Национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра Компьютерных технологий и программирования

Лабораторная работа №4

Выполнили:
Дроботов И.В. М 3232
Бандурин В.А. М 3232
Ларский Н.А. М 3232

Преподаватель:
Свинцов М.В.

Санкт-Петербург
2023

Содержание

1 Цель

Изучить, как применять варианты SGD (torch.optim) в библиотеке PyTorch. Планируется провести исследование и сравнить их эффективность с моими собственными реализациями из двух предыдущих проектов. Кроме того, необходимо изучить использование готовых методов оптимизации из библиотеки SciPy.

2 Решение

Мы сравниваем алгоритмы из библиотеки Pitch с нашими предыдущими реализациями. Анализируем графики сходимости методов, сравниваем время выполнения и количество итераций разных алгоритмов. Делаем краткий вывод в конце каждой задачи, подводя итоги исследования.

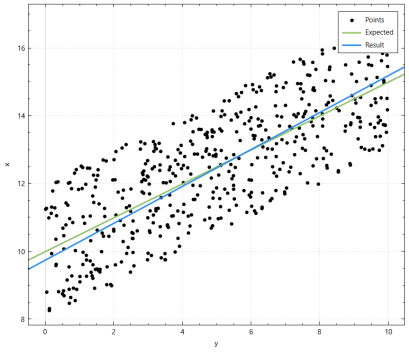
```python
import torch
import torch.optim as optim

def get_model(method, x, y, max_step):
    x = torch.tensor(x, dtype=torch.float32)
    y = torch.tensor(y, dtype=torch.float32)
    mdl = torch.nn.Linear(1, 1)
    cr = torch.nn.MSELoss()
    opt = method(mdl.parameters(), lr=1.e-2)
    sm_var = optim.lr_scheduler.StepLR(opt, step_size=1000, gamma=1.e-3)
    prv = float('inf')
    it = 0

    for epoch in range(max_step):
        it += 1
        opt.zero_grad()
        pred = mdl(x.view(-1, 1))
        diff = cr(pred, y)
        diff.backward()
        opt.step()
        sm_var.step()
        if diff.item() < prv:
            prv = diff.item()
        else:
            break

    return mdl, it

```

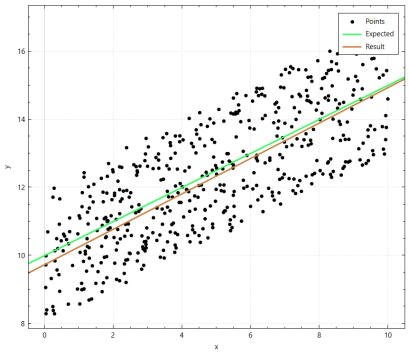
2.1 SDG

Кол-во точек	Шаги	Время	PyTorch Шаги	PyTorch Время
50	1358	52	665	102
100	4458	325	814	215
150	2640	10	631	245
200	3791	18	441	256
250	4521	1613	1165	196

P.S. время указано в миллисекундах.

