

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

Методы оптимизации
Отчёт по лабораторной работе №2

Работу выполнили:

Барсукова Ольга М3233

Зызлаев Артем М3233

Ермольев Михаил М3233

Жунусов Данияр М3233

Преподаватель:

Андреев Юрий Александрович

ИТМО

Санкт-Петербург
2025

Постановка задач и целей работы:

1. Основное задание. Реализуйте следующие методы:
 - 1.1. метод Ньютона с выбором шага и с одномерным поиском (любым методом);
 - 1.2. `scipy.optimize`: метод Newton-CG и один-два квазиньютоновских метода.
2. Сделайте свою реализацию любого квазиньютоновского метода или разберите какую-либо стороннюю библиотечную реализацию данного метода, проанализируйте особенности реализации, модифицируйте и интегрируйте код метода в свой проект. Исследуйте его на эффективность в сравнении с другими методами
3. `optuna`: Примените методы библиотеки к примерам из первой и второй лабораторной. Найдите с их помощью более оптимальные значения гиперпараметров.

Реализация

Метод Ньютона с выбором шага и с одномерным поиском (любым методом)

В нашей работе мы создали свою реализацию метода Ньютона. Реализовано две вариации: с выбором фиксированного шага и метод Ньютона-Рафсона, где шаг выбирается аналитически при помощи любого желаемого метода одномерного поиска, реализованного в лабораторной работе 1.

```
def newton_method(init: np.array, gd, f,
mode="newton_raphson", a=1.0, tol=1e-10, max_iter=100,
eps=1e-6, verbose=False):
    x = np.copy(init)
    res = [x]
    gd.iter_count = 0
    gd.grad_count = 0
    gd.hes_count = 0

    for i in range(max_iter):
        gd.iter_count += 1
```

```

grad = gd.grad(f, x)
gd.grad_count += 1

hess = gd.hessian(f, x)
gd.hes_count += 1

try:
    direction = -np.linalg.solve(hess, grad)
except np.linalg.LinAlgError:
    print("Гессиан вырожден. Прекращаем.")
    break

if mode == "newton_raphson":
    f_line = lambda alpha: f(x + alpha[0] *
direction)
    alpha_result, f_res, iter_count, func_count,
grad_count = gd.find_min(func=f_line, init=np.array([0.0]))
    step_size = alpha_result[-1]
else:
    step_size = a

x_new = x + step_size * direction
res.append(x_new)

gd.func_count+=2
if abs(f(x_new) - f(x)) < tol:
    break

x = x_new

return np.array(res), f(x), gd.iter_count, gd.grad_count,
gd.hes_count

```

Метод Ньютона требует подсчёта матрицы вторых производных – гессиана.

```

def hessian(f, x, eps=1e-5):
    n = len(x)
    hess = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            dx_i = np.zeros(n)
            dx_j = np.zeros(n)
            dx_i[i] = eps
            dx_j[j] = eps
            f1 = f(x + dx_i + dx_j)
            f2 = f(x + dx_i - dx_j)
            f3 = f(x - dx_i + dx_j)
            f4 = f(x - dx_i - dx_j)
            hess[i, j] = (f1 - f2 - f3 + f4) / (4 * eps ** 2)
    return hess

```

Реализация метода Ньютона и квазиньютоновских методов в библиотеке `scipy.optimize`

Библиотека предоставляет множество методов минимизации, в том числе **Newton-CG**, **BFGS** и **L-BFGS-B**. Благодаря этому можно сравнить эффективность наших и библиотечных методов оптимизации.

```

def newton_method_cg(init: np.array, grad, func,
method="Newton-CG", grid=None):
    jacobian = lambda x: grad(func, x)
    print_output_opt(init, func, method, jac=jacobian,
grid=np.array([-6, 6]))

def bfgs(init: np.array, func, method="BFGS"):
    print_output_opt(init, func, method, grid=np.array([-6,
6]))

def l_bfgs_b(init: np.array, func, method="L-BFGS-B"):
    print_output_opt(init, func, method, grid=np.array([-6,
6]))

```

Своя реализация квазиньютоновского метода

В работе реализована вариация метода **BFGS**. Метод итерационно приближает гессиан функции, что позволяет избежать его вычисления напрямую.

```
def bfgs_method(init: np.ndarray, gd, f, start_beta=1.0,
tol=1e-10, max_iter=100, backtrack=False, back_coef=0.8):
    x = np.copy(init)
    ones = np.eye(x.shape[0])
    B = ones * (1 / start_beta)
    res = [x]
    gd.iter_count = 0
    gd.grad_count = 0
    gd.hes_count = 0
    for i in range(max_iter):
        grad_x = gd.grad(f, x)
        sk = -np.dot(B, grad_x)

        if not backtrack:
            x_next = gd.gradient_step(func=f, x=x, dir=sk)
        else:
            alpha = backtracking(x, grad_x, gd, f, sk,
q=back_coef)
            x_next = x + alpha * sk

        res.append(x_next)
        grad_x_next = gd.grad(f, x_next)
        yk = grad_x_next - grad_x
        gd.grad_count += 2
        if abs(np.dot(yk, sk)) < tol:
            break
        pk = 1 / np.dot(yk, sk)
        x = x_next
        if np.linalg.norm(sk) < tol:
            break
        B = np.dot(np.dot(ones - pk * np.outer(sk, yk), B),
ones - pk * np.outer(yk, sk)) + pk * np.dot(sk, sk)
```

```

        gd.iter_count += 1
    return np.array(res), f(x), gd.iter_count, gd.grad_count,
gd.hes_count

```

Выбор шага производится либо одним из методов, реализованных в первой лабораторной, либо методом backtracking-а.

