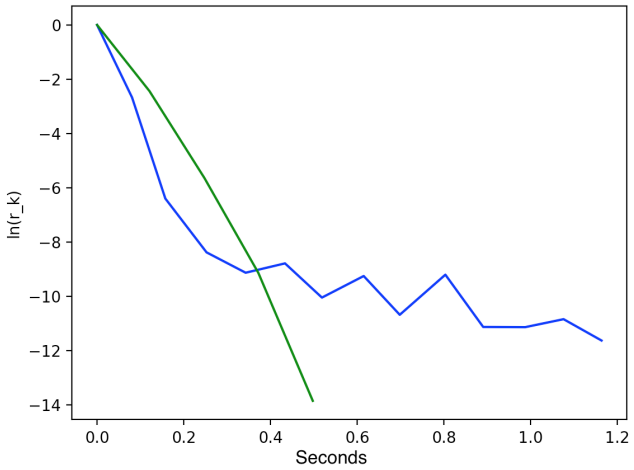
Метод ньютона использует $O(n^2)$ памяти, а градиентный спуск $O(n)$

В данном данном эксперименте сравнивается работа градиентного спуска с методом ньютона. Работа данных методов рассматривается на трех датасетах.

w8a

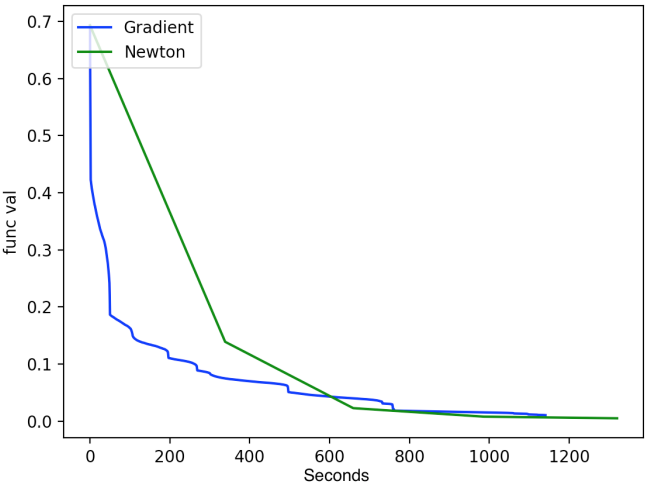
Метод Ньютона сошелся гораздо быстрее чем градиентный спуск, это произошло из за того что размер данной выборки достаточно мал и шаги метода Ньютона более агрессивны, хотя это медленнее с точки зрения вычислений.

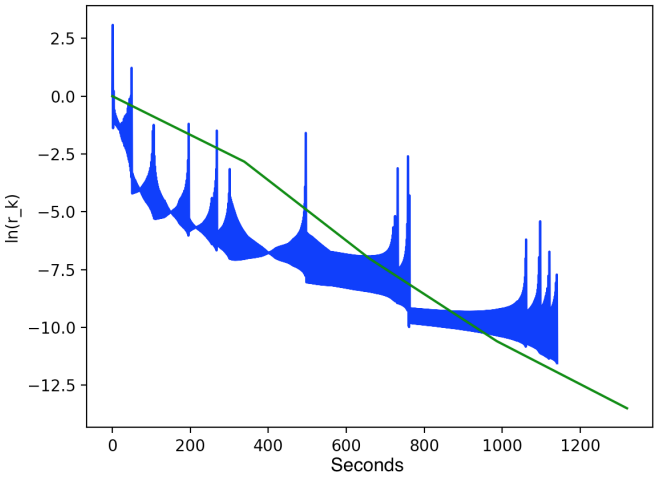




gisette-scale

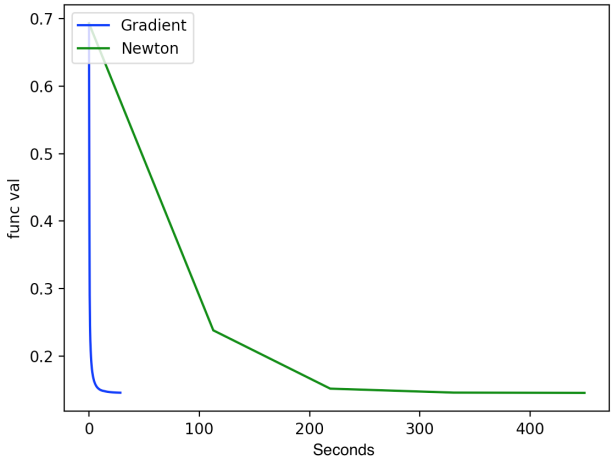
Результаты данного эксперименты отличаются от предыдущего. Данный датасет имел не так много объектов но при этом они были большого размера. Таким образом, метод Ньютона работал очень долго, и даже не смотря на то что он делает более "верные"шаги, время, которое требуется, чтобы сделать это слишком велико.

