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

Методы оптимизации

Отчет по лабораторной работе №4

Работу выполнили:

Захаров Кирилл M32391

Мавлютов Эрвин M32391

Шилкин Артем M32391

Преподаватель:

Шохов Максим Евгеньевич

г. Санкт-Петербург

2023 г.

Оглавление

Постановка задач:

1. Изучение использования вариантов SGD (torch.optim) из PyTorch.
2. Исследование эффективности и сравнение с собственными реализациями.

Весь проект можно найти по ссылке.

Реализация методов из PyTorch

Мы использовали модель линейной регрессии, которая существует внутри PyTorch. Мы использовали различные методы для реализации SGD: torch.optim.SGD, torch.optim.Adagrad, torch.optim.RMSprop, torch.optim.Adam. Также мы использовали наши старые методы из второй лабораторной работы. Для чистоты эксперимента мы использовали те же самые константы для работы одинаковых методов, то есть метод, написанный нами и метод из PyTorch запускались с одинаковыми значениями констант.

Тестирование проводилось аналогично тестированию второй лабораторной работы. Мы брали 200 точек, брошенных вдоль прямой $y = kx + b$ со случайным параметром k и b. И точки выбирались при фиксированном x таким образом, чтобы $|kx + b - y| \leq \delta$, где $\delta = 10$.

Мы исследовали следующие величины при фиксированном числе epoch и batch: надежность (вероятность того,что наше решение отличается от оптимального не более чем на константу), среднее отклонение от оптимального решения, время работы и среднее потребление памяти.

Теперь перейдем к графику наших решений:

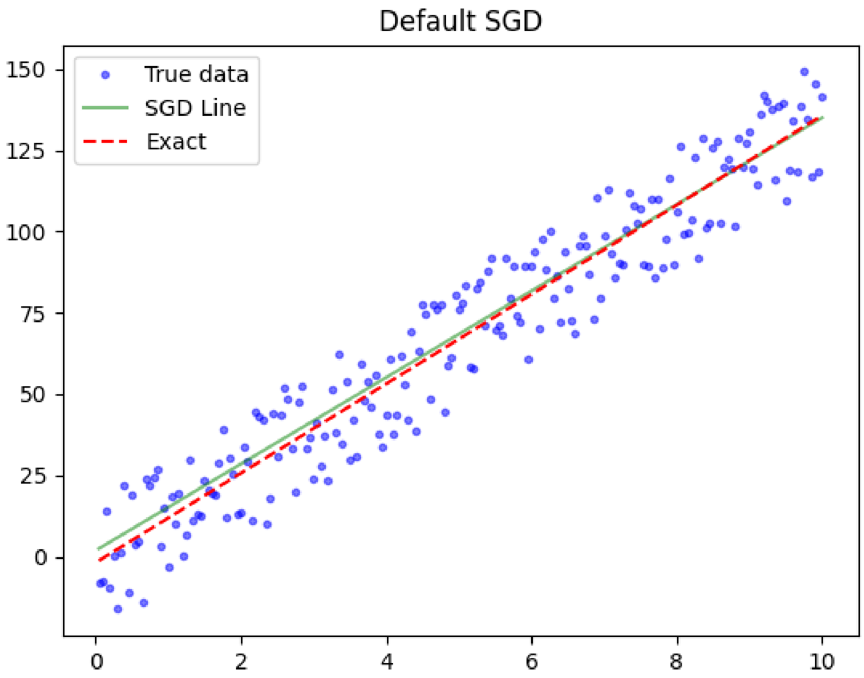

Рис. 1 – График аппроксимации множества точек линейной функцией с помощью библиотечного *DefaultSGD*.

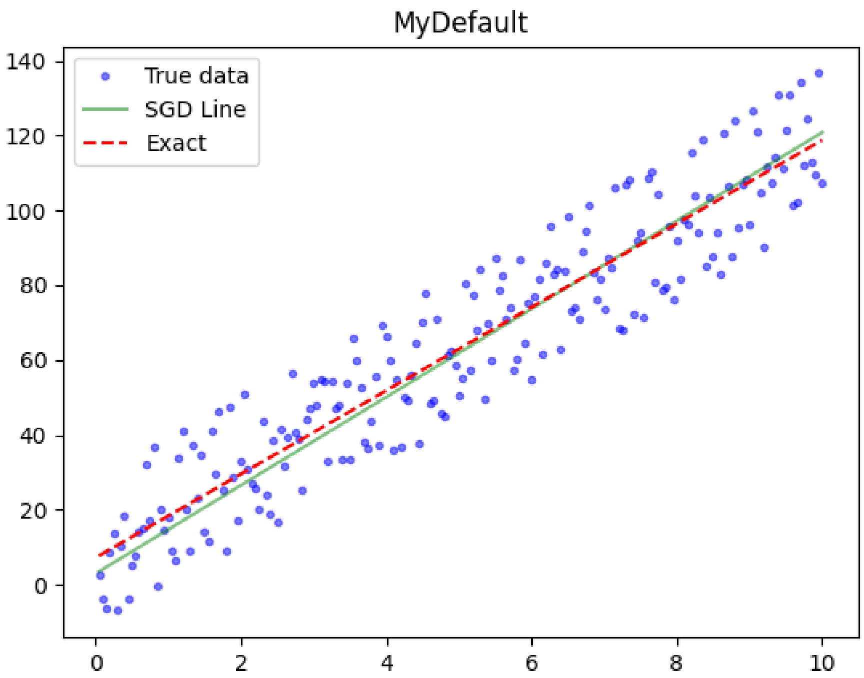

Рис. 2 – График аппроксимации множества точек линейной функцией с помощью реализованного *DefaultSGD*.