Поиск оптимальных значений гиперпараметров

С использованием средств библиотеки **optuna** были выбраны значения гиперпараметров, которые должны улучшить результаты оптимизации.

```

# для тестирования градиентных спусков
def objective_gd(trial):
    gd = GradientDescending()
    gd.step_strategy =
    trial.suggest_categorical("step_strategy", ["fixed", "decay",
    "sqrt_decay", "exponential"])
    learning_rate = trial.suggest_float("learning_rate",
    0.001, 1.0)
    eps = trial.suggest_float("eps", 1e-5, 1e-1)
    noise_scale = trial.suggest_float("noise_scale", 0.0,
    0.1)

    def f(x):
        return (x[0] - 2) ** 2 + (x[1] + 3) ** 2

    init = np.array([10.0, -10.0])
    result_path, f_val, iters, func_calls, grad_calls =
    gd.find_min(
        f, init=init, learning_rate=learning_rate, eps=eps,
        max_iterations=1000, noise_scale=noise_scale
    )

    return f_val

# для тестирования метода ньютона

```

```

def objective_newton(trial):
    tol = trial.suggest_float("tol", 1e-8, 1e-3)
    max_iter = trial.suggest_int("max_iter", 10, 100)

    # Можно задать произвольный метод одномерного спуска
    gd = GradientDescending()

    def f(x):
        return (x[0] - 3) ** 2 + (x[1] + 1) ** 2

    init = np.array([5.0, -5.0])

    x_opt, f_val, iter_count, grad_calls, hess_calls =
    newton_method(init=init, gd=gd, f=f, tol=tol,
    max_iter=max_iter)
    return f_val

```

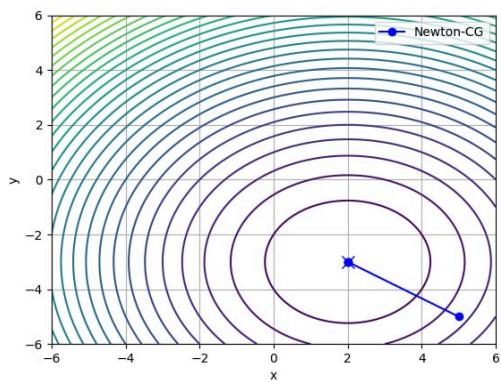
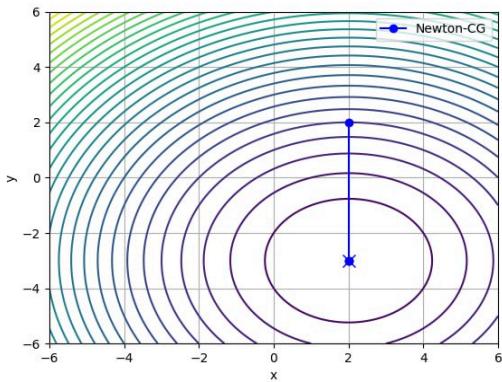
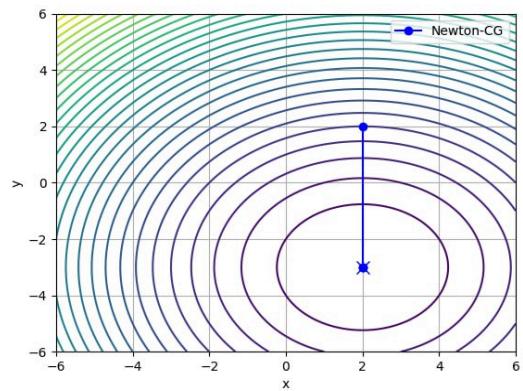
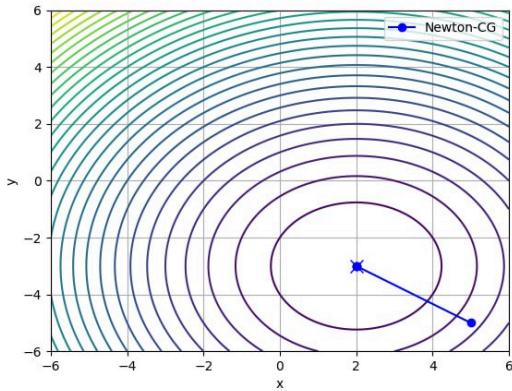
Исследование на различных функциях

Методы Ньютона: наш и библиотечный

$$(x - 2)^2 + (y + 3)^2$$

Реализация	start	method	final	min	iter	grad count	hessian count
Наша	(5, -5)	GD	(2.079, -3.052)	0.009	211	211	100
Scipy	(5, -5)	CG	?	7.4e-18	3	5	-
Наша	(2, 2)	GD	(2, -2.961)	0.0015	206	206	100
Scipy	(2, 2)	CG	?	1.007e-17	3	5	-
Наша	(2, 2)	DGD	(2, -2.885)	0.0131	33	10	2

Scipy	(2, 2)	CG	?	1.0079 e-17	3	5	-
Наша	(5, -5)	DGD	(2.197, -3.131)	0.0131	49	14	2
Scipy	(5, -5)	CG	?	7.3858 e-18	3	5	-