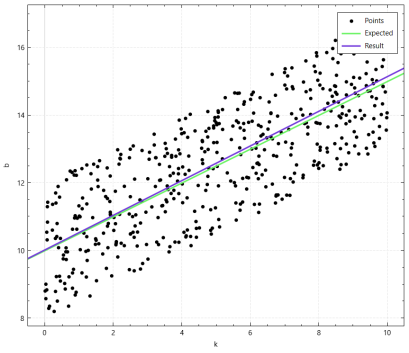
Рис. 1. $f(x) = \frac{1}{2}x + 10$

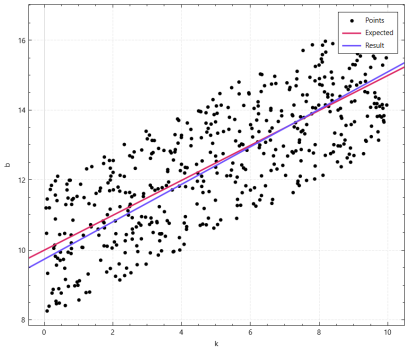




2.2 AdaGrad

Рис. 2. $f(x) = \frac{1}{2}x + 10$

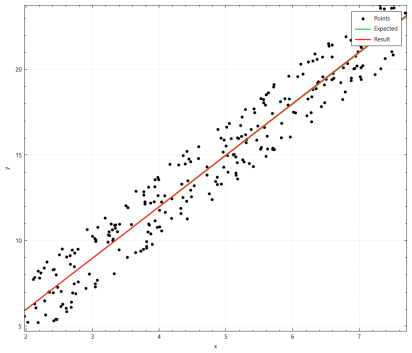


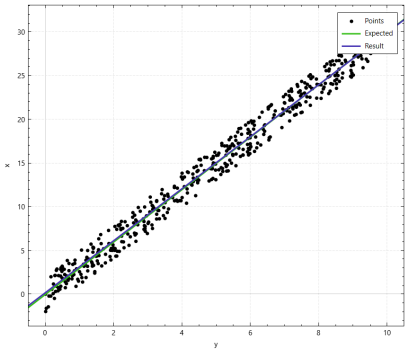


Кол-во точек	Шаги	Время	PyTorch Шаги	PyTorch Время
50	292	67	9	59
100	499	201	9	27
150	678	491	29	75
200	1015	823	18	46
250	2032	2066	13	41

2.3 RMSProp

Рис. 3. $f(x) = 3x$

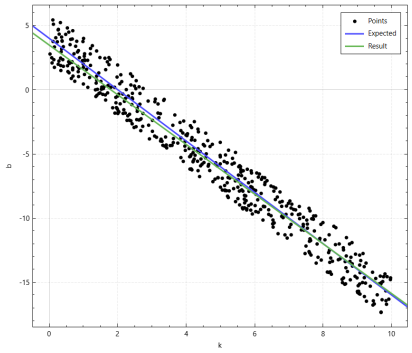


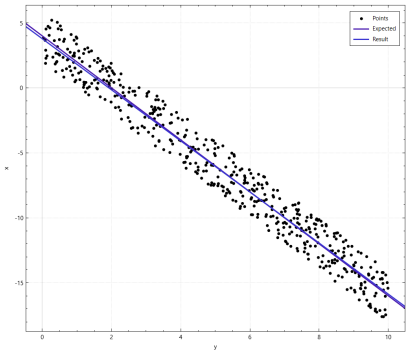


Кол-во точек	Шаги	Время	PyTorch Шаги	PyTorch Время
50	339	55	2267	2203
100	419	132	179	357
150	464	232	2267	5211
200	457	341	189	469
250	521	426	2588	5075

2.4 Adam

Рис. 4. $f(x) = -2x + 4$





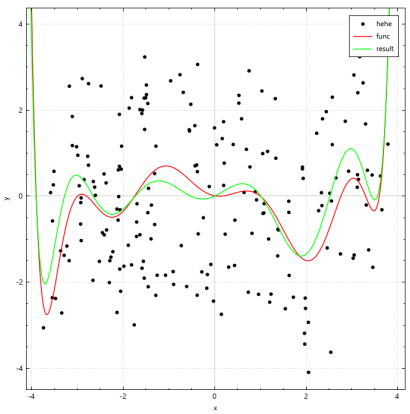
Кол-во точек	Шаги	Время	PyTorch Шаги	PyTorch Время
50	590	16	44	249
100	848	264	43	1
150	930	282	28	1
200	888	256	80	1
250	848	262	30	3