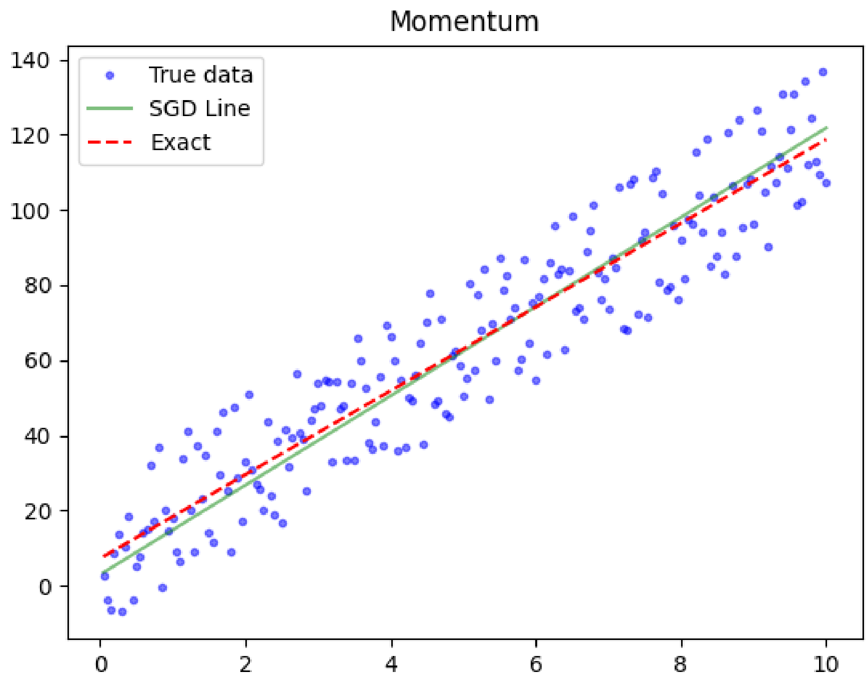

Рис. 3 – График аппроксимации множества точек линейной функцией с помощью библиотечного *Momentum.*

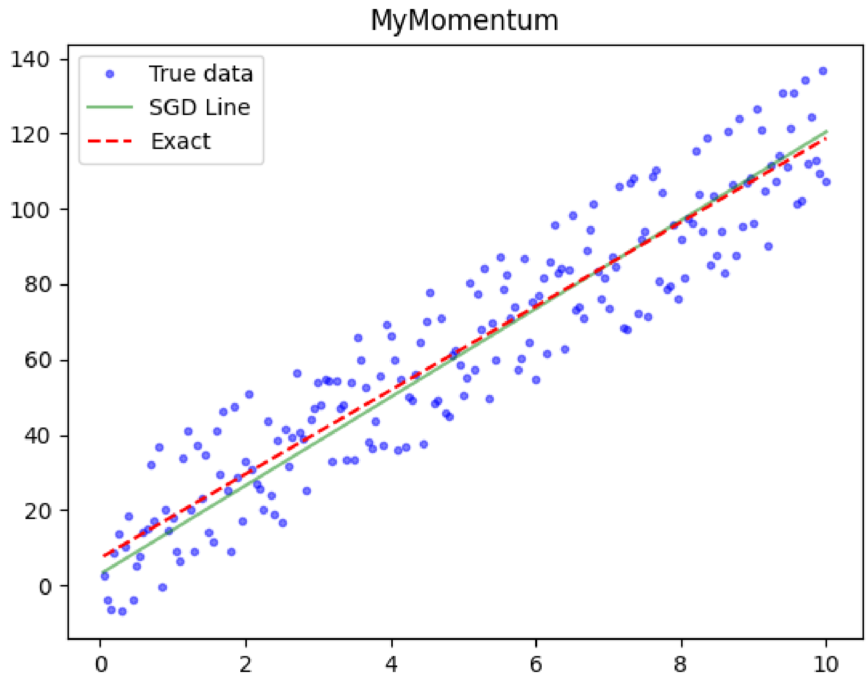

Рис. 4 – График аппроксимации множества точек линейной функцией с помощью реализованного *Momentum.*

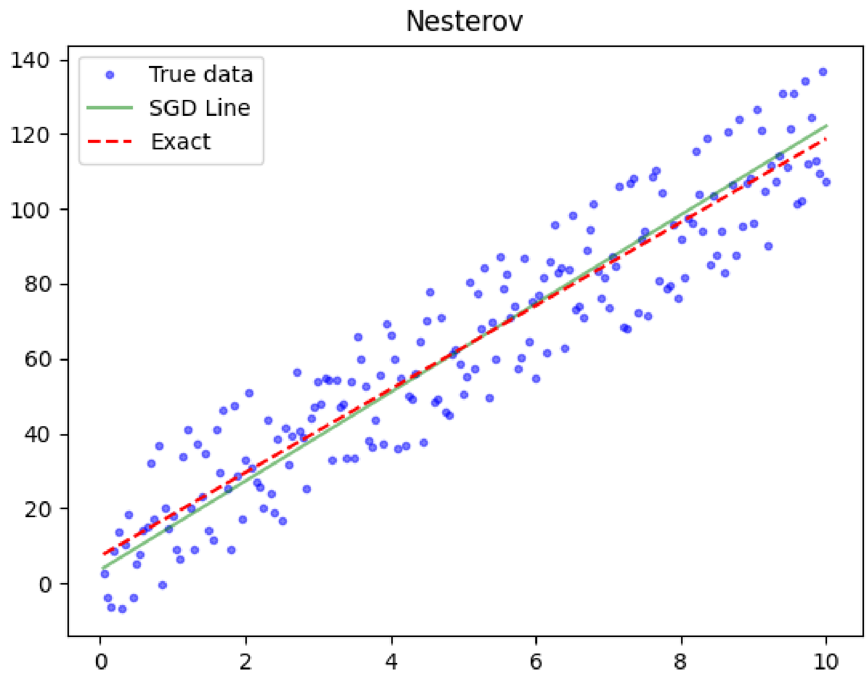

Рис. 5 – График аппроксимации множества точек линейной функцией с помощью библиотечного *Nesterov.*