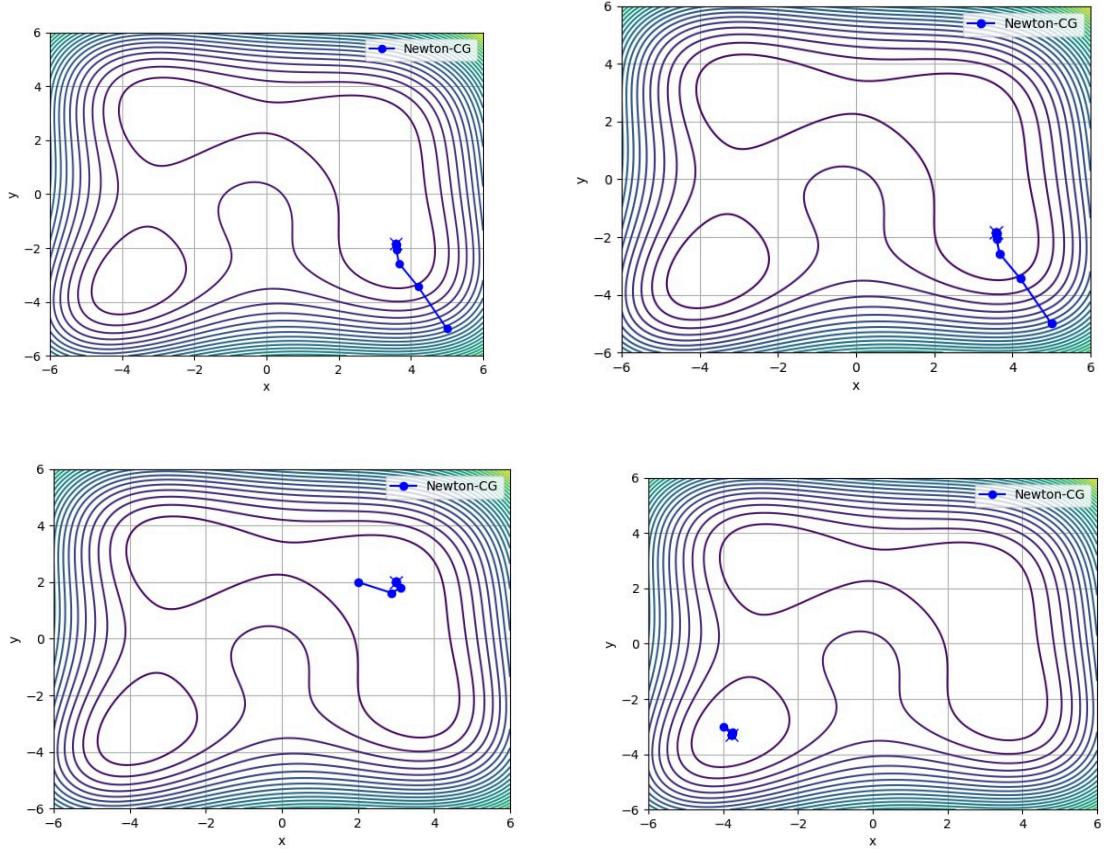


Optuna:

<i>method</i>	<i>scheduler</i>	<i>learning rate</i>	<i>eps</i>	<i>noise scale</i>	<i>best result</i>
GD	<i>decay</i>	0.52	4.5524e-5	0.021379	2.168e-23

$$(x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

<i>Реализация</i>	<i>start</i>	<i>method</i>	<i>final</i>	<i>min</i>	<i>iter</i>	<i>grad count</i>	<i>hessian count</i>
Наша	(5, -5)	GD	(-0.005, 2.92)	67.607	15	2	2
Scipy	(5, -5)	CG	?	5.2958 8e-12	8	19	?
Наша	(2, 2)	GD	(-0.039, 2.94)	67.516	215	201	100
Scipy	(2, 2)	CG	?	5.2958 23e-12	8	19	?
Наша	(2, 2)	DGD	(2.975, 1.98)	0.013	255	214	100
Scipy	(2, 2)	CG	?	2.6382 6e-17	7	27	?
Наша	(-4, -3)	DGD	(-3.82, -3.3)	0.0131 3	253	215	100
Scipy	(-4, -3)	CG	?	9.1533 8e-19	6	15	?