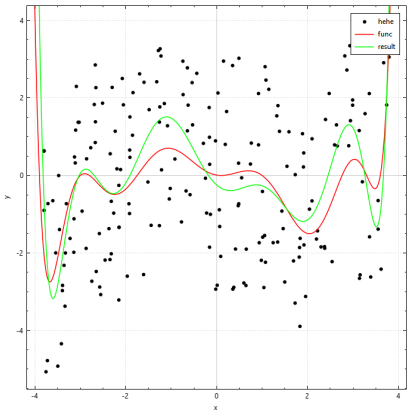
2.5 Scipy

Тут запускаем различные методы для решения задачи полиномиальной регрессии с использованием библиотек scipy.optimize.minimize и scipy.optimize.least squares. Затем сравниваем их реализации с нашими собственными методами и анализируем различия между ними.

```
29  def pf(x, cf):
30      deg = len(cf) - 1
31      return np.polyval(cf, x)
32  def start_pred(deg):
33      return np.zeros(deg + 1)
34  def inn(cf, x, y):
35      predicted_y = pf(x, cf)
36      residuals = y - predicted_y
37      mse = np.mean(residuals ** 2)
38      return mse
39  def polyMinReg(x, y, deg):
40      initial_guess = start_pred(deg)
41      result = minimize(inn, initial_guess, args=(x, y))
42      fitted_coeffs = result.x
43      return fitted_coeffs
44  def pMin(self, x_in, y_in, deg):
45      return polyMinReg(x_in, y_in, deg)
46  def inn(cf, x, y):
47      predicted_y = pf(cf, x)
48      residuals = y - predicted_y
49      return residuals
50  def polyLeastSqrReg(x, y, deg):
51      initial_guess = start_pred(deg)
52      result = least_squares(inn, initial_guess, args=(x, y))
53      fitted_coeffs = result.x
54      return fitted_coeffs
55  def pls(self, x_in, y_in, deg):
56      return polyLeastSqrReg(x_in, y_in, deg)
57  s_o = ScipyOptimizer()
58  cf_ls = s_o.pls(x, y, deg)
59  cf_min = s_o.pMin(x, y, deg)
```

При использовании метода minimize из библиотеки scipy.optimize для оптимизации полиномов высокой степени возникают проблемы из-за неэффективности метода оптимизации BF GS. В таких случаях рекомендуется использовать метод least-squares.





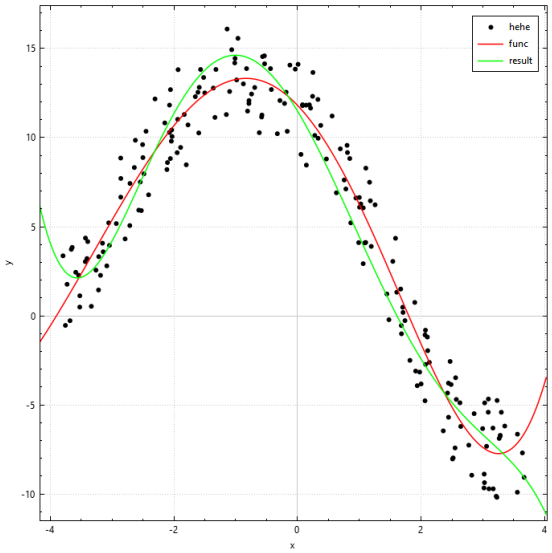
Кол-во точек	Время	Scipy Время
50	13	24
100	12	50
150	31	46
200	27	48
250	24	62

На низких степенях полинома нет заметных отличий между нашей и встроенной реализацией. Но на высоких степенях полинома встроенные методы лучше в аппроксимации и предотвращении переобучения. Наша реализация быстрее из-за умножения матриц.

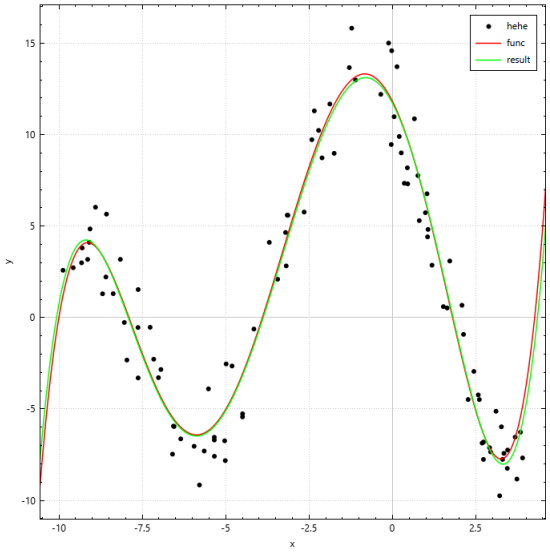
2.5.1 Gauss and Powell

```
63  def obj(cf, poly_deg, X, y):
64      y_pred = np.polyval(np.flip(cf), X)
65      rs = y - y_pred
66      return np.sum(rs ** 2)
67      
68  def jacobian(cf, poly_deg, X, y):
69      jacob = np.zeros((sampl, feat))
70      for i in range(sampl):
71          for j in range(feat):
72              jacob[i][j] = -X[i] ** (feat - j - 1)
73      return jacob
74
75  def get_coef(cf, poly_deg, X, y):
76      sampl = X.shape[0]
77      feat = poly_deg + 1
78      cf = np.zeros(feat)
79
80      gs = np.zeros(feat)
81      res = minimize(obj, gs, method='Powell', jac=jacobian)
82      cf = res.x
83      return cf
84
85  def predict(cf, poly_deg, X):
86      y_pred = np.polyval(np.flip(cf), X)
87      return y_pred
88
89  md = PowellDoglegPolynomialRegression(poly_deg=poly_deg)
90  cf = None
91  s = time.time()
92  get_coef(cf, poly_deg, x, y)
93  frst = time.time() - s
94  reg = Regression()
95  s = time.time()
96  cf, iters = reg.powell_dog_leg_regression(x, y, poly_deg)
97  two = time.time() - s
```