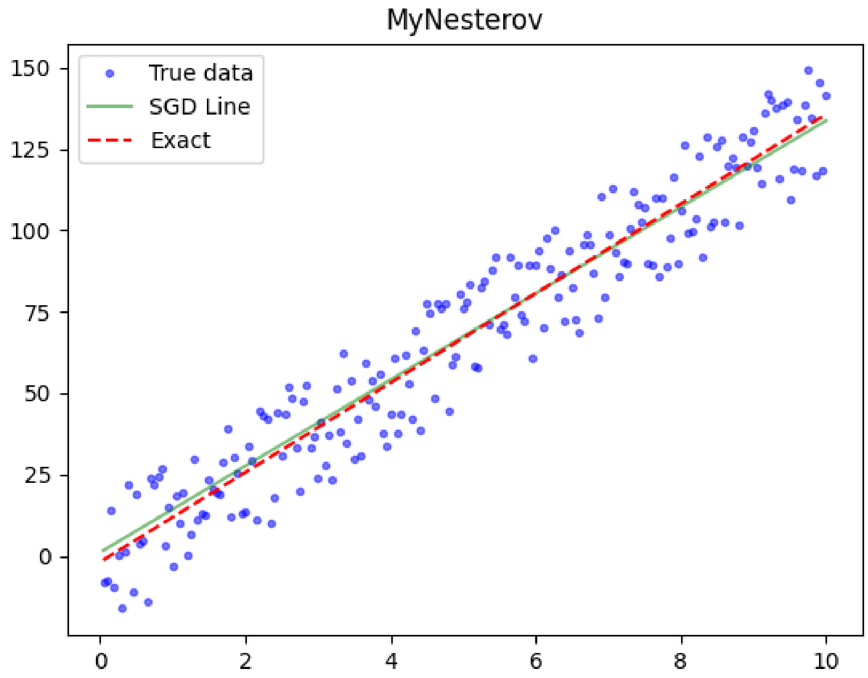

Рис. 6 – График аппроксимации множества точек линейной функцией с помощью реализованного *Nesterov.*

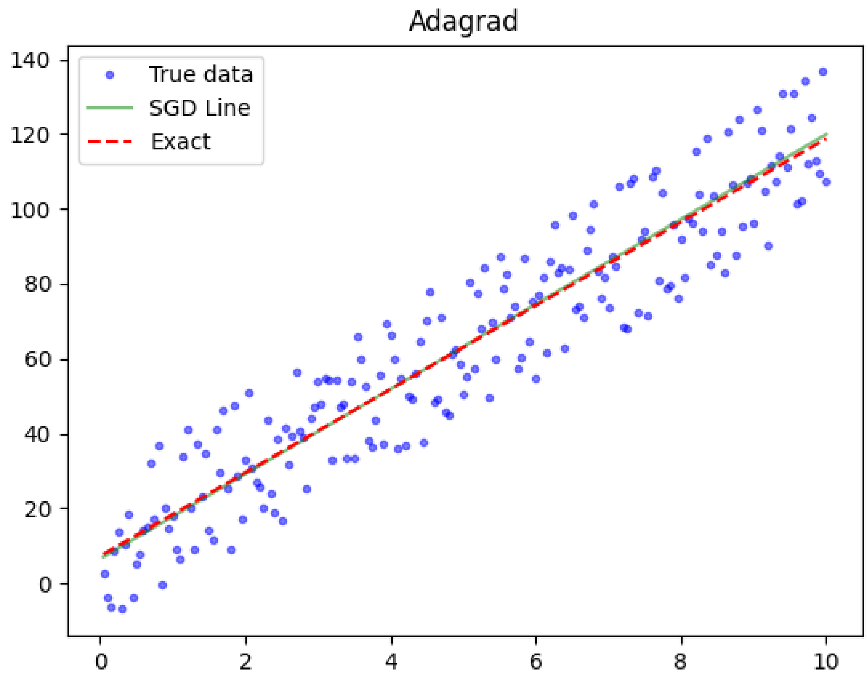

Рис. 7 – График аппроксимации множества точек линейной функцией с помощью библиотечного *AdaGrad.*

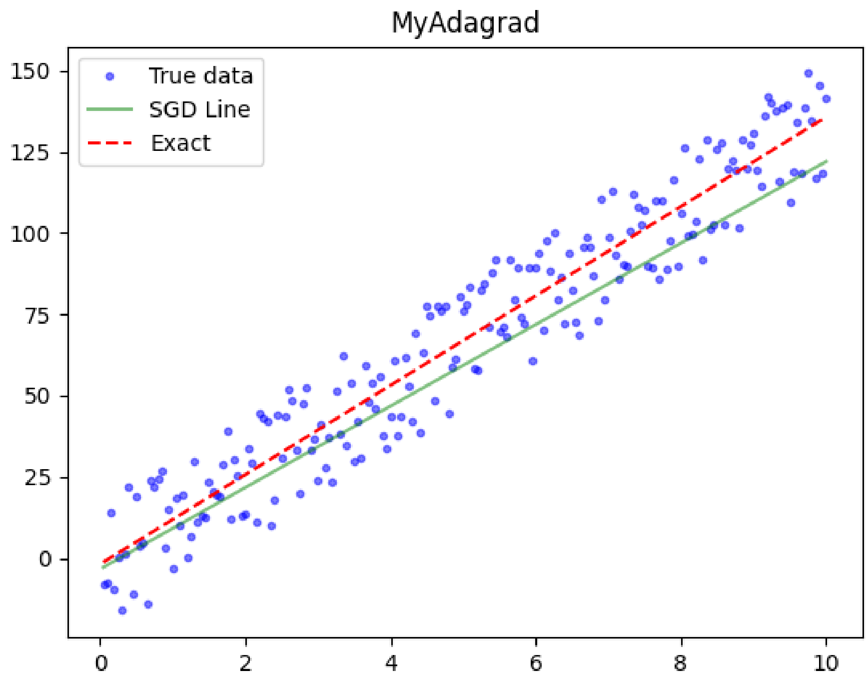

Рис. 8 – График аппроксимации множества точек линейной функцией с помощью реализованного *AdaGrad.*