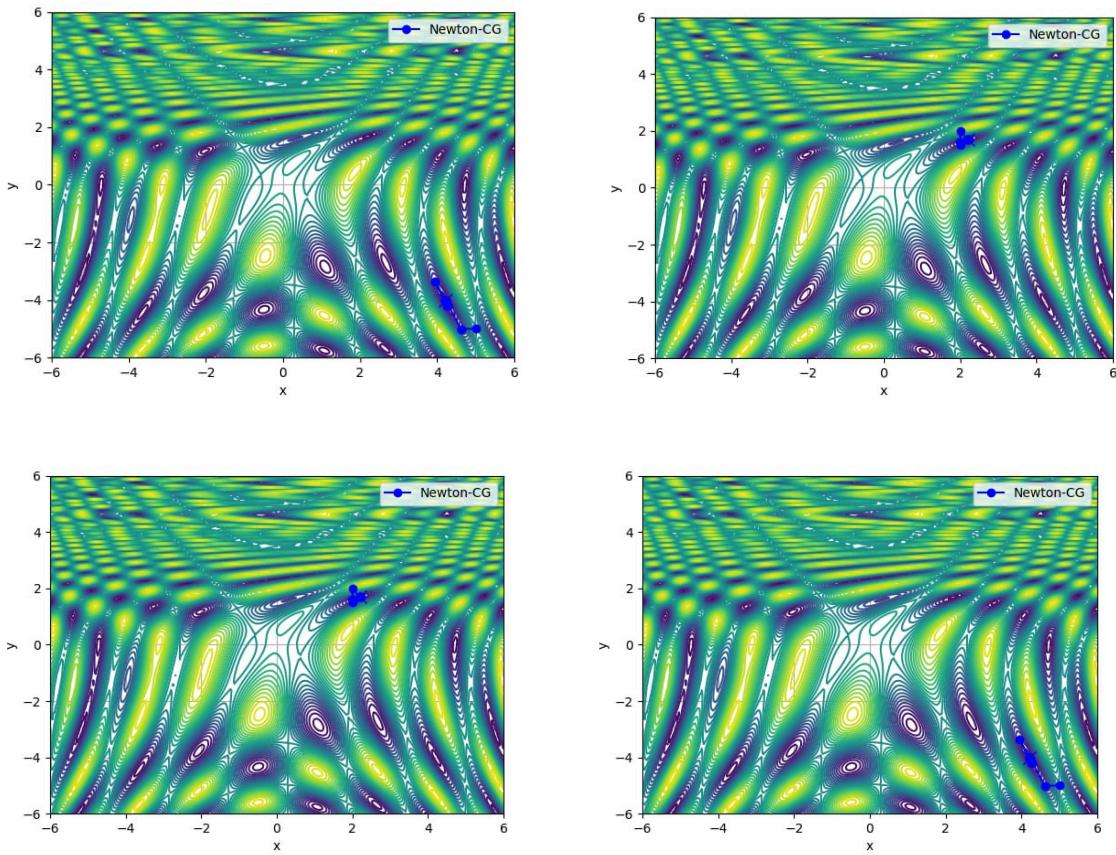


Optuna

<i>method</i>	<i>scheduler</i>	<i>learning rate</i>	<i>eps</i>	<i>noise scale</i>	<i>best result</i>
GD	decay	0.91139	1.13218	0.0981	1.5436e-10

$$\sin(0.5 * x^2 - 0.25 * y^2 + 3) * \cos(2 * x + 1 - e^y)$$

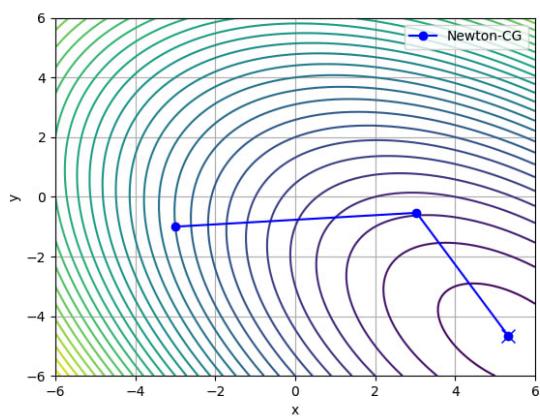
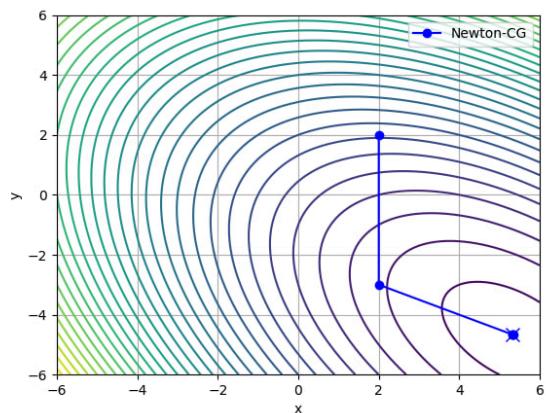
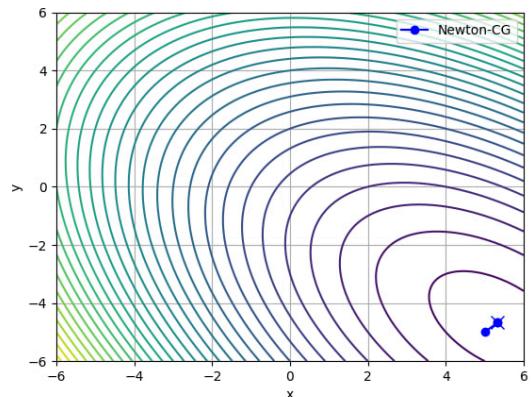
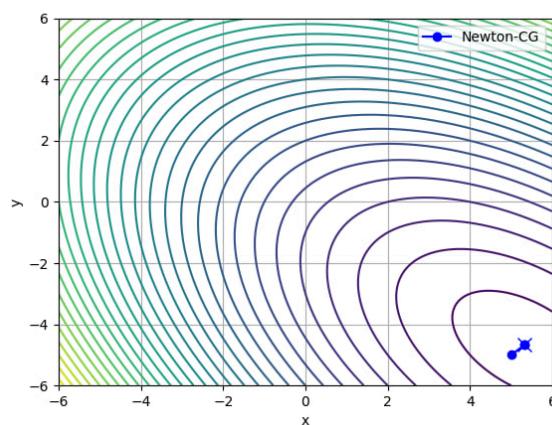
<i>Реализация</i>	<i>start</i>	<i>method</i>	<i>final</i>	<i>min</i>	<i>iter</i>	<i>grad count</i>	<i>hessian count</i>
Наша	(5, -5)	GD	(4.999, -5.000)	-0.0004	200	200	100
Scipy	(5, -5)	CG	?	-0.999	13	55	?
Наша	(5, -5)	GSS	(4.999, -5.000)	-0.0004	200	200	100
Scipy	(5, -5)	CG	?	-0.999	13	55	?
Наша	(2, 2)	GSS	(1.474, 1.763)	0.0511	224	214	100
Scipy	(2, 2)	CG	?	-0.999	7	30	?
Наша	(5, -5)	GD	(1.474, 1.763)	0.0511	214	214	100
Scipy	(5, -5)	CG	?	-0.999	7	30	?