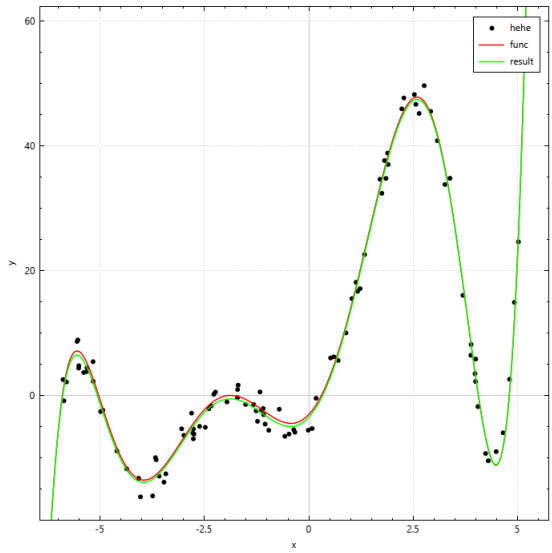
При малых показателях степени полинома наши методы сходятся хорошо, однако при повышенных степенях полинома алгоритмы из scipy проявляют более выраженную сходимость. Кроме того, время сходимости встроенных алгоритмов заметно превосходит время наших реализаций.



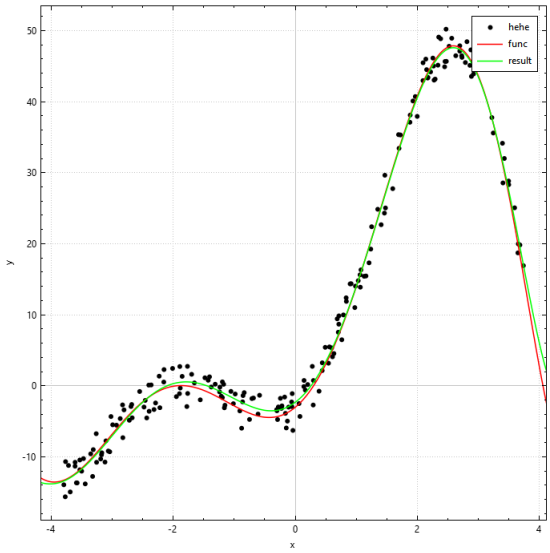
(a) Степень 5



(b) Степень 5



(a) Степень 7



(b) Степень 7

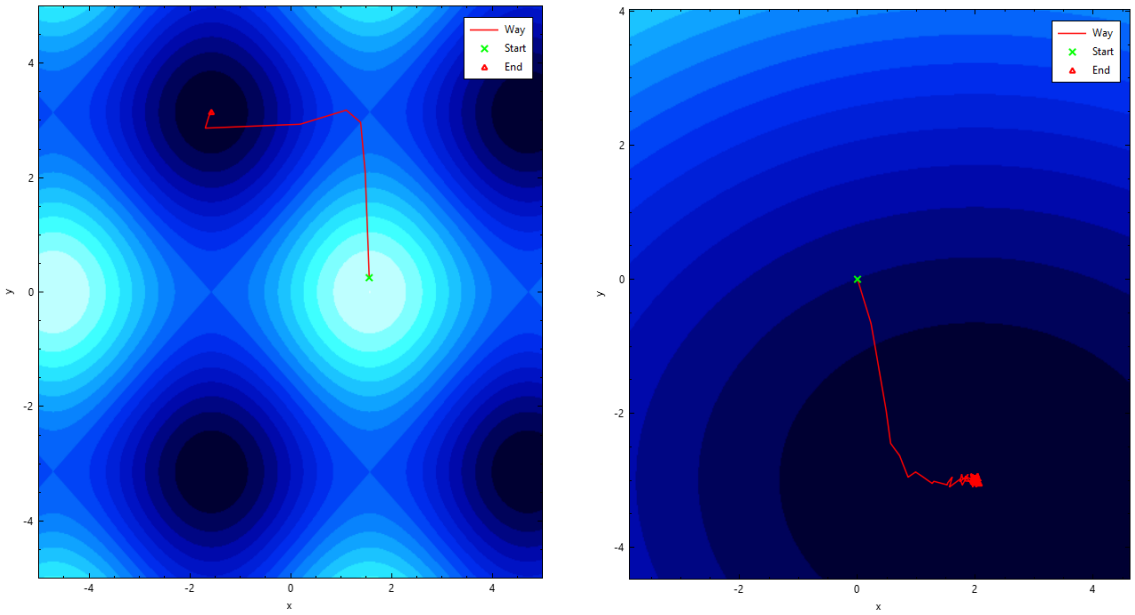
