Лабораторная работа №3

Метод стохастического градиентного спуска (SGD) и его модификации

Задачи:

1. Описание следующих методов:
    - метод SGD с разным размером батча
    - модификации SGD (Nesterov, Momentum, AdaGrad, RMSProp, Adam)
2. Реализация вышеперечисленных методов
3. Исследование методов на эффективность на нескольких функциях:
    - сравнение эффективности SGD при разном размере батча
    - сравнение эффективности SGD при разных функциях изменения шага
    - сравнение эффективности реализованного SGD и библиотечного SGD его модификаций

Часть 1. Описания используемых методов

SGD

Алгоритм похож на алгоритм стандартного градиентного спуска, только для подсчета нового приближения весов на каждом шаге используется одно или некоторая подвыборка слагаемых минимизируемой функции. Также для ускорения пересчета значения функции можно использовать формулу скользящего среднего.

Модификации SGD:

Все модификации SGD были взяты из библиотеки torch.optim

Momentum

SGD, но на каждой новой итерации оптимизации используется скользящее среднее градиента (в библиотечной реализации есть гиперпараметр momentum отвечающий за это)

Nesterov

Momentum, но вычисление градиента в следующей точке по направлению импульса

AdaGrad

SGD, но каждый вес имеет собственную скорость обучения (полученную путем умножения матрицы на общую скорость обучения). Это увеличивает скорость обучения для параметров с редкими данными и уменьшает скорость обучения для параметров с менее редкими.

RMSProp

SGD, но скорость обучения настраивается для каждого параметра. Идея заключается в делении скорости обучения для весов на скользящие средние значения недавних градиентов для этого веса. (есть гиперпараметр α, отвечающий за это)

Adam

RMSProp, но используются скользящие средние как градиентов, так и вторых моментов градиентов (в гиперпараметрах есть β коэффициенты, используемые для вычисления текущих средних значений градиента и его квадрата)

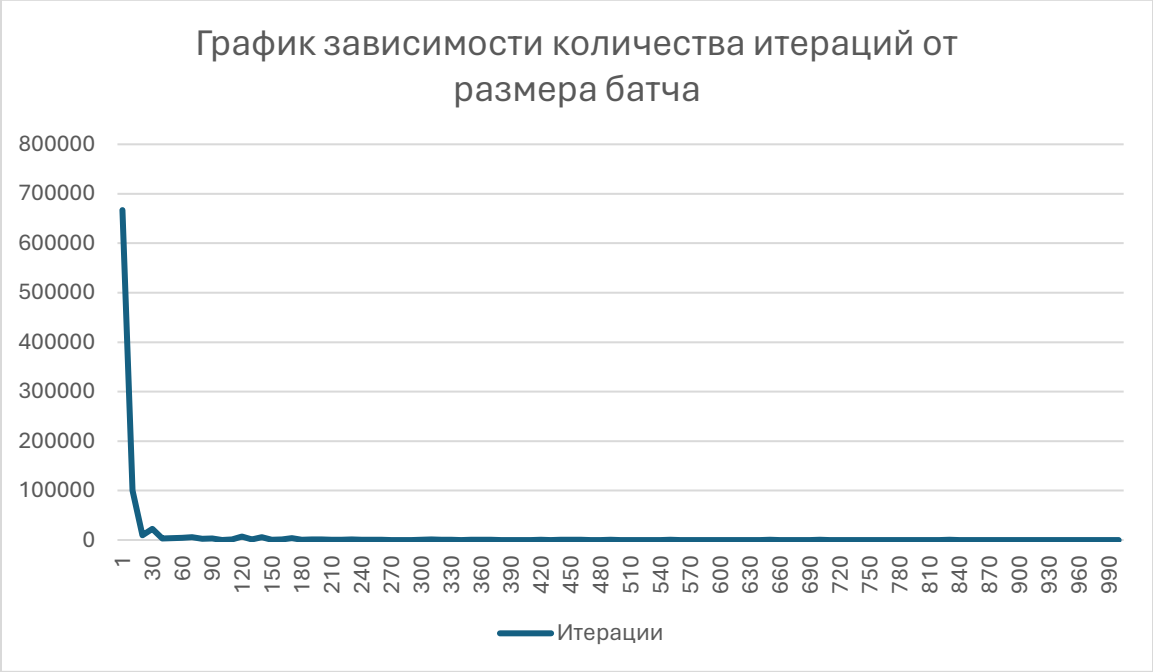
Часть 2. Исследование эффективности

Зависимость эффективности работы стохастического градиентного спуска от размера батча (y = 666x – 666):