Optuna:

<i>method</i>	<i>scheduler</i>	<i>learning rate</i>	<i>eps</i>	<i>noise scale</i>	<i>best result</i>
GD	<i>decay</i>	0.964217	1.03659e-05	0.092046	-0.999

$$(x - 3)^2 + (y + 2)^2 + x * y$$



Реализация	start	method	final	min	iter	grad count	hessian count
Наша	(5, -5)	GD	(5.1155 1378, -4.8844 8618)	-12.190 997284 862938	200	200	100
Scipy	(5, -5)	CG	?	-12.333 332720 24906	2	5	?
Наша	(5, -5)	DGD	(5, -5)	-12	2	2	1
Scipy	(5, -5)	CG	?	-12.333 332720	2	5	?

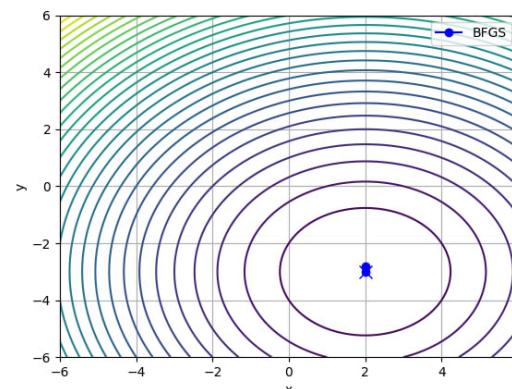
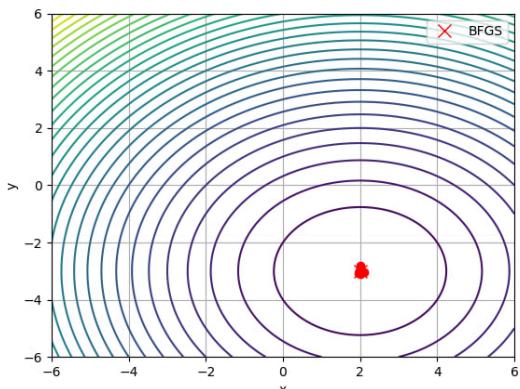
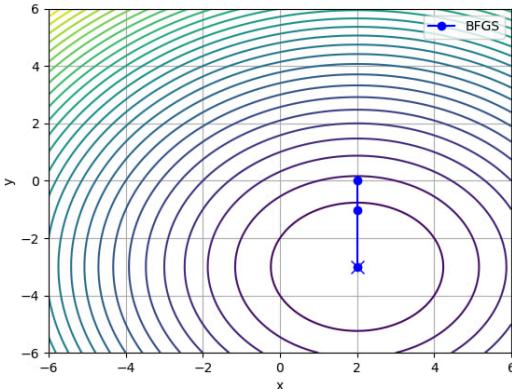
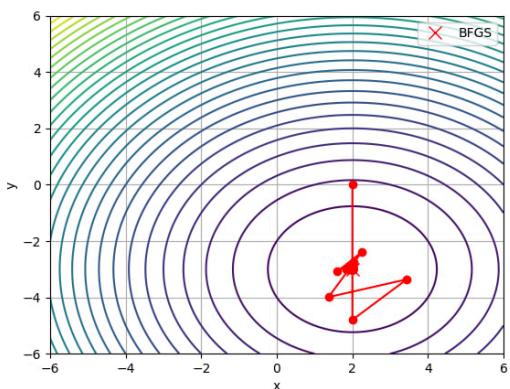
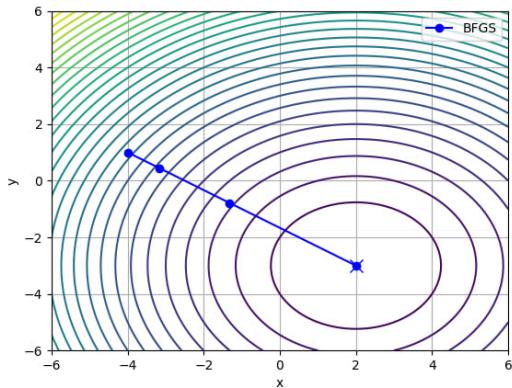
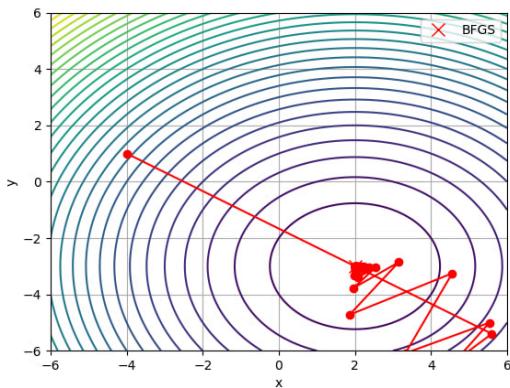
				24906			
Наша	(2, 2)	DGD	(5.2777 9043, -4.5556 3909)	-12.324 087989 648655	26	8	2
Scipy	(2, 2)	CG	?	-12.333 332745 300787	4	97	?
Наша	(-3, -1)	DGD	(5.3137 4189, -4.6581 3173)	-12.333 043875 239692	12	5	2
Scipy	(-3, -1)	CG		-12.333 333147 372304	2	86	