Таблица результатов для Gauss-Newton:

Кол-во точек	Время	Scipy Время
50	322	19
100	623	46
150	926	10
200	1239	7
250	1544	22

Таблица результатов для Powell Dog Leg:

Кол-во точек	Время	Scipy Время
50	18	1022
100	28	1013
150	4	976
200	20	865
250	5	791

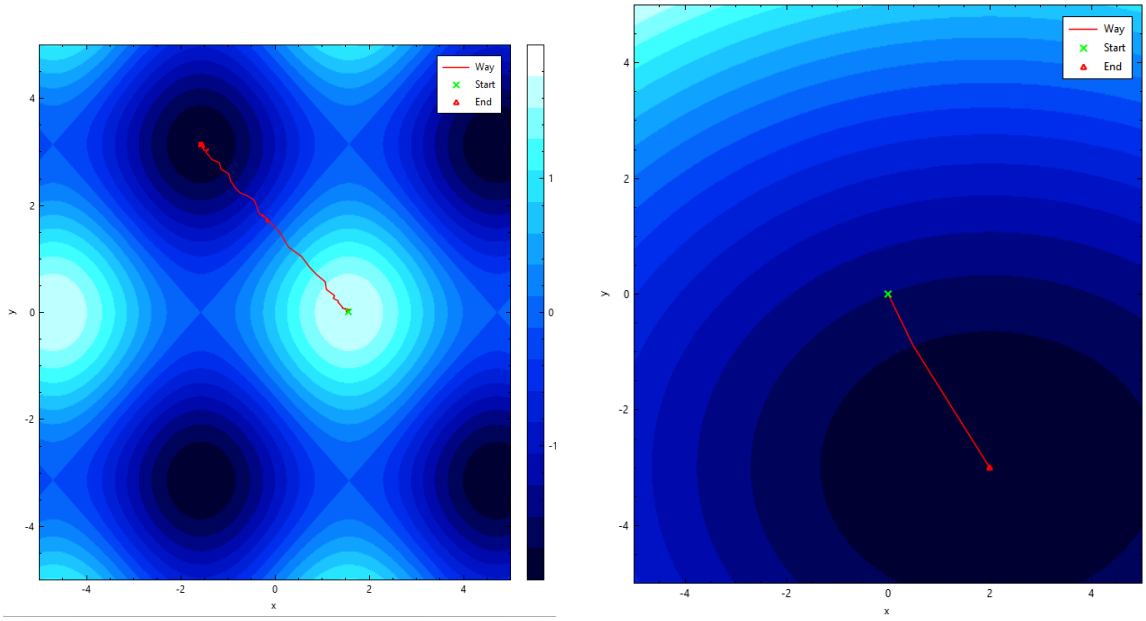
2.6 BFGS



Для функции $f(x, y) = \sin(x) + \cos(y)$

Начальная точка	BFGS с 3 лабы	Scipy BFGS
(0, 0)	6	4
(-1, 0)	4	3
(0, 2)	5	3
(-0.5, -0.5)	3	3
(0, 0.1)	5	2