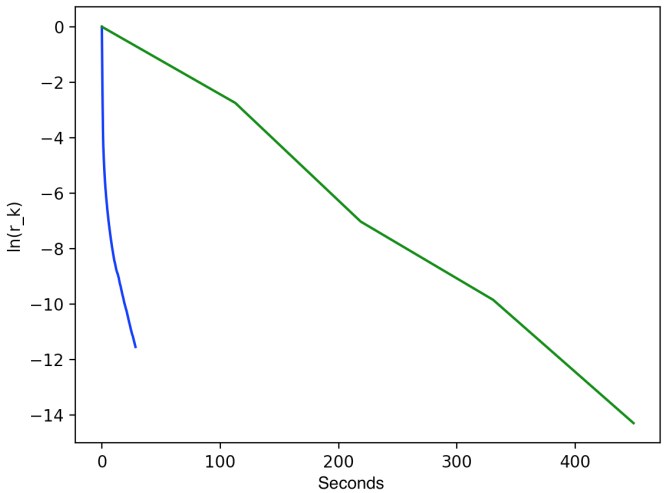




real-sim

На данной выборке метод ньютона показал себя так же плохо, он работал во много раз дольше градиентного спуска, который, в свою очередь, показал очень хорошие результаты сойдясь чуть ли не линейно.

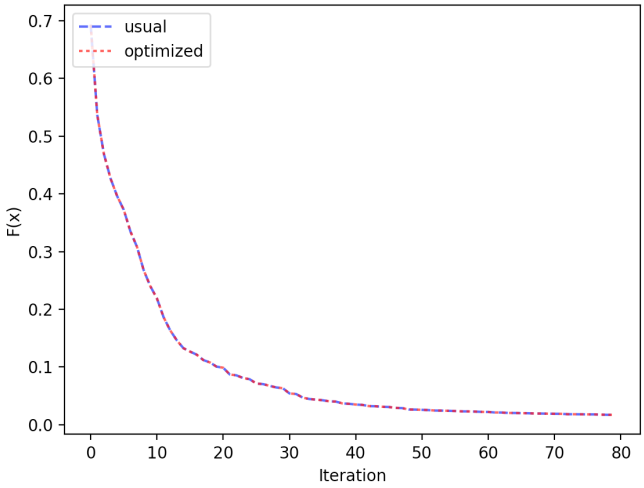




Данный эксперимент показал как плох метод ньютона на объектах из пространства с большой размерностью. Но в пространстве объектов с малой размерностью он работает гораздо быстрее чем градиентный спуск.

Оптимизация вычислений в градиентном спуске.

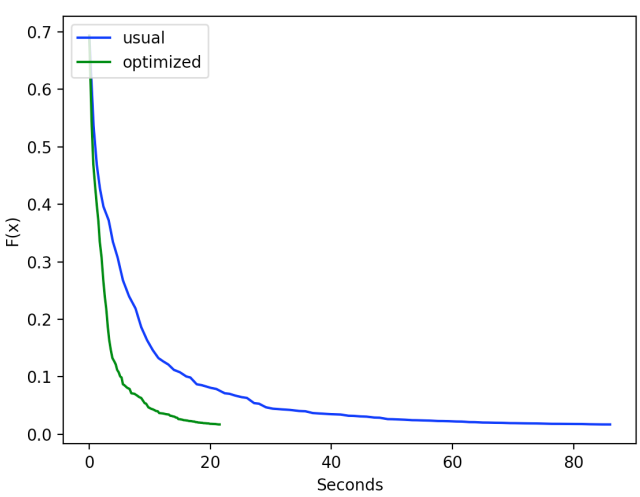
Траектории обоих методов на первом графике совпадают поскольку оптимизированный оракул не отличается от обычного в плане формул использованных для вычисления градиента или значения функции. Он просто умеет быстрее посчитать значения данных функций.