Размер батча	Количество эпох	Количество итераций	Время выполнения, с	Использованная память, байт
1	666	667000	66.10243	71576
10	999	100000	47.92384	59852
20	195	9800	8.81217	63564
30	678	22407	29.22013	60147
40	127	3200	5.6043	59244
50	176	3540	7.67459	59244
60	287	4608	12.05628	59272
70	383	5376	16.18527	59272
80	186	2244	7.74552	59244
90	283	3124	12.09529	59272
100	3	40	0.17939	58864
110	122	1107	5.19553	59761
120	883	7072	36.26558	59328
130	202	1421	7.98721	59244
140	830	5817	34.33564	59272
150	128	774	5.07916	59244
160	235	1416	9.61854	59244
170	723	3620	26.39765	59272
180	119	600	4.7325	59244
190	164	825	6.72062	59244
200	181	910	7.68606	60040
210	65	264	2.37643	61628
220	60	244	2.29778	62988
230	280	1124	10.9913	64376
240	91	368	3.75996	65996
250	146	588	6.29534	67384
260	264	795	9.00989	68744
270	29	90	1.08069	70284
280	54	165	2.04972	71756
290	51	156	2.00256	73116
300	127	384	4.99702	76666
310	316	951	12.57079	76863
320	109	330	4.53741	77548
330	64	195	2.75316	78908
340	84	170	2.57851	80240
350	324	650	10.1019	81680
360	152	306	4.90973	83400

370	107	216	3.53822	84732
380	26	54	0.91102	86064
390	26	54	0.9363	87368
400	47	96	1.69298	88812
410	20	42	0.74878	92420
420	221	444	8.0625	93744
430	36	74	1.41007	93228
440	262	526	9.91972	94728
450	279	560	10.82562	96116
460	330	662	13.01663	97988
470	52	106	2.12449	99292
480	33	68	1.39221	100540
490	216	434	8.85841	102040
500	38	78	1.70444	103372
510	88	89	2.10545	104592
520	43	44	1.09088	106064
530	55	56	1.36616	107944
540	52	53	1.30887	109360
550	515	516	12.73038	110776
560	64	65	1.6655	112080
570	121	122	3.14565	113440
580	72	73	1.94677	114800
590	76	77	2.08065	116160
600	82	83	2.26306	118160
610	85	86	2.33395	119520
620	47	48	1.32591	120880
630	41	42	1.17614	122240
640	125	126	3.63834	123600
650	229	230	6.67036	124960
660	41	42	1.25403	126320
670	57	58	1.73849	127680
680	61	62	1.87868	129744
690	67	68	2.08495	131104
700	221	222	6.80876	132464
710	46	47	1.50375	133712
720	48	49	1.55491	135128
730	95	96	3.06695	136544
740	102	103	3.29103	137904
750	50	51	1.6745	139264
760	120	121	3.95099	140624
770	56	57	1.87557	142784
780	114	115	3.90291	144144
790	63	64	2.19202	145504
800	54	55	1.89908	146864
810	51	52	1.8192	148224

820	60	61	2.16054	149584
830	180	181	6.38721	150944
840	44	45	1.63088	152304
850	64	65	2.37171	153664
860	42	43	1.58647	155024
870	65	66	2.56831	157280
880	68	69	2.64042	158640
890	50	51	2.00979	160000
900	53	54	2.05187	161360
910	47	48	1.87608	162720
920	54	55	2.15003	164080
930	89	90	3.60783	165440
940	45	46	1.81404	166744
950	49	50	1.99914	168160
960	51	52	2.08367	169520
970	62	63	2.61383	170880
980	48	49	2.03021	173264
990	69	70	3.04214	174624
1000	50	51	2.18876	175984