Optuna:

<i>method</i>	<i>scheduler</i>	<i>learning rate</i>	<i>eps</i>	<i>noise scale</i>	<i>best result</i>
GD	exponential	0.9785595 20057319 8	4.7763685 14966029 e-05	0.0675191 92345876 44	-12.33333 33333333 36

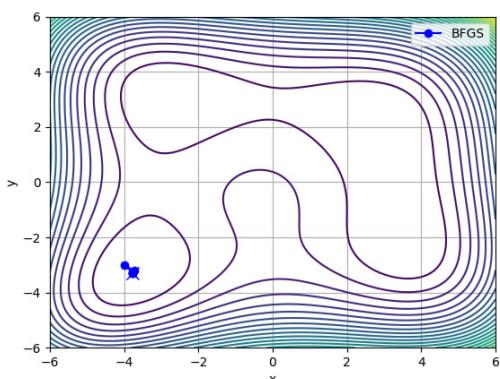
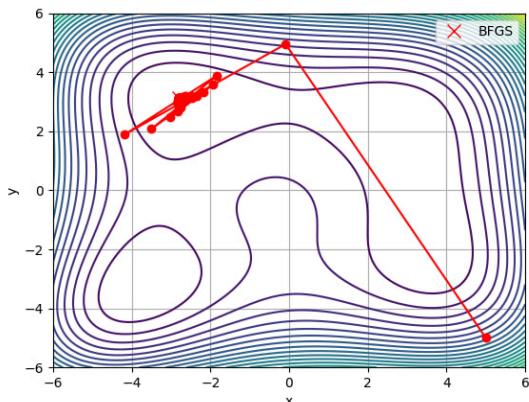
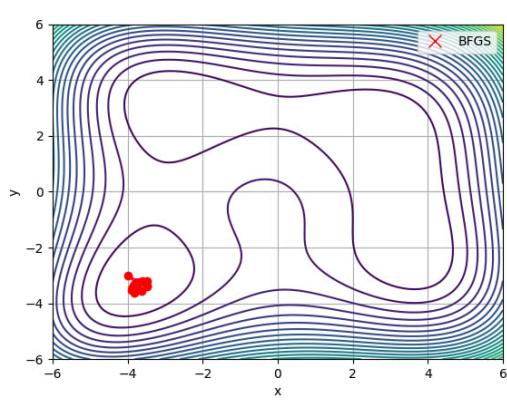
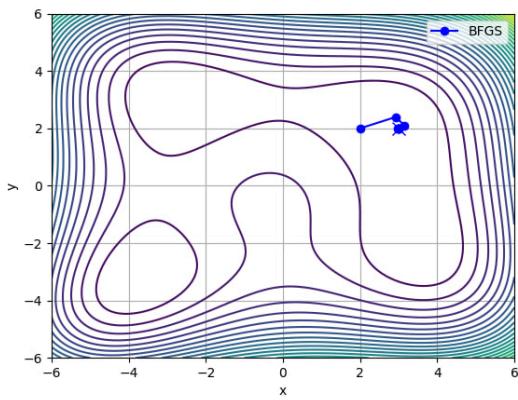
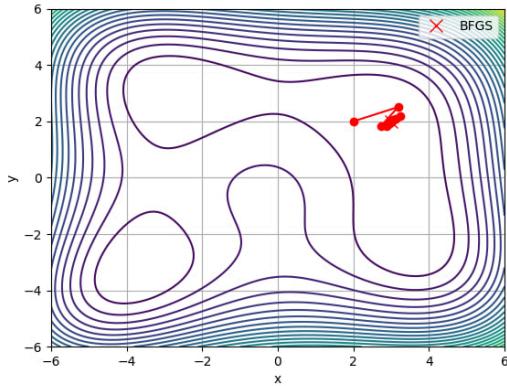
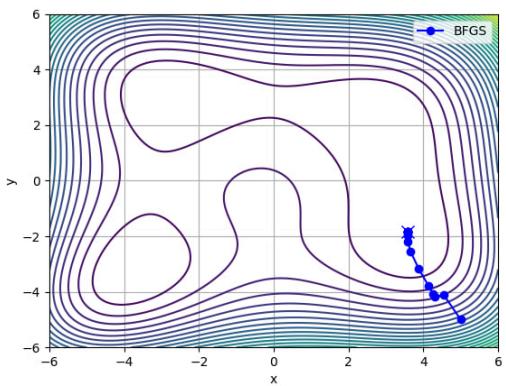
BFGS: наш и библиотечный

Количество вычислений гессиана, ожидаемо, равно нулю.