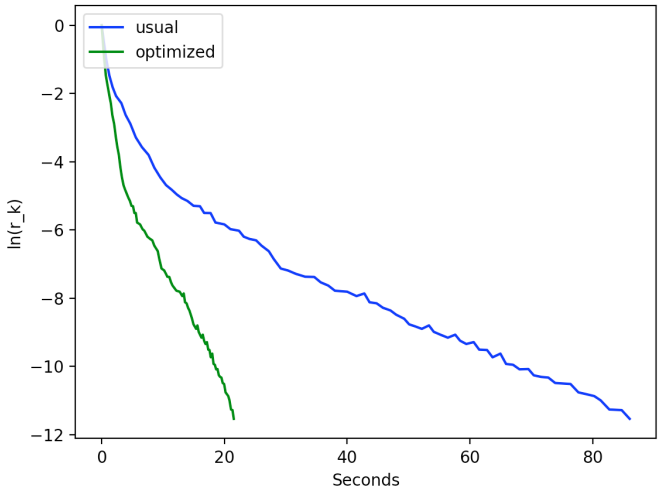


Для данного эксперимента был реализован оптимизированный оракул, который кеширует некоторые результаты сложных операций такие как матричные умножения.

Длее был проведено сравнение работы обычного оракула и оптимизированного с помощью градиентного спуска на случайно сгенерированных данных(m, n = 10000, 8000)

Из данных графиков видно что оптимизированный оракул действительно работает гораздо быстрее чем обычный.

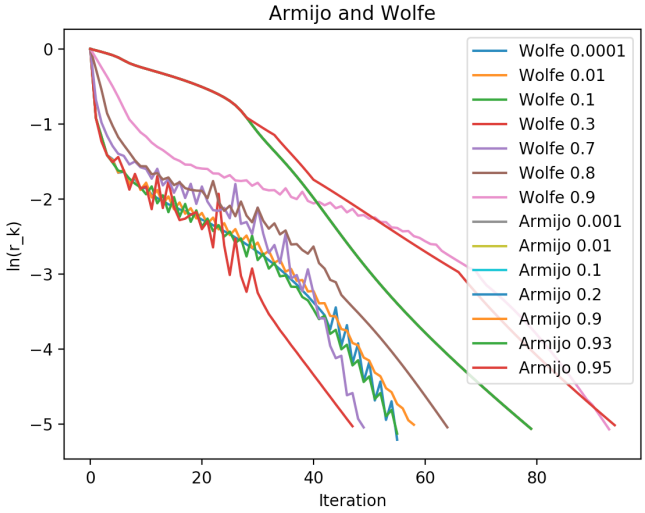


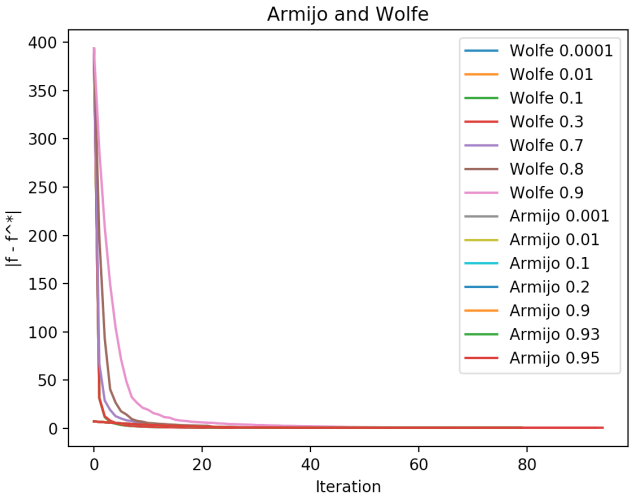


Стратегия выбора длины шага в градиентном спуске

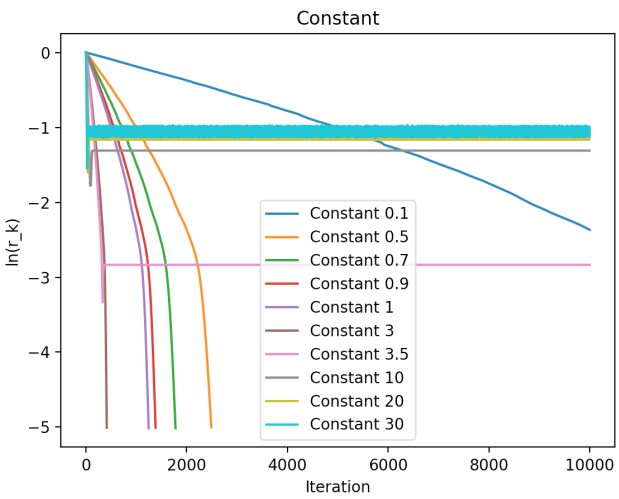
Для данного теста были сгенерированны случайные данные, далее на них были запущен градиент с различными стратегиями подбора шага.