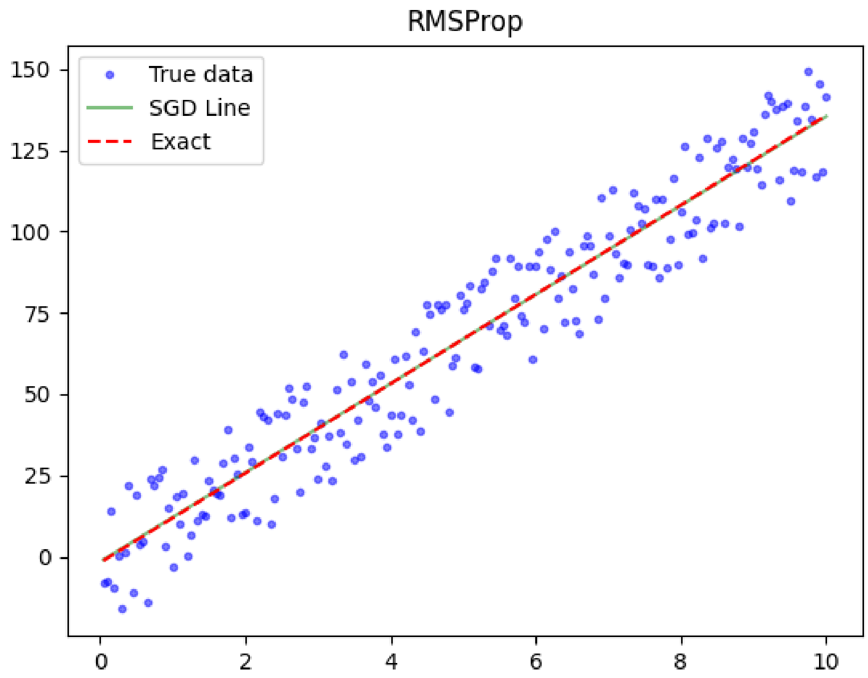

Рис. 9 – График аппроксимации множества точек линейной функцией с помощью библиотечного *RMSProp.*

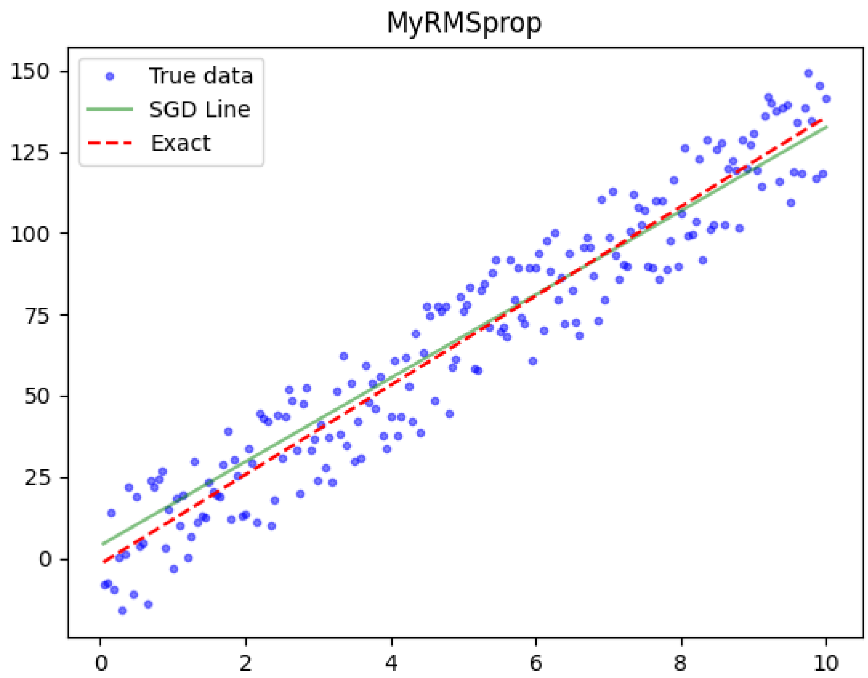

Рис. 10 – График аппроксимации множества точек линейной функцией с помощью реализованного *RMSProp.*

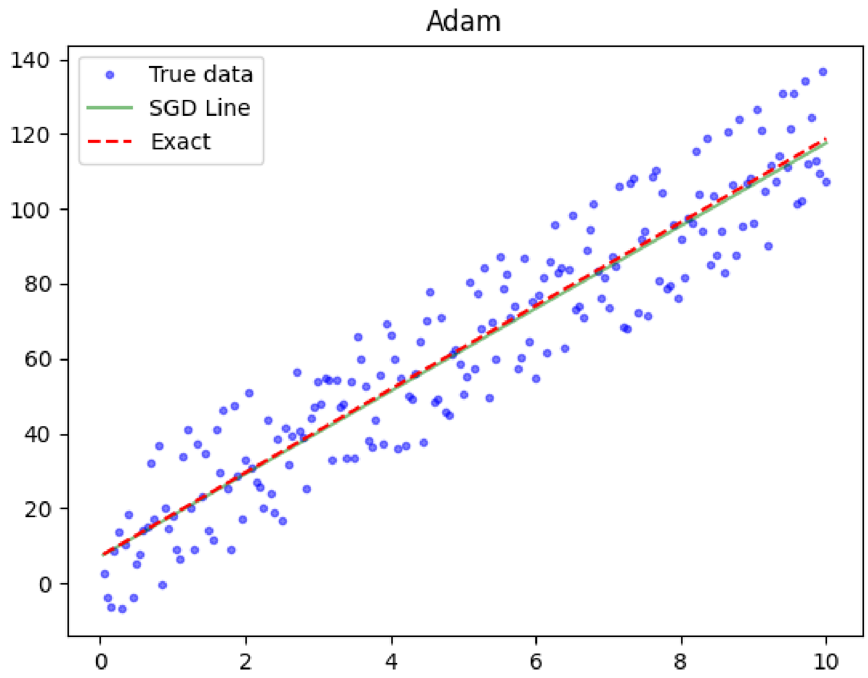

Рис. 11 – График аппроксимации множества точек линейной функцией с помощью библиотечного *Adam.*