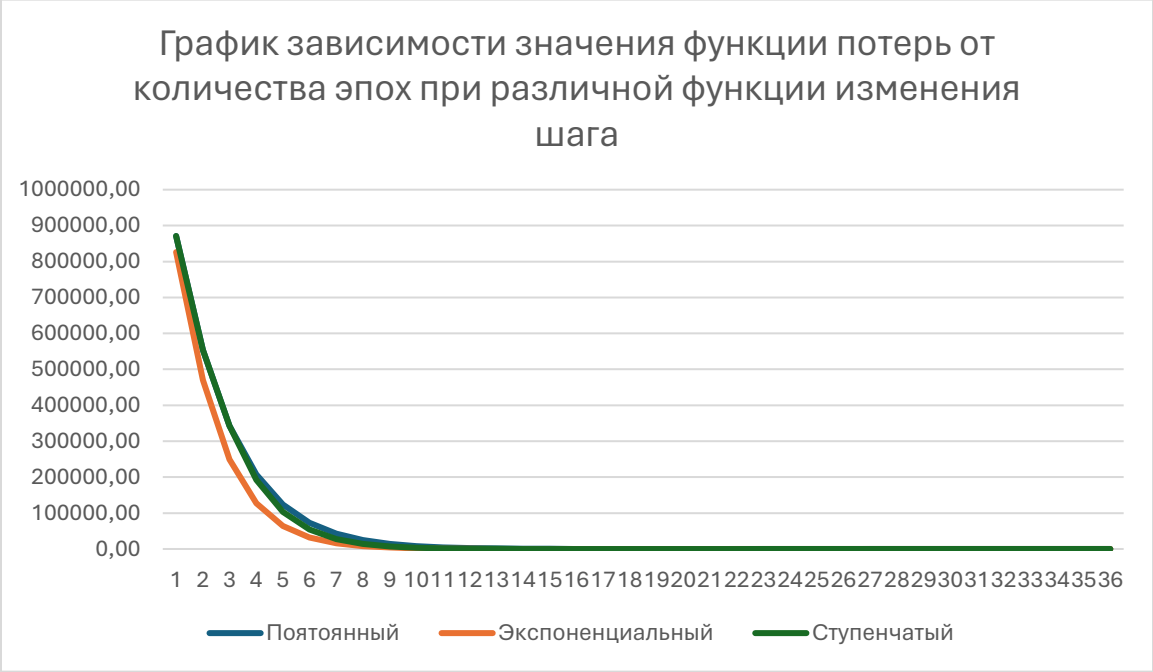
Количество итерации уменьшается с увеличением размера батча, т. к. возрастает точность вычисления направления шага и средних

потерь. При этом при увеличении батча растет потребление памяти, т. к. батчи требуется хранить.

Зависимость эффективности работы стохастического градиентного спуска от функции изменения шага (learning rate)

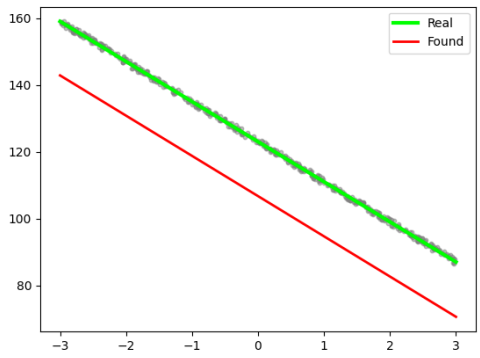
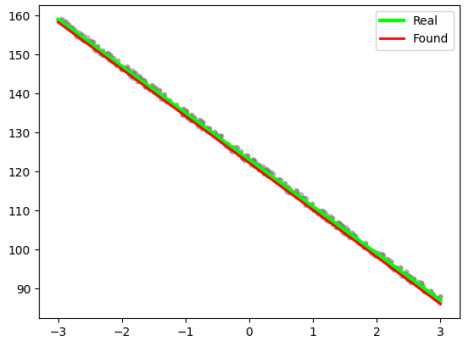
Количество эпох	Постоянный шаг $(1e-3)$	Экспоненциальный шаг $(1e-3\cdot e^{-epoech\cdot 0.05})$	Ступенчатый шаг $(1e-3\cdot 0.9^{\frac{epoech+1}{5}})$
1	871039.428	826259.086	871039.428
2	553929.009	470126.455	553929.009
3	342380.755	249410.924	342380.755
4	207463.55	127367.49	192479.778
5	123866.78	64039.62	103170.272
6	73113.713	32055.532	53841.652
7	42765.525	16031.773	27655.699
8	24830.69	8016.933	14067.267
9	14330.168	4008.955	7074.739
10	8228.526	2004.775	3544.543
11	4704.904	1002.612	1773.639
12	2680.53	501.499	887.193
13	1522.533	250.93	443.8
14	862.561	125.639	222.071
15	487.608	62.991	111.204
16	275.16	31.666	55.771
17	155.068	16.002	28.054
18	87.32	8.17	14.195
19	49.17	4.254	7.266
20	27.72	2.296	3.802
21	15.675	1.316	2.069
22	8.921	0.827	1.203
23	5.137	0.582	0.77
24	3.019	0.459	0.554
25	1.835	0.398	0.445
26	1.173	0.368	0.391
27	0.803	0.352	0.364
28	0.597	0.345	0.351
29	0.482	0.341	0.344
30	0.418	0.339	0.34
31	0.382	0.338	0.339
32	0.362	0.337	0.338
33	0.351	0.337	0.337
34	0.345	0.337	0.337