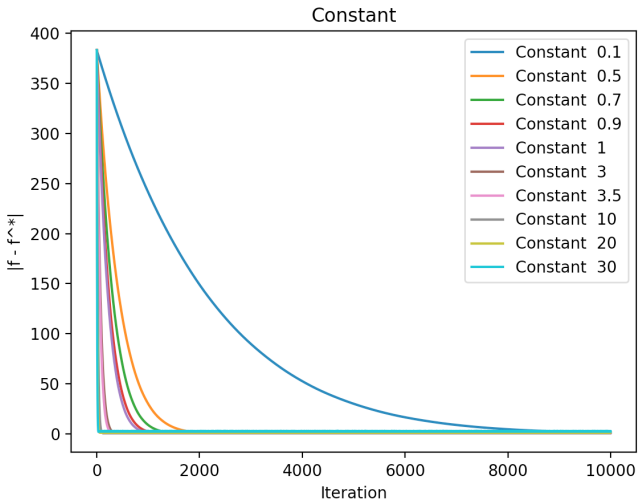
Между данными стратегиями нет большой разницы, только Wolfe на левом графике ведет себя более стабильно. Сходятся данные стратегии достаточно хорошо.





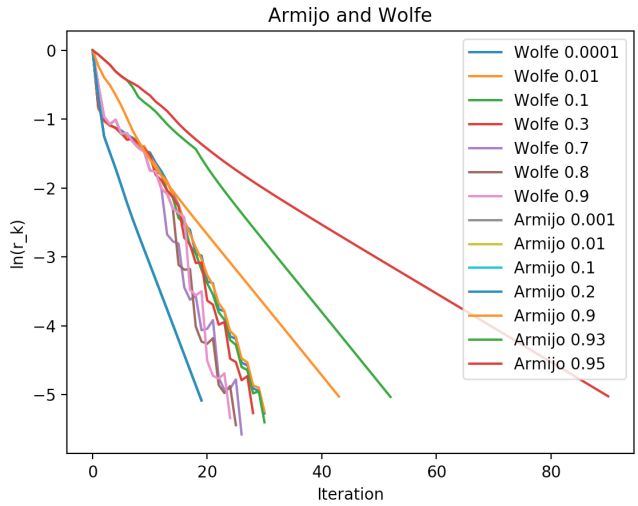
Данная стретегия показывает себя не очень хорошо, заметна явная зависимость от выбора параметра 'c', причем если его выбрать неудачно то можно вообще не сойтись.

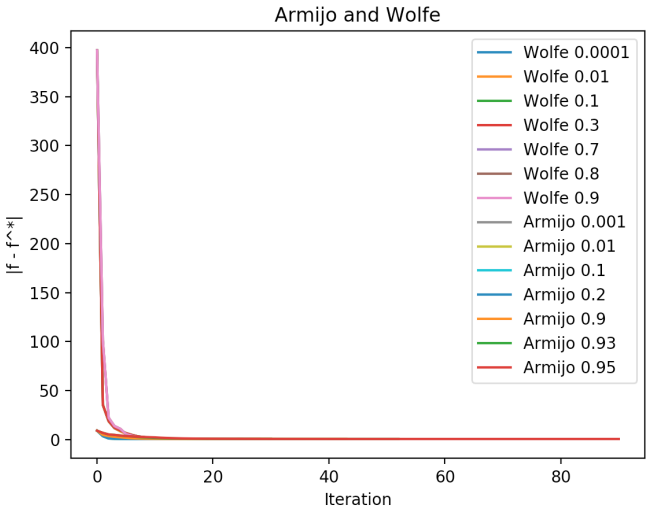




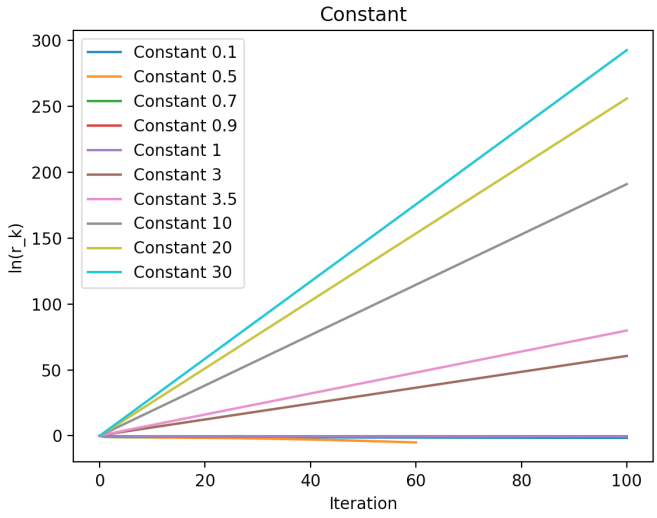
Стратегия выбора длины шага в методе ньютона