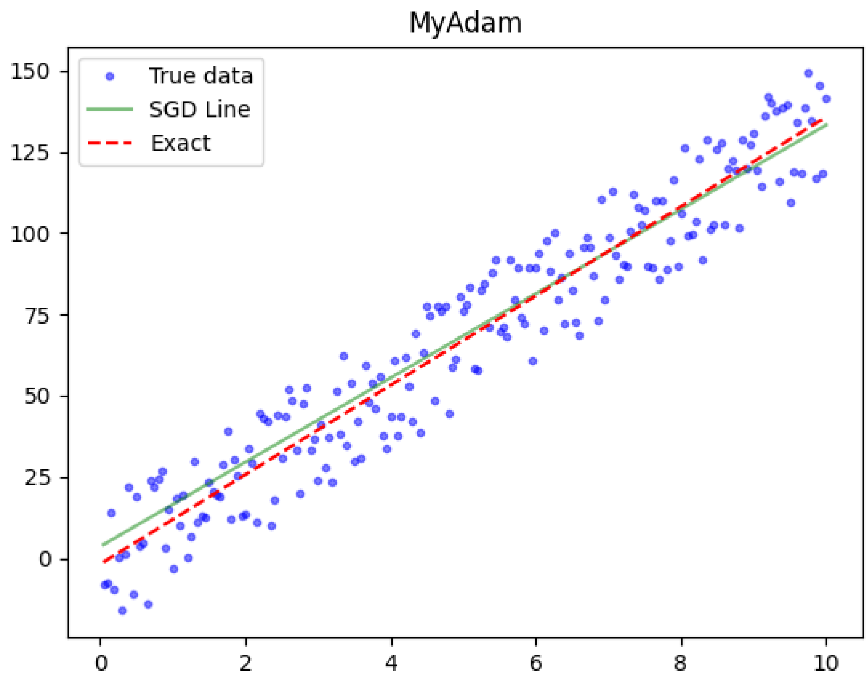

Рис. 12 – График аппроксимации множества точек линейной функцией с помощью реализованного *Adam.*

Теперь перейдем к статистике.

Метод	Вероятность сходимости	Значение минимиз. функции	Время (мс)	Память (Кб)
epoch=10				
Default SGD	0.85	12.4117	147.06	129.382
Momentum	0.95	8.33605	152.171	130.366
Nesterov	0.95	5.33252	158.93	131.257
Adagrad	0.65	20.6071	182.896	132.392
RMSProp	1,00	1.01595	172.223	133.22
Adam	1,00	0.447137	216.51	134.824
MyDefault	0.95	6.16304	55.7918	136.803
MyMomentum	0.95	6.09014	57.9984	137.133
MyNesterov	0.95	6.09925	59.0685	137.536
MyAdagrad	0.75	45089,00	58.3309	137.865
MyRMSprop	0.8	13.6804	61.7731	138.192
MyAdam	0.85	12.359	77.4148	138.58
epoch=20				
Default SGD	0.8	10.4115	260.184	114.148
Momentum	0.9	5.22984	293.284	115.476
Nesterov	0.95	3.07645	295.952	115.885
Adagrad	0.7	12.3173	325.291	117.015
RMSProp	1	0.622606	324.241	117.523
Adam	1	0.248411	421.131	119.007
MyDefault	0.85	10.4256	103.474	120.93
MyMomentum	0.85	10.2492	108.207	121.293
MyNesterov	0.85	10.2535	116.807	121.664
MyAdagrad	0.8	13.5278	120.331	122.012
MyRMSprop	1	1.87912	121.254	122.34
MyAdam	1	1.21777	126.058	122.728

Таблица 1 — сравнение различных реализаций SGD.

Заметно, что наши алгоритмы работают быстрее, чем реализация PyTorch. Этому есть простое объяснение: для пересчета градиента мы использовали функцию вычисления градиента посчитанную в явном виде, в то время как PyTorch использует приближенную формулу через соседние точки. Можно также заметить, что наши реализации потрябляют большее количество памяти чем реализации на PyTorch, что связано с тем, что многие вещи у нас написаны на чистом python, в то время как PyTorch делает это внутри себя на C. Также у нас наблюдается более худшая сходимость, но при большем количестве epoch (20) данные расхождения можно уже трактовать как надежность того или иного алгоритмов. В целом тенденция сохранилась: чем сложнее алгоритм, тем выше его вероятность сходимости при фиксированном batch и тем выше его потребление памяти и более долгое время работы.

Сравнение рукописных методов минимизации с библиотечными реализациями

Сравним использование методов минимизации из 3-ей лабораторной с соответствующими функциями из пакета *scipy.optimize*.

Функция Розенброка

В качестве минимизируемой функции для начала возьмем функцию Розенброка:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2.$$

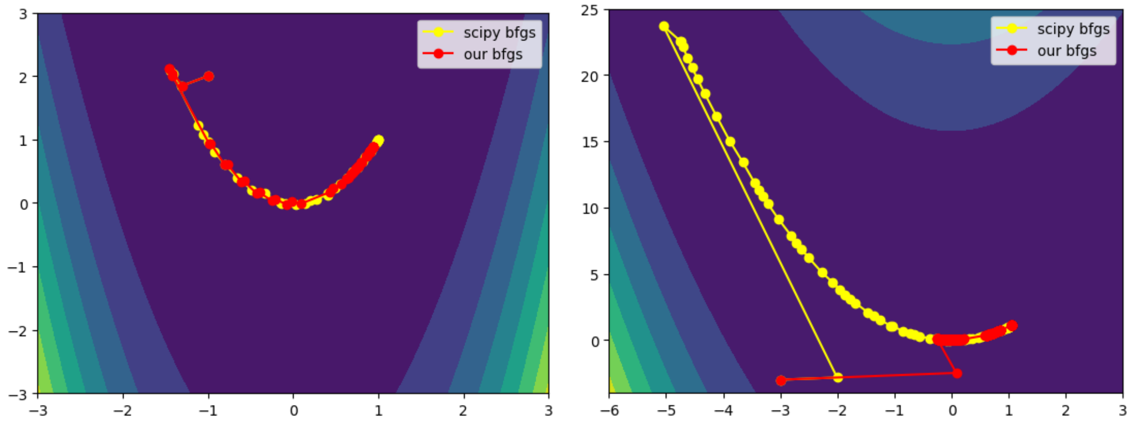

Рис. 13 – BFGS на функции Розенброка из точек (-1, 2) и (-3, 3).

Можно заметить, что шаги у нашего метода более редкие, что объясняется использованием сильных условий Вольфе в библиотечной реализации. В то время как наша реализация использует метод дихотомии.