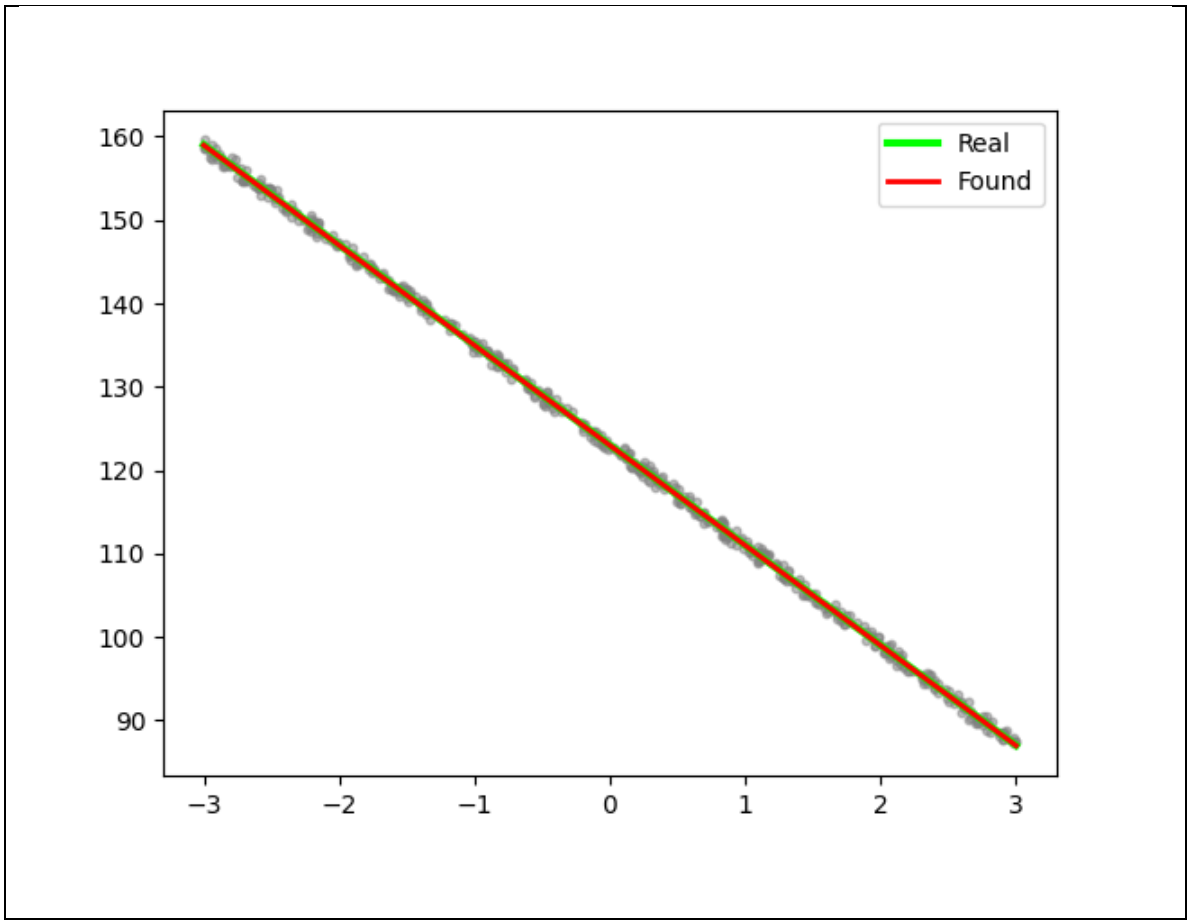
35	0.341	0.337	0.337
36	0.337	0.337	0.337



SGD с экспоненциальным и ступенчатым шагом быстрее сходятся, так как учитывают номер текущей эпохи.

Результаты при различном количестве эпох:

20	50
	
100	



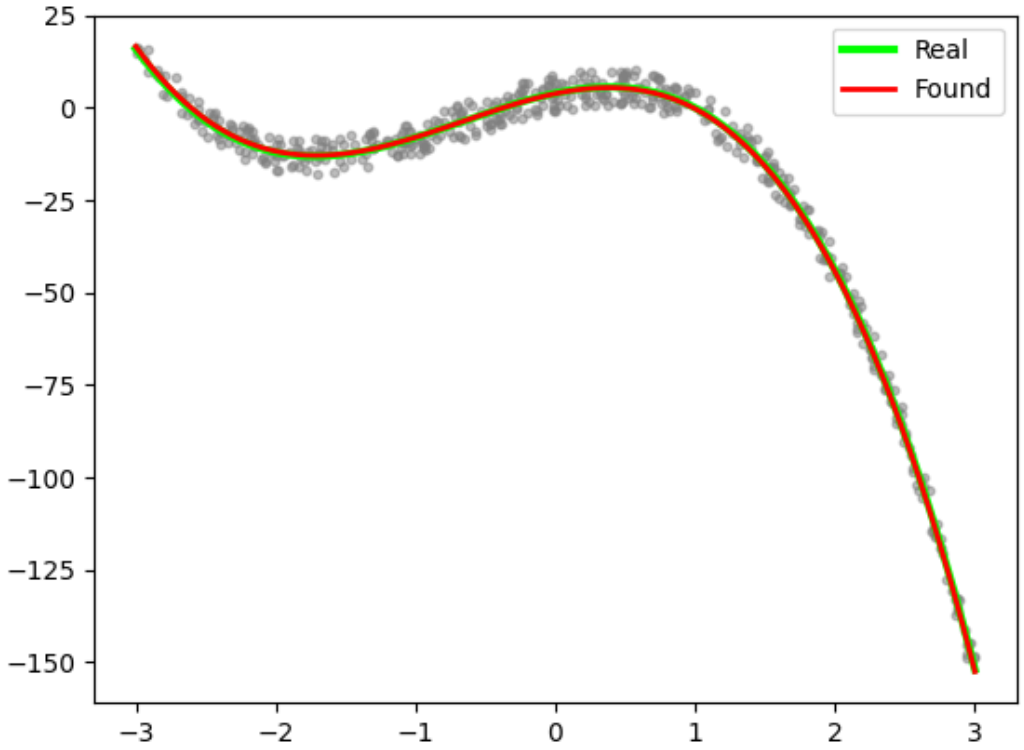
Исследование эффективности библиотечных функций

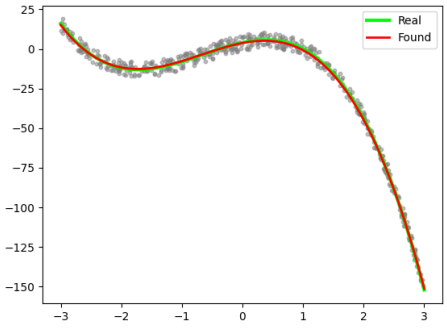
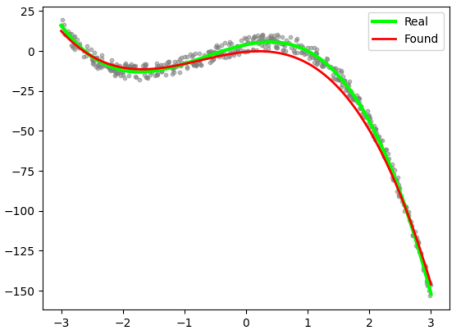
	Время работы	Использованная память	Значение функции потерь
SGD (lr = 1e-2)	0.59129	67132	0.328
Momentum (lr = 1e-2)	0.31365	12923	0.328
Nesterov (lr = 1e-2)	0.30447	22494	0.329
AdaGrad (lr = 1000)	0.26214	20803	0.335
RMSProp (lr = 10)	0.30287	15679	0.333
Adam (lr = 10)	0.43081	17703	0.328

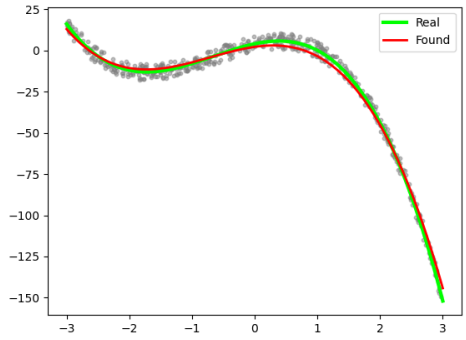
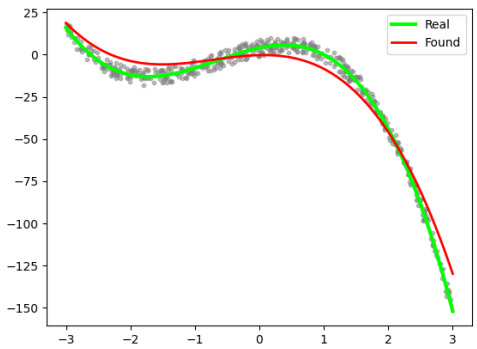
Модификации работают быстрее простого SGD за счёт более тщательного изменения весов.

Дополнительное задание 1.

