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

Методы оптимизации

Отчет по лабораторной работе №4

Работу
выполняли:
Кольченко Антон M32371
Гайнанов Ильдар M32371
Муфтиев Руслан M32331

Санкт-Петербург
2023

Введение

Постановка задачи:

1. Изучить использование вариантов SGD (torch.optim) из PyTorch. Исследовать эффективность и сравнить с собственными реализациями из 2 работы.
2. Изучить использование готовых методов оптимизации из SciPy

   (scipy.optimize.minimize, scipy.optimize.least_squares)

   a) Исследовать эффективность и сравнить с собственными реализациями из 3 работы.

   b) Реализовать использование PyTorch для вычисления градиента и сравнить с другими подходами.

   c) Исследовать, как задание границ измерения параметров влияет на работу методов из SciPy.
3. Исследовать использование линейных и нелинейных ограничений при использовании scipy.optimize.minimize из SciPy. Рассмотреть случаи, когда минимум находится на границе заданной области и когда он расположен внутри.

Глава 1

Практическая часть

1.1 Сравнение реализаций SGD

Сравнение путей реализаций

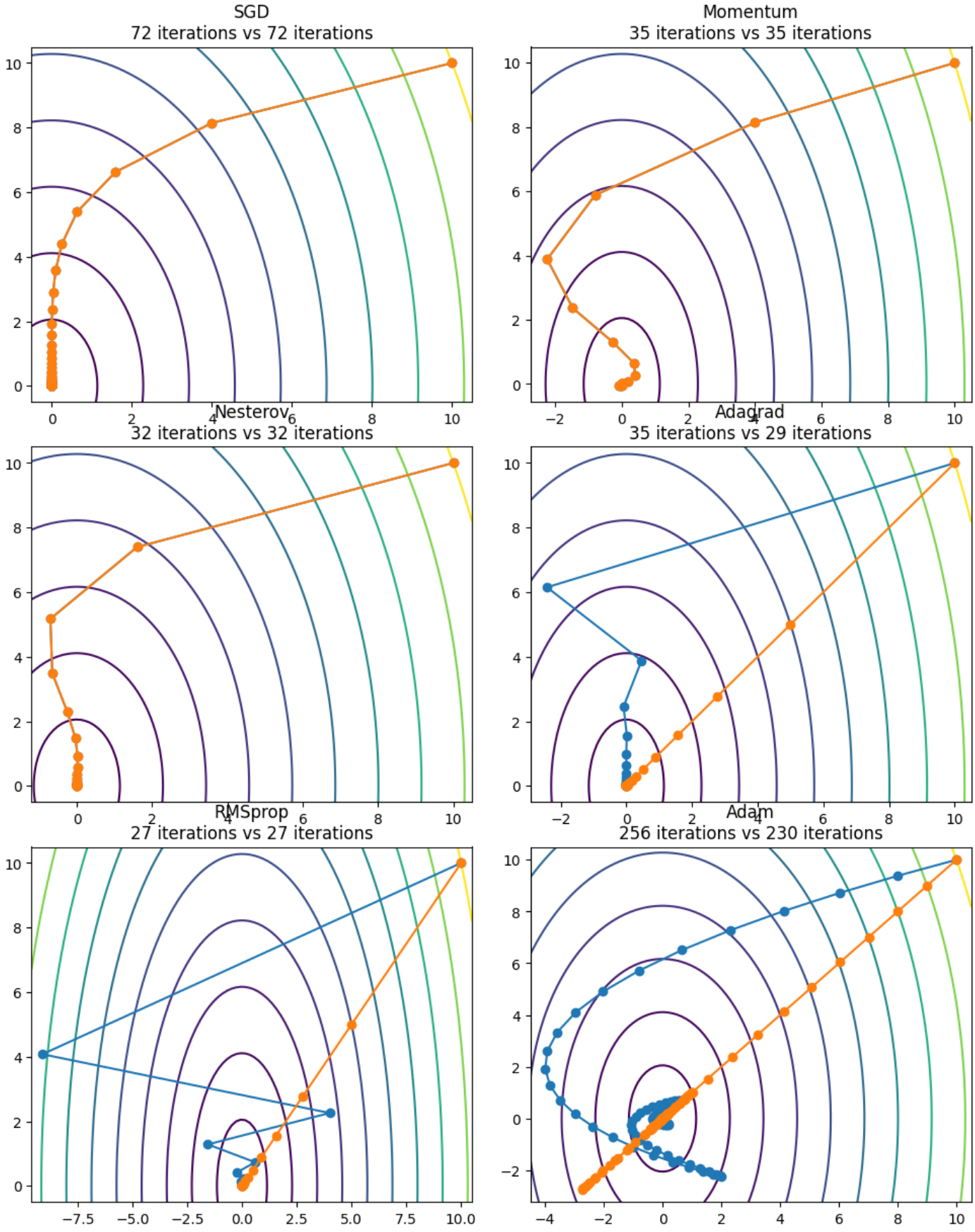


Рис. 1.1: Хорошо обусловленная функция. Синий - наша реализация, оранжевый - реализация из Pytorch