2.7 L-BFGS



Для функции $f(x, y) = \sin(x) + \cos(y)$

Начальная точка	L-BFGS с 3 лабы	Scipy L-BFGS
(0, 2)	5	2
(0, 0)	6	3
(-1, 0)	4	2
(0, 0.1)	5	3
(-0.5, -0.5)	3	1

2.8 Градиент с помощью PyTorch и SciPy

```python
def sciPyGrad(f, s):
    return approx_fprime(s, f, epsilon=0.0001)
def pyTorchGrad(f, s):
    x, y = torch.tensor(s[0], requires_grad=True), torch.tensor(s[1], requires_grad=True)
    output = f(x, y)
    output.backward()
    return x.grad, y.grad
def gradient_by_hand(f, s, h=0.00001):
    return (f([s[0] + h, s[1]]) - f([s[0] - h, s[1]])) / (2 * h), (f([s[0], s[1] + h]) - f([s[0], s[1] - h])) / (2 * h)
```

Функция $f(x, y) = x^4 + y^4 - 4xy$

Старт	Наша реализация	Scipy	PyTorch
(2, -2)	(1.001, 1)	(1, 1)	(1, 1)
(-2, 2)	(-1, -1.2)	(-1, -1)	(-1, -1)
(3, 0)	(1.001, 1)	(1, 1)	(1, 1)

Функция $f(x, y) = (x - 2)^2 + (y + 3)^2 - 3xy$

Старт	Наша реализация	Scipy	PyTorch
(0, 0)	(1.999, -3)	(2, -3)	(2, -3)
(4, 1)	(2.0001, -3)	(2, -3)	(2, -3)
(1, 4)	(2, -2.9)	(2, -3)	(2, -3)

Функция $f(x, y) = x^3 - 3xy + y^3$

Старт	Наша реализация	Scipy	PyTorch
(2, -2)	(0.999, 1)	(1, 1)	(1, 1)
(-2, 2)	(-1, -0.99)	(-1, -1)	(-1, -1)
(3, 3)	(1.1, 1)	(1, 1)	(1, 1)

Выбор метода для вычисления градиента несущественен, поскольку все методы достаточно точно приближают значения градиента в заданных точках.

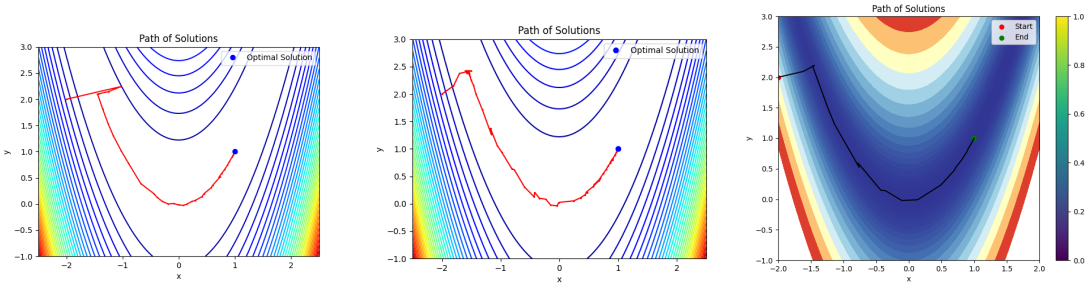
2.9 L-BFGS, Nelder-Mead и Trust-Constr

Давайте рассмотрим функцию Розенберга $f(x, y) = (1-x)^2+100(y-x^2)^2$. Мы будем искать ее минимум с помощью трех различных алгоритмов: L-BFGS, Nelder-Mead и Trust-Constr.

На второй паре графиков мы ограничим область поиска значений $x < 2.5$. Это означает, что мы будем искать минимум функции только в этой области, в то время как y может принимать любые значения.

На третьей паре графиков мы ограничим область поиска значений $0 < x < 2.5$. Это означает, что мы будем искать минимум функции только в этой области, а значения y могут быть любыми.

Таким образом, мы применяем три различных алгоритма оптимизации к функции Розенброка с различными ограничениями на область поиска значений x.



2.10 Доп. Задание