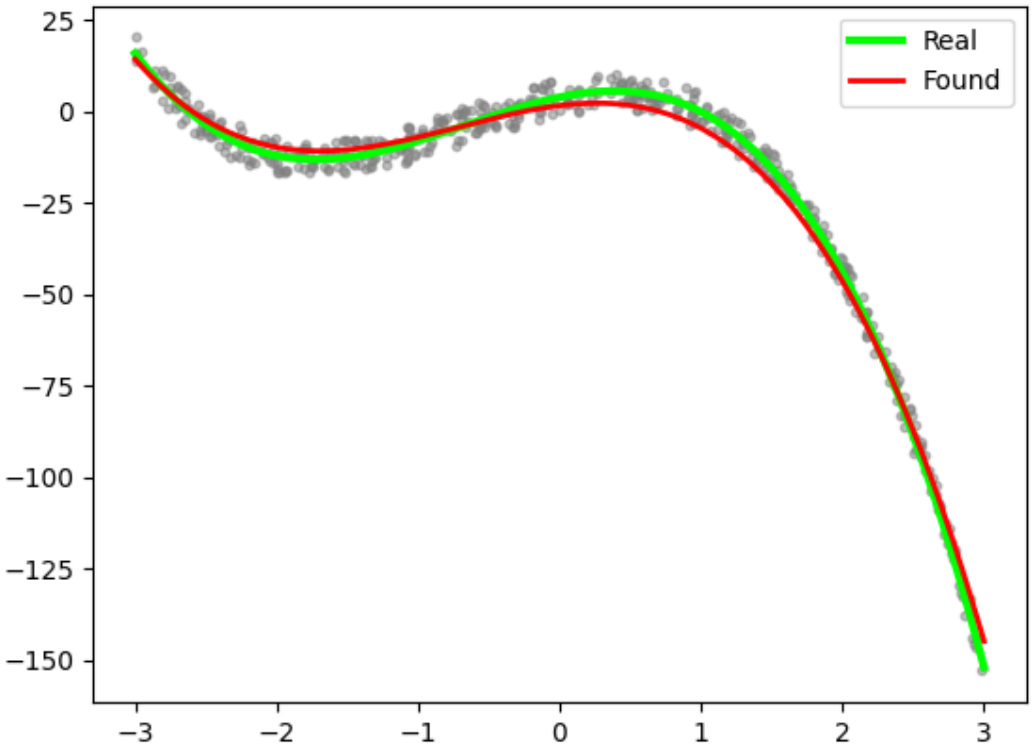
1. Полином $4 + 8x - 8x^2 - 4x^3$



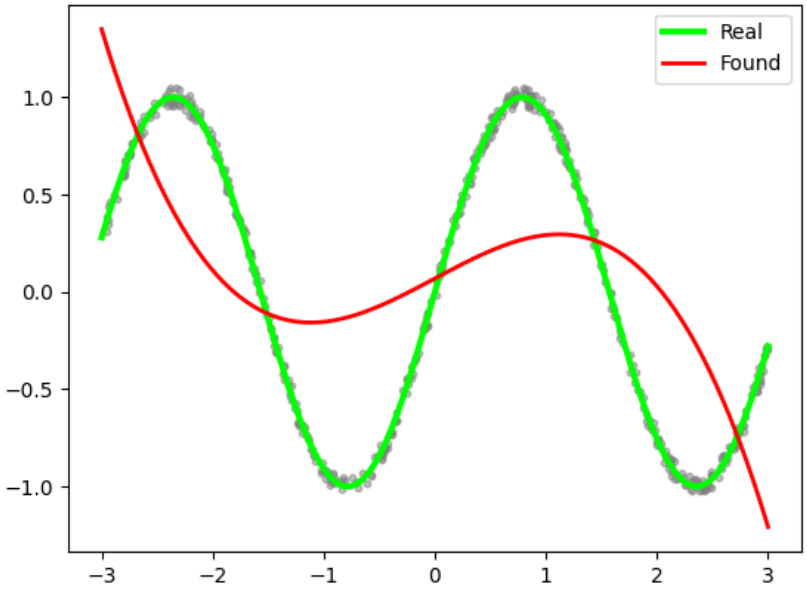
	Коэффициент важности	
	0.5	5
L1		

L2		

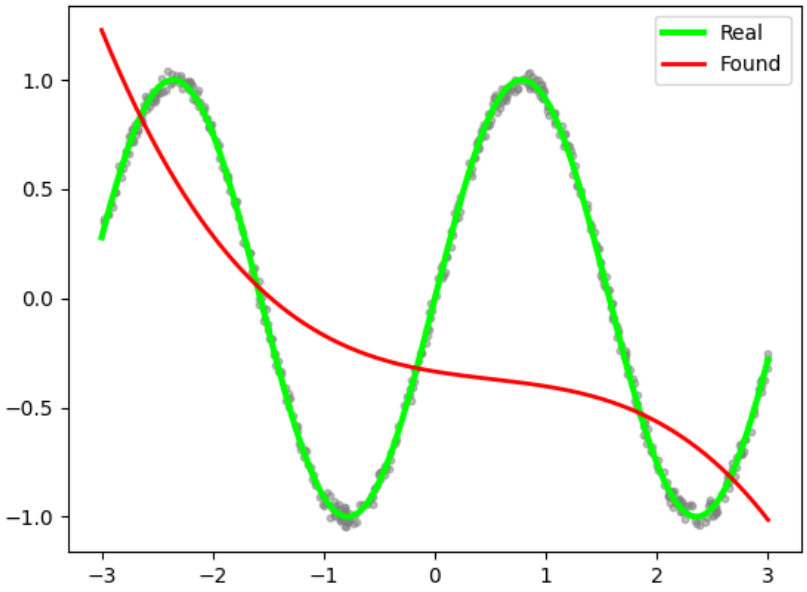