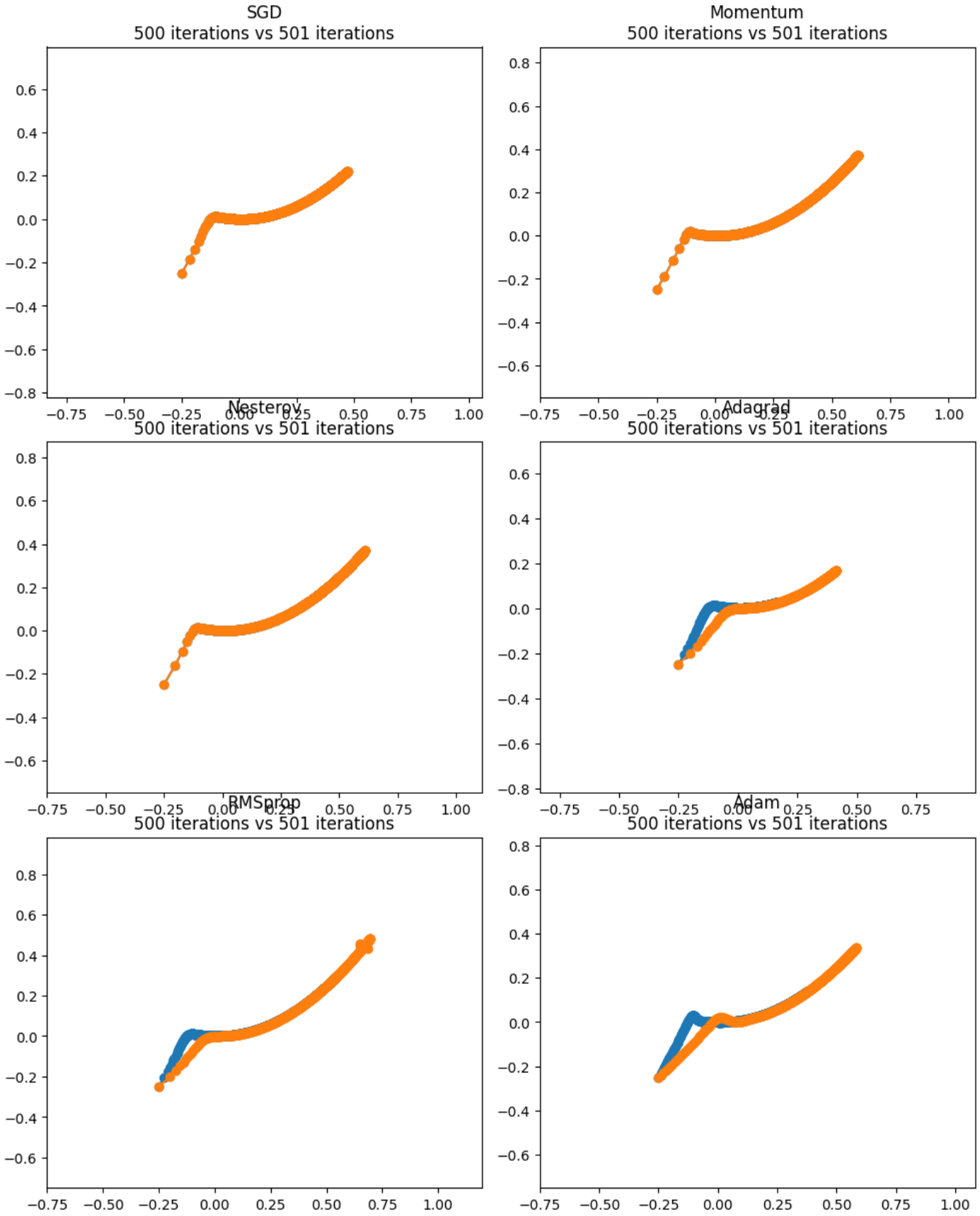


Рис. 1.2: Плохо обусловленная функция - функция Розенброка. Синий - наша реализация, оранжевый - реализация из Pytorch

Выводы

1. Путь реализаций стандартного стохастического градиентного спуска, модификаций Momentum и Nesterov при равных параметрах полностью совпадают. Необходимо будет присмотреться к другим характеристикам методов.

2. Пути остальных алгоритмов не совпадают. Однако, поиграв с параметрами, можно добиться схожих скоростей сходимости. Если посмотреть на реализации методов из PyTorch, можно заметить, что уже первый шаг имеет отличающееся