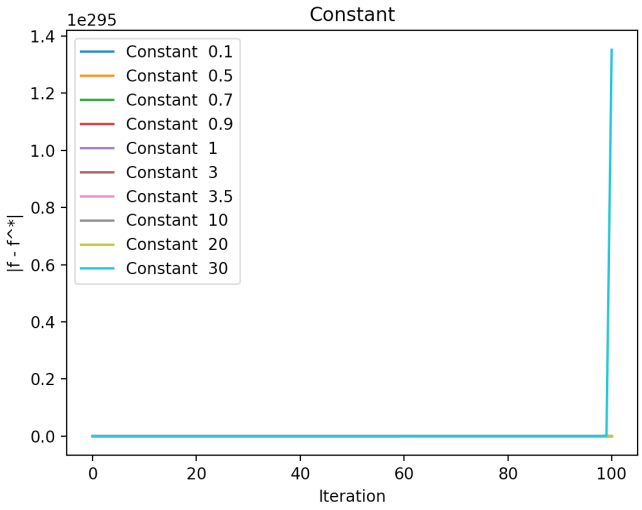
Про данные стратегии выводы такие же как и в предыдущем эксперименте.

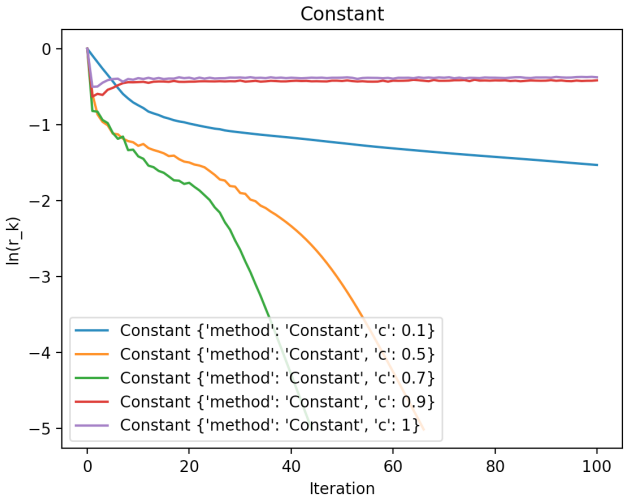


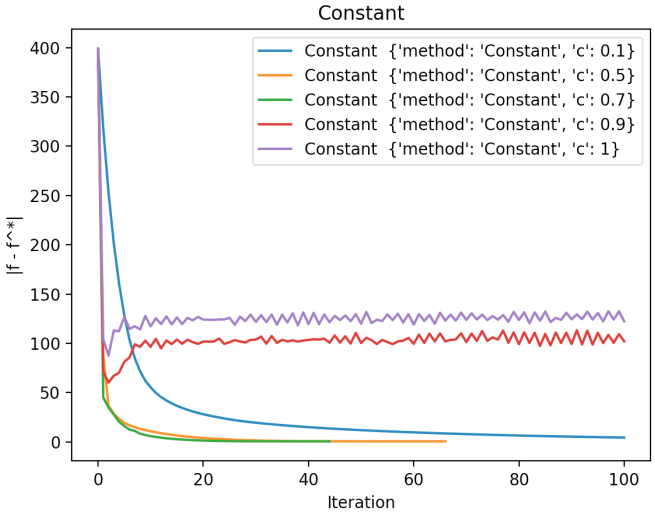


При $c > 1$ видно что алгоритм не сходится(расходится). При некоторых маленьких 'c' метод сходится нормально, но при некоторых может никогда не сойтись.


Constant
Constant 0.1
Constant 0.5
Constant 0.7
Constant 0.9
Constant 1
Constant 3
Constant 3.5
Constant 10
Constant 20
Constant 30
ln(r_k)
Iteration


Constant
1e295
Constant 0.1
Constant 0.5
Constant 0.7
Constant 0.9
Constant 1
Constant 3
Constant 3.5
Constant 10
Constant 20
Constant 30
|f - f^*|
Iteration


Constant
Constant {'method': 'Constant', 'c': 0.1}
Constant {'method': 'Constant', 'c': 0.5}
Constant {'method': 'Constant', 'c': 0.7}
Constant {'method': 'Constant', 'c': 0.9}
Constant {'method': 'Constant', 'c': 1}
ln(r_k)
Iteration


Constant
Constant {'method': 'Constant', 'c': 0.1}
Constant {'method': 'Constant', 'c': 0.5}
Constant {'method': 'Constant', 'c': 0.7}
Constant {'method': 'Constant', 'c': 0.9}
Constant {'method': 'Constant', 'c': 1}
|f - f^*|
Iteration