Elastic (с коэффициентом 0.5):



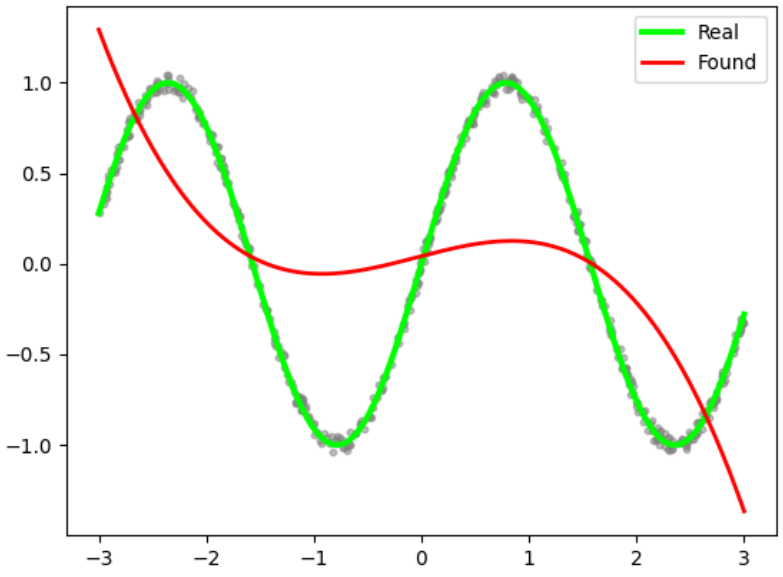
2. Функция sin $(2x)$



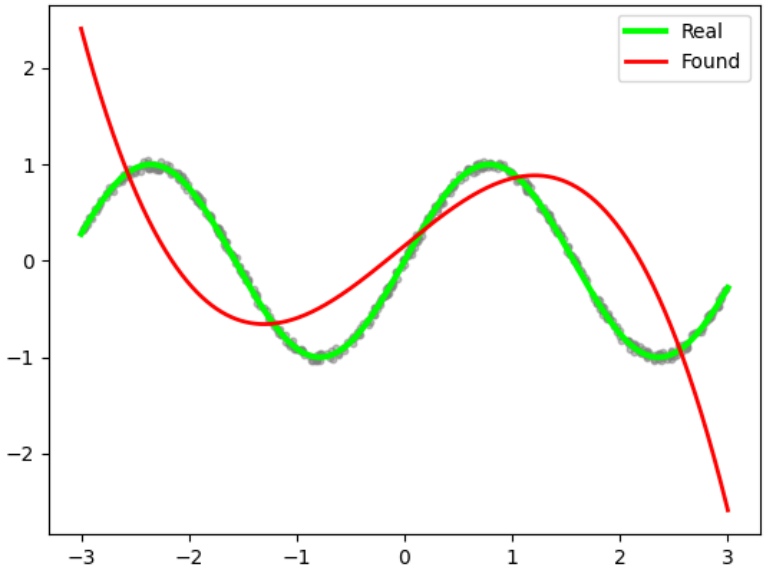
L1 (с коэффициентом 0.5):