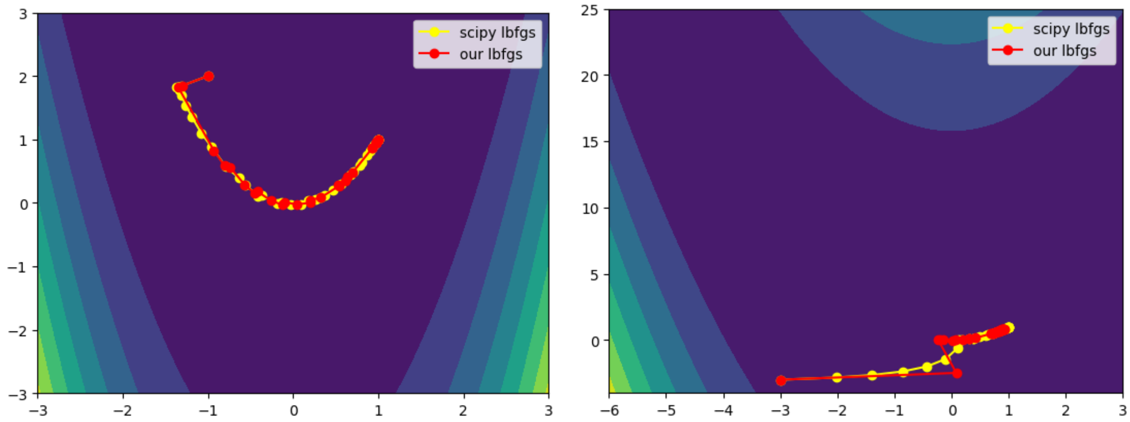

Рис. 14 – LBFGS на функции Розенброка из точек (-1, 2) и (-3, 3).

На втором запуске видно, что траектория LBFGS более плавная, что может быть связано с тем, что гессиан в начальном приближении считается равным единичной матрице и из-за чего поиск старается не перемещаться по сильно обусловленной переменной.

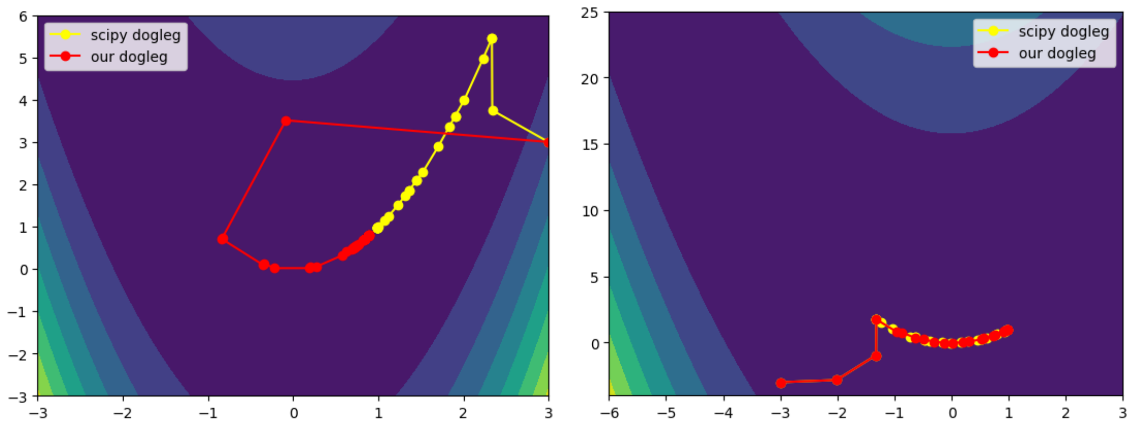

Рис. 15 – Dogleg на функции Розенброка из точек (-1, 2) и (-3, 3).

В первом случае видно различие в доверительных регионах двух методов, на втором первые шаги квадратичные модели давали одинаковые предсказания и траектории совпали.

Функция SinCos

В качестве второй функции рассмотрим:

$$sincos(x, y) = sin(x) + cos(y).$$

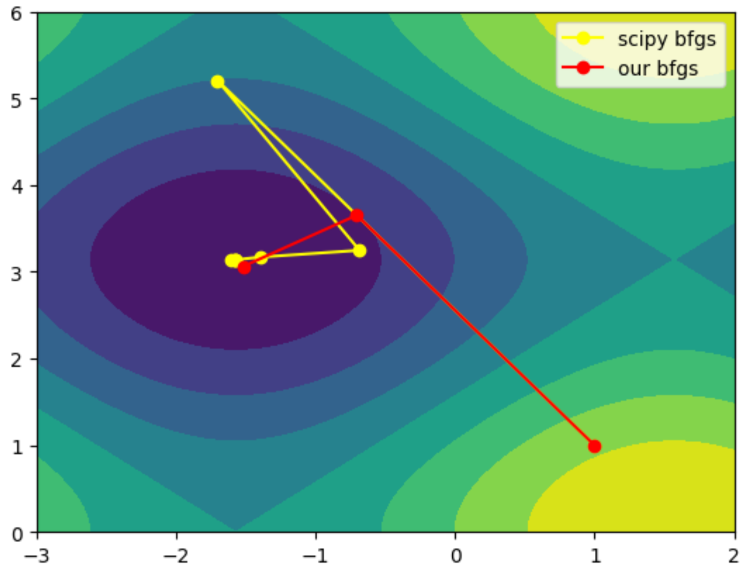

Рис. 16 – BFGS на функции SinCos.

Здесь вновь видно различие между линейным поиском с условиями Вольфе и без.

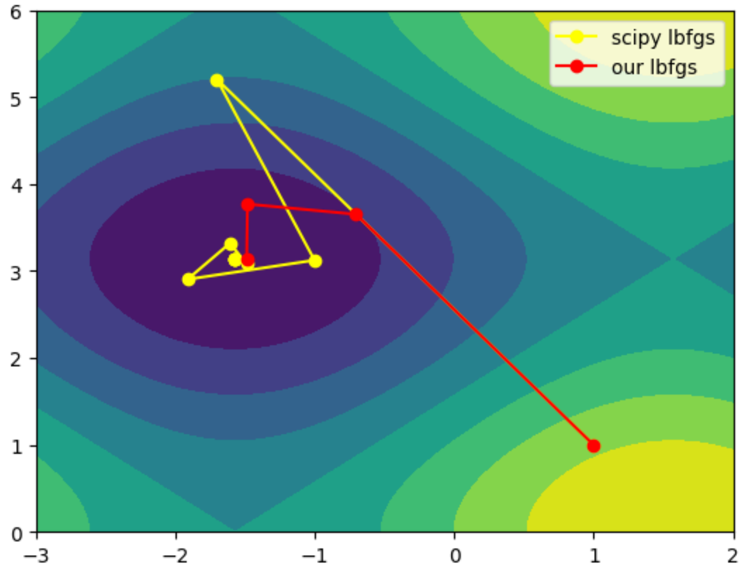

Рис. 17 – LBFGS на функции SinCos.