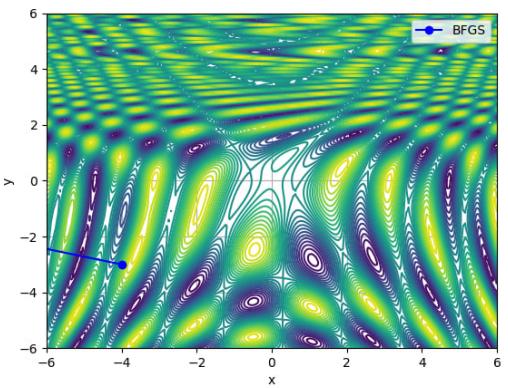
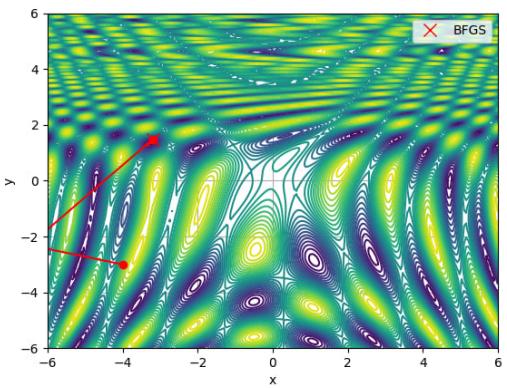
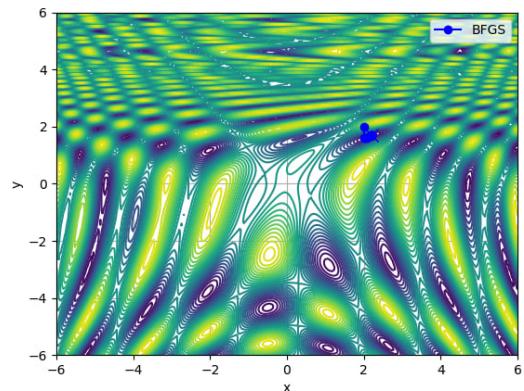
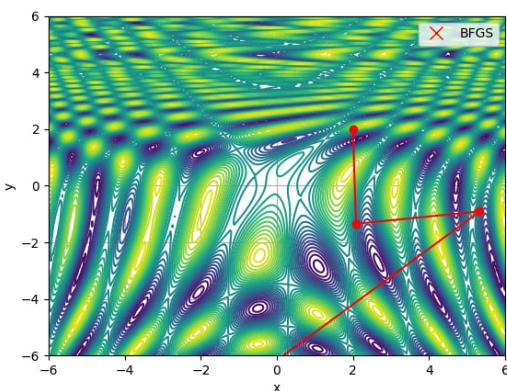
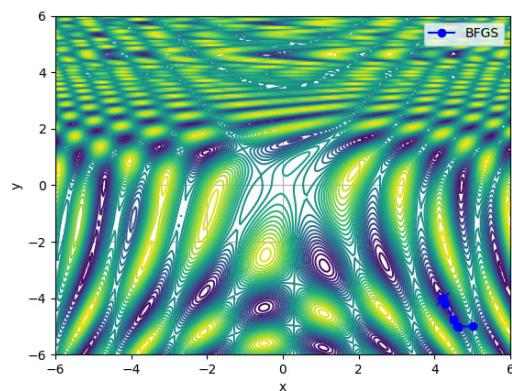
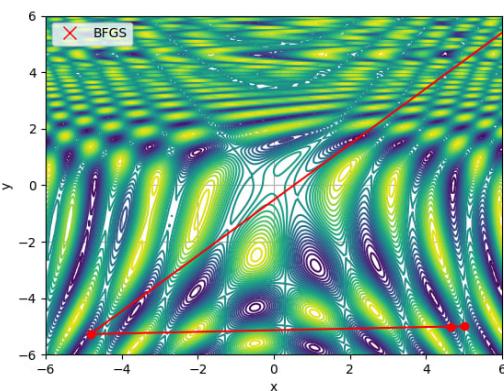
Реализация	start	final	min	iter	grad count
Наша	(-4, 1)	(2.000005, -3.000026)	7.09e-10	60	122
Scipy	(-4, 1)	(2.000002 65,-2.9999 9588)	2.39e-11	3	4

Наша	(2, 0)	(1.999965 29,-2.9999 9153)	1.27e-09	49	100
Scipy	(2, 0)	(1.999999 91,-2.9999 9998)	8.08e-15	2	3
Наша	(2.0, -2.8)	(1.999927 26,-2.9999 8641)	5.47e-09	31	64
Scipy	(2.0, -2.8)	(1.999999 99,-3.0000 0000)	6.96e-17	1	3