направление. Делаем вывод, что PyTorch в этих методах вычисляет градиент иным способом.

3. Чудес не бывает - несмотря на отличия в пути, ни одна из реализаций алгоритмов не справляется с функцией Розенброка. Для нее все еще нужна смена алгоритма.

Сравнение производительности реализаций

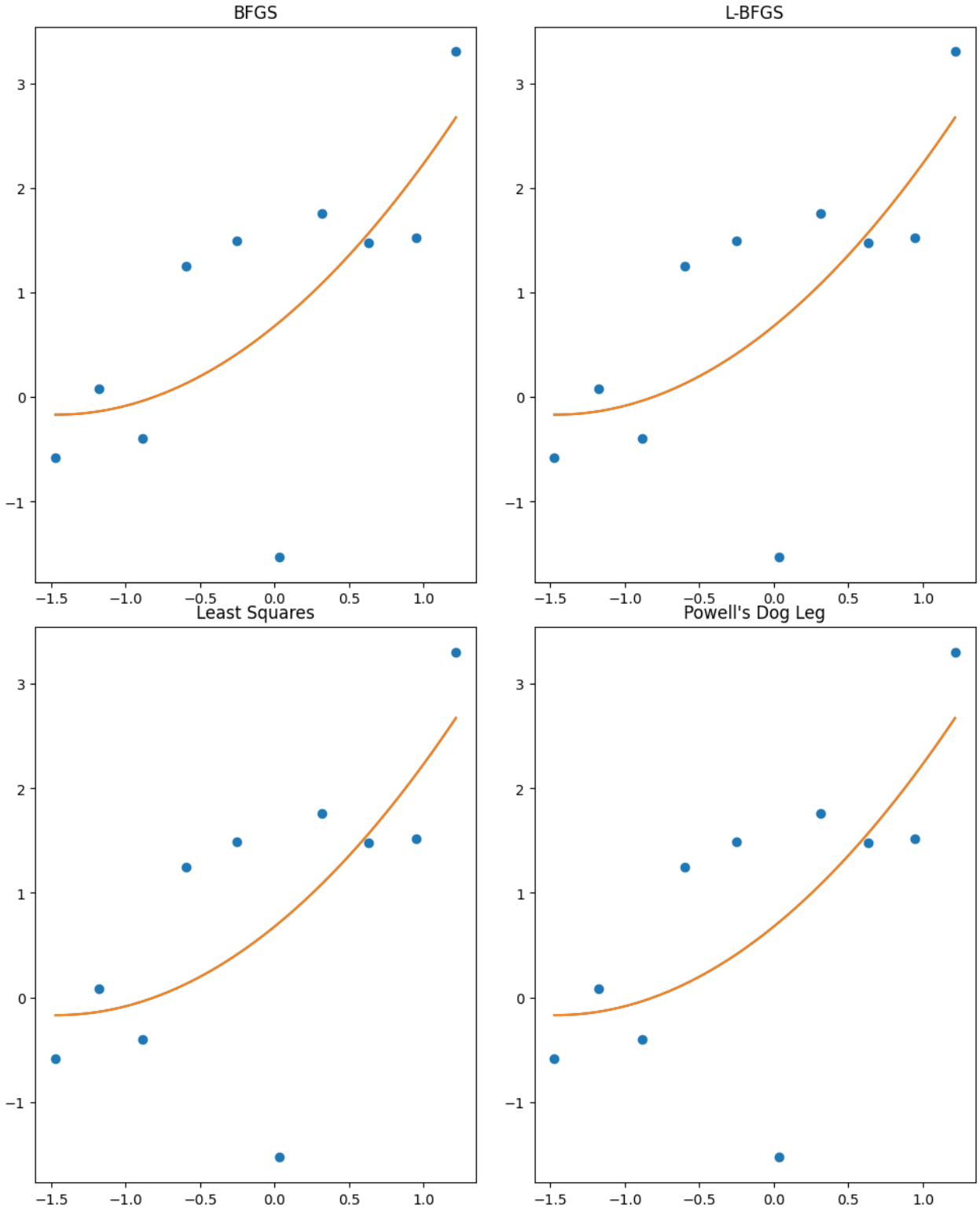
	time, s	mem usage
SGD	0.44	74070
PT SGD	0.053	15611
Momentum	0.203	44648
PT Momentum	0.026	8980
Nesterov	0.187	42526
PT Nesterov	0.025	8362
AdaGrad	0.189	44333
PT AdaGrad	0.023	8191
RMSProp	0.158	37673
PT RMSProp	0.022	7409
Adam	1.458	142363
PT Adam	0.204	45147