Если приглядеться, можно заметить, что здесь траектории более плавные в сравнении с BFGS.

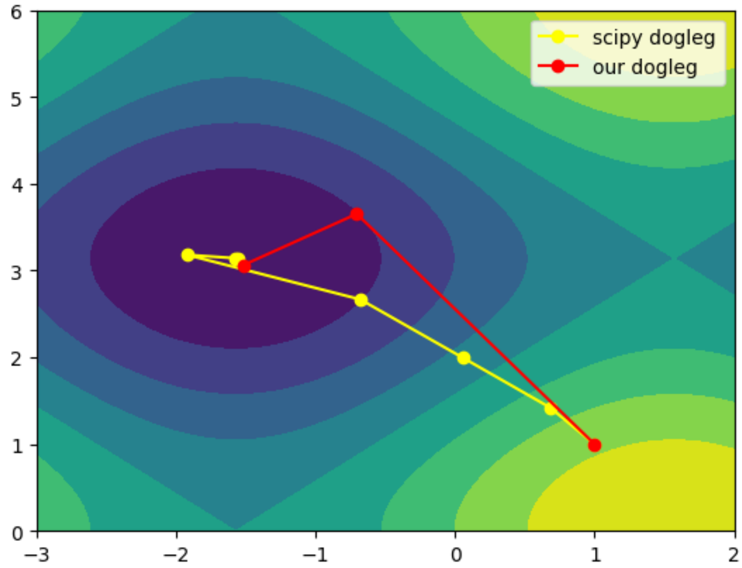

Рис. 18 – Dogleg на функции SinCos.

Опять же, можно наблюдать разницу в размере доверительных регионов.

Полиномы

Для тестирования функции *least_squares* мы будем решать задачу полиномиальной регрессии. Сравнение будет происходить с реализованным в лабораторной работе №3 методом *Dogleg*.

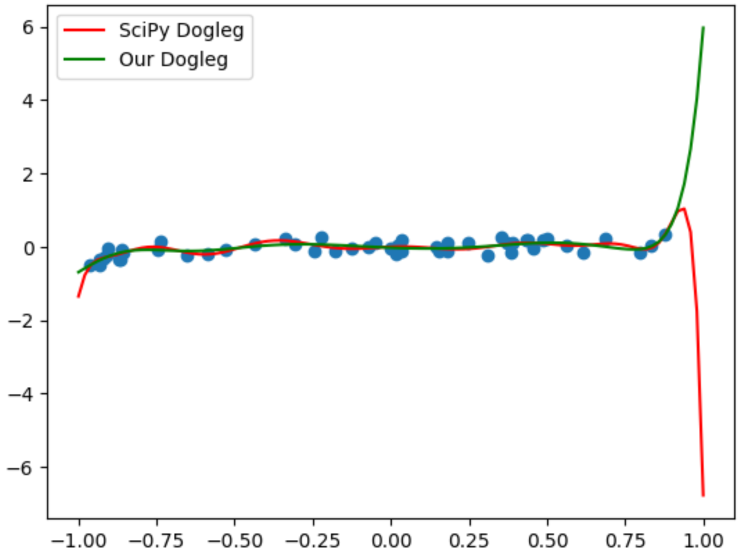

Рис. 19 – Результаты Dogleg'а для полинома 24-ой степени.

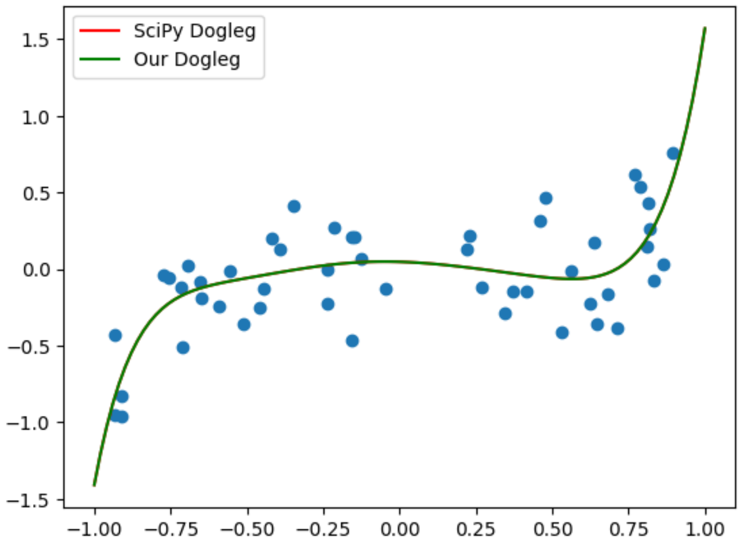

Рис. 20 – Результаты Dogleg'а для полинома 7-ой степени.

Методы сходились одинаково качественно, но, аналогично ситуации с PyTorch'ем, рукописные были быстрее ввиду отсутствия лишней "абстракции".

Сравнение использования PyTorch с другими методами подсчета градиента

Мы использовали два различных подхода для подсчета градиента в точке. Первый заключался в том, что мы считали значение функции в текущей точке и затем отходили на eps по каждой из координат. Еще мы использовали метод отхода на *eps* дважды по каждой из координат.

```
def f(x):
    return x[0] ** 4 + x[1] ** 2


def grad(func, point):
    x = torch.tensor(point, requires_grad=True)
    y = func(x)
    y.backward()
    return x.grad


def grad_simple(func, point, eps=1e-8):
    ret = np.zeros(len(point))
    func_val = func(point)
    for i in range(len(point)):
        point[i] += eps
        ret[i] = (func(point) - func_val)/eps
        point[i] -= eps
    return ret


def grad_advanced(func, point, eps=1e-8):
    ret = np.zeros(len(point))
    for i in range(len(point)):
        point[i] += eps
        ret[i] = func(point)
        point[i] -= 2 * eps
        ret[i] = (ret[i] - func(point))/(2 * eps)
        ret[i] += eps
    return ret


print(grad_simple(f, [1.0, 2.0]))
print(grad_advanced(f, [1.0, 2.0]))
print(grad(f, [1.0, 2.0]))
```

```
[4.00000006 3.99999998]
[4.00000003 4.00000003]
tensor([4., 4.])
```

Рис. 21 — пример работы нашего алгоритма и алгоритма PyTorch.