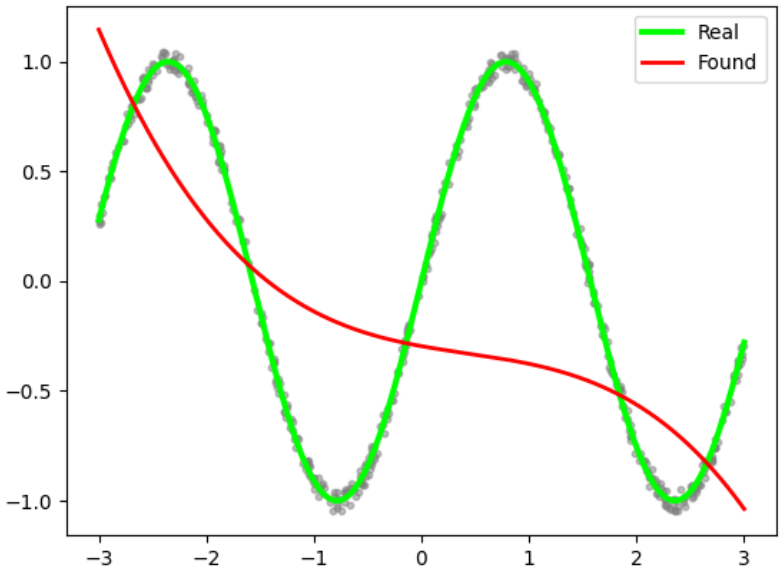
L2 (с коэффициентом 0.5):



L2 (с коэффициентом -1):



Elastic (с коэффициентом 0.5):


Real
Found
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3

Дополнительное задание 2. Метод опорных векторов

Метод опорных векторов (SVM) — один из наиболее популярных методов обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом.

Алгоритм работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора. Вектора, лежащие ближе всех к разделяющей гиперплоскости, называются опорными векторами (support vectors).

В пространстве $\mathbb{R}^n$ уравнение $\langle \vec{w},\ \vec{x} \rangle\ -\ b\ =\ 0$ при заданных $\vec{w}, b$ определяет гиперплоскость. Параметр $\vec{w}$ определяет вектор нормали к гиперплоскости, а через $\frac{b}{\|\vec{w}\|}$ выражается расстояние от гиперплоскости до начала координат.

Метод опорных векторов строит классифицирующую функцию $F(x) = sign(\langle \vec{w},\ \vec{x} \rangle -\ b)$

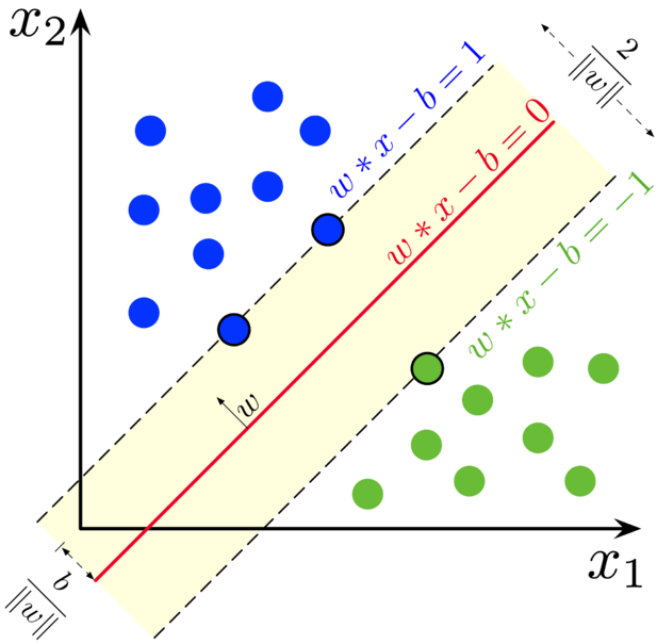
Отступ (англ. *margin*) — характеристика, оценивающая, насколько объект "погружён" в свой класс. Для линейного классификатора отступ определяется уравнением: $M_i(\vec{w}\,, b) = y_i(\langle \vec{w},\ \vec{x_i} \rangle -\ b$.

Далее, мы хотим выбрать такие $\vec{w}$ и b, которые максимизируют расстояние до каждого класса. Можно подсчитать, что данное расстояние равно $\frac{1}{\|\vec{w}\|}$. Проблема нахождения максимума $\frac{1}{\|\vec{w}\|}$ эквивалентна проблеме нахождения минимума $\|\vec{w}\|^2$

Это приводит нас к постановке задачи оптимизации:

$$\begin{cases} \|\vec{w}\|^2 \to \min\limits_{w,b} \\ M_i(\vec{w}\,, b) \geq 1, \qquad i = 1 \ldots m \end{cases}$$

На практике случаи, когда данные можно разделить гиперплоскостью довольно редки. В этом случае поступают так: все элементы обучающей выборки вкладываются в пространство более высокой размерности с помощью специального отображения. При этом отображение выбирается так, чтобы в новом пространстве выборка была линейно разделима. Построение SVM в таком случае происходит так же, как и раньше, но в качестве векторов признаковых описаний используются векторы $\varphi(x)$



Пара ссылок с иллюстрациями работы метода:

- https://www.youtube.com/watch?v=OdlNM96sHio&ab_channel=udiprod

- https://www.youtube.com/watch?v=_YPScrckx28&ab_channel=VisuallyExplained