Таблица 1.1: Таблица расхода ресурсов на использование методов

Выводы

1. Неудивительно, что реализации на PyTorch оказались в разы экономнее, чем наши реализации. Наши реализации написаны полностью на Python, который не знаменит высокой производительностью, тогда как методы в PyTorch «под капотом» реализованы на C++.

1.2 Сравнение реализаций нелинейной регрессии



	time, s	mem usage
BFGS	1.32	270272
SP BFGS	0.012	14357
L-BFGS	2.238	119209
SP L-BFGS	0.008	23017
Gauss-Newton	0.016	104732
SP Gauss-Newton	0.034	11512
Dog Leg	0.226	156694
SP Dog Leg (dogbox)	0.032	12930

Таблица 1.2: Таблица расхода ресурсов на использование методов

Выводы

Выводы схожи с пунктом про градиентные спуски.

1. Наши реализации методов и реализации в SciPy выдают примерно одинаковый результат (разница в миллионные доли, на графике никак не будет видно).

2. Реализации в SciPy работают быстрее (за исключением метода наименьших квадратов) и занимают в разы меньше памяти.

1.3 Сравнение реализаций градиентов

Будем вычислять градиент методом двусторонней разности, с помощью библиотеки numdifftools и PyTorch.

	Δ_{manual}	time, s	mem usage
Manual	N/A	0.00022	6632
numdifftools	$3.75 * 10^{-9}$	0.02	22304
PyTorch	$3.75 * 10^{-9}$	0.0005	1281

Таблица 1.3: Таблица расхода ресурсов на использование методов

Выводы

Выводы схожи с пунктом про градиентные спуски.

1. Удивительно, но ручное вычисление градиента работает быстрее всего...
2. ...но при этом меньше всего памяти расходует вычисление через PyTorch.
3. Вероятно, Torch мог бы быть быстрее, но какое-то время тратится на конверсию np.ndarray -> torch.Tensor -> np.ndarray
4. В целом, вычисление градиента все равно занимает не очень много ресурсов, так что выбор алгоритма не будет кардинально менять время работы.