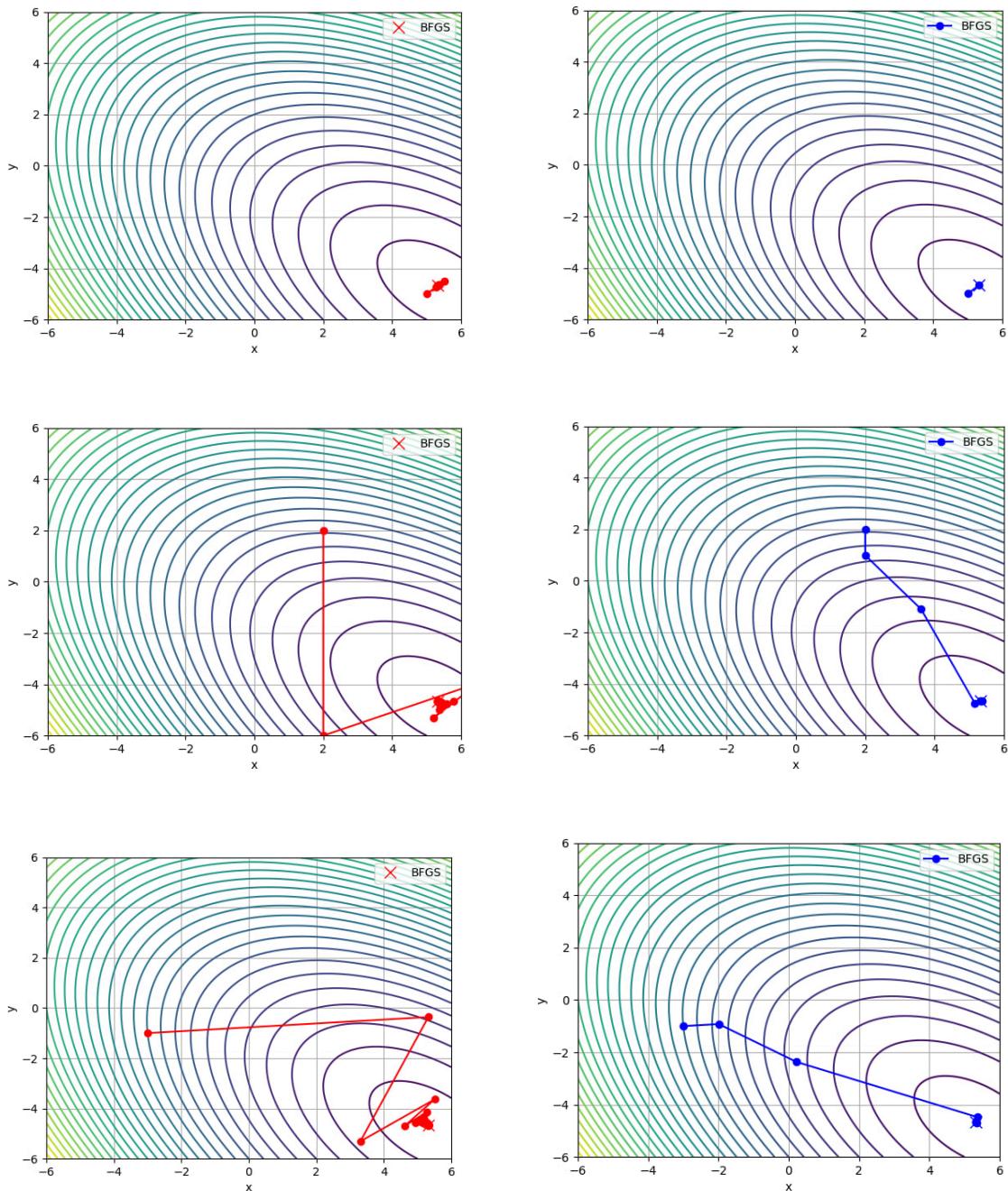


Реализация	<i>start</i>	<i>final</i>	<i>min</i>	<i>iter</i>	<i>grad count</i>
Наша	(5, -5)	(-2.80498, 3.128889)	0.000236	100	200
Scipy	(5, -5)	(3.58442, -1.848)	5.473e-15	15	19

Наша	(2, 2)	(4.999, -5.000)	8.77e-11	100	200
Scipy	(2, 2)	(3, 2)	4.889e-14	7	9
Наша	(-4, -3)	(-3.779, -3.2833)	2.653e-07	100	200
Scipy	(4, -3)	(-3.779, -3.283)	5.32e-15	5	8



<i>Реализация</i>	<i>start</i>	<i>final</i>	<i>min</i>	<i>iter</i>	<i>grad count</i>
Наша	(5, -5)	(-69902.22 0,-11712.4 63)	-0.829251	100	200
Scipy	(5, -5)	(4.221296 23,-4.0277 4861)	-0.99999	8	16
Наша	(2, 2)	(-16.12243 122,-22.07 82021)	-0.97954	4	10
Scipy	(2, 2)	(2.201787, 1.6870605 6)	-0.99999	8	11
Наша	(-4, -3)	(-3.202013 55,1.4765 9539)	-0.90225	2	6
Scipy	(-4, -3)	(-8.576554 34,-1.6489 1663)	-0.79192	7	15



Реализация	start	final	min	iter	grad count
Наша	$(5, -5)$	$(5.333, -4.66664)$	-12.333	11	24
Scipy	$(5, -5)$	$(5.333, -4.66667)$	-12.333	1	3
Наша	$(2, 2)$	$(5.333, -4.66667)$	-12.333	71	144

		-4.66667)			
Scipy	(2, 2)	(5.333, -4.66667)	-12.333	7	9
Наша	(-3, -1)	(5.333, -4.66666)	-12.333	100	200
Scipy	(-3, -1)	(5.333, -4.66667)	-12.333	6	7

Выводы

Эффективность методов второй лабораторной по сравнению с методами первой лабораторной сильно зависит от входных параметров даже на относительно простых функциях. На простых функциях библиотечные реализации выполняются значительно быстрее, однако на сложных функциях (см. функция 3) в зависимости от стартовой точки, благодаря хаотичной природе функции, наши методы могут сойтись и быстрее и к лучшему локальному минимуму. Методы библиотеки *optuna* для подбора гиперпараметров показывают исключительную результативность: даже на сложных функциях всегда находится набор гиперпараметров, позволяющий достичь хорошего минимума. Также становится видно, что разные методы изменения *learning rate* могут быть оптимальны для различных функций (см. функция 4, оптимальный *scheduler* отличается от других).