Задание границ изменения параметров у методов из SciPy.

При оптимизации функций можно задавать границы изменения каждого их параметров. Рассмотрим возможные эффекты на примере предыдущих функций.

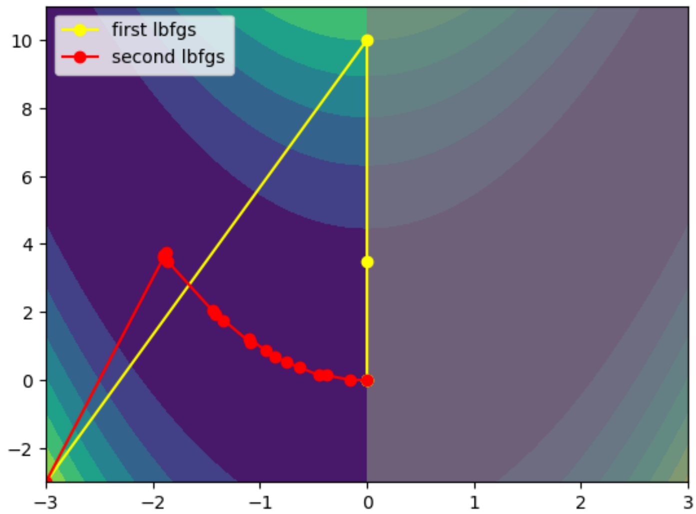

Рис. 22 – Неочевидное влияние ограничений на характер сходимости

Сразу хочется обратить внимание на данный случай. Для первой траектории ограничения на x были $(-\infty, 0)$, для второй – $(-10, 0)$. При этом шаги отличаются довольно сильно.

Для начала рассмотрим траектории сходимости различных методов без наложения ограничений.

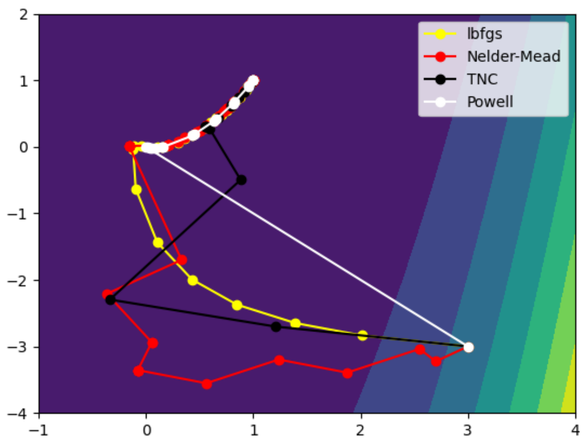

Рис. 23 – Сходимость методов без ограничений

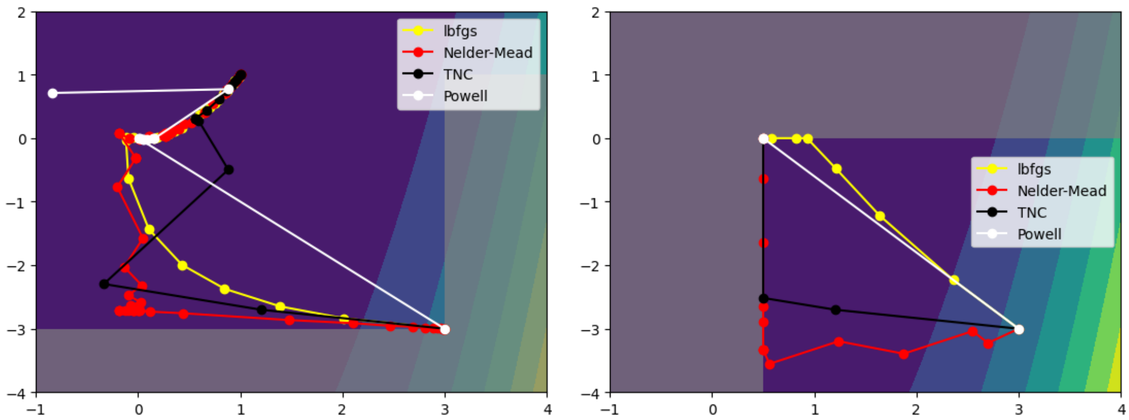

Рис. 24 – Сходимость методов с ограничениями

Видно, что процесс спуска отличается для различных ограничений. При этом в первом случае если не оставлять расстояния между начальной точкой и границей, то иногда методы завершались с ошибкой (в этом случае есть зазор в 0.01).

Еще с ограничениями мы можем запускать методы минимизации на функциях, не имеющих локального минимума, и не боятся переполнений. Например, рассмотрим выпуклую вверх функцию

$f(x, y) = -(x-1)^2 - (y-2)^2$

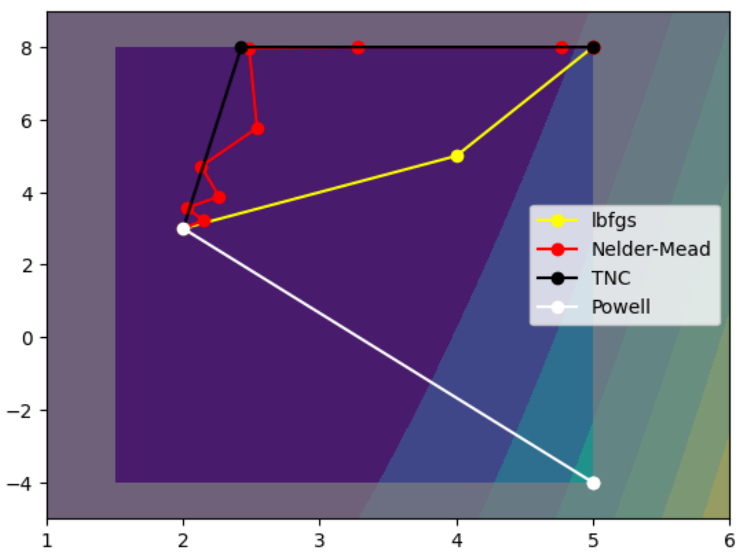

Рис. 25 – Запуск с ограничениями на функции, не имеющей локального минимума