1.4 Зависимость результатов минимизации от границ переменной

Рассматриваем функцию $y = 3 + 3x + 3x^2$, ограничиваем переменную отрезком $[-a, a]$.

Gauss-Newton:

a	f_{loss}	time, s	mem usage
1	129.6	0.05	13205
3	0.016	0.1	13243
5	0.013	0.128	12844
10	0.013	0.1	12792
100	0.013	0.11	12840

Powell's Dog Leg:

a	f_{loss}	time, s	mem usage
1	129.6	0.012	12942
3	0.016	0.03	12978
5	0.013	0.07	13177
10	0.013	0.08	13132
100	0.013	0.07	13178

Таблица 1.4: Таблица расхода ресурсов на использование методов

Выводы

1. Выбор ограничений не оказывает значимого влияния на расход оперативной памяти.

2. Чем больше ограничение, тем медленнее работает метод - вплоть до какого-то значения. В какой-то момент оно перестает иметь смысл - шаг методов никогда не окажется вне слишком больших ограничений.

3. Однако если сделать ограничение слишком маленьким, то метод остановится до достижения минимума.

1.5 Зависимость результатов минимизации от границ переменной, линейные / нелинейные ограничения

$x_0 = (-30; 50)$

$f = f_{Rosenbrock}$:

constraint	x	itercount	time, s	mem usage	note
$\varnothing$	$\approx min$	172	0.458	14235	
$y \geqslant 0.5x$	$(0.054; 0.027)$	67	0.133	18553	1
$y \leqslant 2x$	$\approx min$	34	0.065	18055	
$y \leqslant x$	$\approx min$	36	0.083	21268	
$y \geqslant x$	$(5.55; 30.8)$	100	0.188	18585	1
$y = x$	$\approx min$	18	0.032	20119	
$B((1;1),1)$	$\approx min$	37	0.107	20995	
$B((0;1),1)$	$\approx min$	44	0.118	20552	
$B((0;0),1)$	$(0.79; 0.62)$	45	0.115	24083	2
$\overline{B((1;1),1)}$	$(0.43; 0.18)$	75	0.205	20400	2
$\overline{B((0;0),1)}$	$(-0.78; 0.62)$	37	0.18	21020	1

f - простая функция с точкой минимума $(1, 1)$:

constraint	x	itercount	time, s	mem usage	note
$\varnothing$	$\approx min$	30	0.129	14021	
$y \geqslant 0.5x$	$\approx min$	4	0.003	18537	
$y \leqslant 2x$	$\approx min$	22	0.061	18191	
$y \leqslant x$	$\approx min$	15	0.03	18191	
$y \geqslant x$	$\approx min$	26	0.05	18055	
$y = x$	$\approx min$	19	0.04	20981	
$B((1;1),1)$	$\approx min$	25	0.085	21050	
$B((0;1),1)$	$\approx min$	32	0.085	20552	
$B((0;0),1)$	$(0.81; 0.59)$	23	0.057	23481	2
$\overline{B((1;1),1)}$	$(1.71; 1.71)$	20	0.058	20866	2
$\overline{B((0;0),1)}$	$\approx min$	25	0.061	20368	

1 - минимум лежит в области ограничения, но само ограничение расположено так, что функция не может сойтись

2 - минимум не лежит в области

Таблица 1.5: Таблица расхода ресурсов на использование методов

Выводы

1. Линейные и нелинейные ограничения требуют заметно больше оперативной памяти (но все еще в пределах разумного)

2. Уменьшение области позволяет ускорить алгоритм (и с точки зрения количества итераций, и с точки зрения процессорного времени), при этом факт нахождения минимума на границе / внутри области не оказывает значимого влияния на скорость сходимости.

3. Важно держать в голове примерное направление сходимости функции. В противном случае можно "перегородить"самый кратчайший путь и замедлить сходимость / лишиться сходимости полностью.