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

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

курса «Методы оптимизации»

Работу выполнили:

Файзиева Юлия M32331
Петрова Анаастасия M32341
Надеждин Игорь M32331

Преподаватель:

Свинцов М. В.

Санкт-Петербург
2023

Содержание

Лабораторная работа 1

Лабораторная работа

1.1. Градиентный спуск с постоянным шагом

1.1.1. Теоретическая часть

Основная идея метода заключается в том, чтобы осуществлять оптимизацию в направлении наискорейшего спуска, а это направление задаётся антиградиентом -∇f.

Ввод: функция f: $\mathbb{R}^n \to \mathbb{R}$. Вывод: найденная точка минимума функции.

Алгоритм:

Если не выполнен критерий останова, выполняется: $x_{k+1} = x_k - \lambda_k \nabla f(x_k)$.

Иначе, возвращается x_{k+1}

За критерий останова взят: $f(x_{k+1}) - f(x_k) \leq \varepsilon$ За ε - 10^{-5} За λ - 10^{-2}

1.1.2. Реализация метода

Листинг 1.1: Градиентный спуск и посторение исследуемого графика

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (7, 7)

def f(arg):
    return arg[0] ** 2 + 30 * arg[1] ** 2

def grad(x):
    h = 1e-5
    return (f(x[:, np.newaxis] + h * np.eye(2)) - f(x[:, np.newaxis] -
    h * np.eye(2))) / (2 * h)

def simple_gradient_descents(list_range, eps, step):
    k = 0
    x = np.array(list_range)
    function = []
    p = []
    count = 0
    function.append(f(x))
    p.append(x)
    while True:
        x1 = x - step * np.array(grad(x))
        function.append(f(x1))
        p.append(x1)
        if abs(f(x) - f(x1)) >= eps:
```

```
                k = k + 1
                count = count + 1
                x = x1
            else:
                print("number of gradient evaluations: ", k * 2)
                print("number of function evaluations: ", count)
                print(x1, " ", '{:f}'.format(f(x1)))
                t = np.linspace(-10, 10, 100)
                x_1, y_1 = np.meshgrid(t, t)
                x = np.array(p)
                plt.plot(x[:, 0], x[:, 1], '-o')
                plt.contour(x_1, y_1, f([x_1, y_1]), levels=sorted([f(p)
    for p in x]))
                ax = plt.figure().add_subplot(projection='3d')
                ax.plot_surface(x_1, y_1, f([x_1, y_1]))
                plt.show()
                break

def main():
    # input
    list_range = [1, 2]  # input

    simple_gradient_descents(list_range, 1e-5, 1e-2)

if __name__ == "__main__":
    main()
```

1.2. Метод одномерного поиска (Метод дихотомии)

Данный метод описывает нахождение корней функции. Осуществляется поиск минимума функции.

Метод дихотомии:

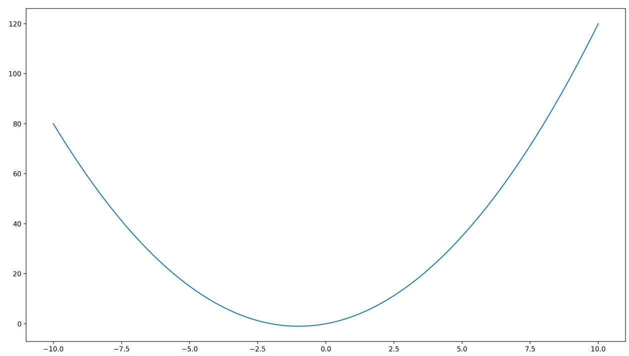
Дано: $f(x) : [a, b] \longrightarrow \mathbb{R}$, $f(x) \in C[a, b]$.
1) Находим половину интервала $[a, b] = m = \frac{a+b}{2}$ и вычисляем две симметричные от m точки $x_1 = m - \delta$ b $x_2 = m + \delta$, где δ - число в интервале $[0, \frac{b-a}{2}]$, возьмем $\delta = \varepsilon$
2) Если $f(x_1) <= f(x_2)$, то корень лежит на отрезке $[a, x_1]$.
3) Иначе, нужно искать корень на отрезке $[x_2, b]$.
4) Ответ - середина последнего отрезка.
5) Критерий останова: длина отрезка равна удвоенному значению ε. Возьмем $\varepsilon = 1e - 5$.

Листинг 1.2: Метод дихотомии

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (4, 4)
```

```

def f(x):
    return x ** 2 + 2 * x

def diff(x):
    h = 1e-5
    return (f(x[:, np.newaxis] + h * np.eye(1)) - f(x[:, np.newaxis] - h * np.eye(1))) / (2 * h)

def dichotomy(eps):
    a, b = 0.001, 10
    delta = eps
    while (b - a) / 2 >= eps:
        merge = (a + b) / 2
        x1 = merge - delta
        x2 = merge + delta
        f1, f2 = f(x1), f(x2)
        if f1 <= f2:
            b = x1
        else:
            a = x2
    t = np.linspace(-10, 10, 100)
    plt.plot(t, f(t))
    plt.show()
    return (a + b) / 2

def main():
    min_of_f = dichotomy(1e-5)
    print("min: ", min_of_f, ", f in min:", '{:f}'.format(f(min_of_f)))

if __name__ == "__main__":
    main()
```

Рис. : Исследуемая функция: $f(x) = x^2 + 2 \cdot x$



1.3. Градиентный спуск на основе метода дихотомии

Градиентный спуск на основе метода дихотомии заключается в следующем: Дано: $f(x) : [a, b] \longrightarrow \mathbb{R}$, $f(x) \in C[a, b]$.
1) Находим половину интервала $[a, b] = m = \frac{a+b}{2}$ и вычисляем две симметричные от m точки $x_1 = m - \delta$ b $x_2 = m + \delta$, где δ - число в интервале $[0, \frac{b-a}{2}]$, возьмем $\delta = \varepsilon$
x_1 и x_2 - шаг для градиентного спуска и рассчитываем градиент. Каждый раз выбираем оптимальный шаг.
2) Если $f(x_1) <= f(x_2)$, то берем отрезок $[a, x_1]$.
3) Иначе - $[x_2, b]$.
4) Ответ - $x = x - \frac{a+b}{2} \nabla f(x)$. 5) Критерий останова: длина отрезка равна удвоенному значению ε. Возьмем $\varepsilon = 1e - 5$.

1.4. Градиентный спуск на основе метода дихотомии

Листинг 1.3: Метод градиентного спуска на основе дихотомии

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (7, 7)

def f(x):
    return x[0] ** 2 + 0.1 * x[1] ** 2

def grad(x):
    h = 1e-5
    return (f(x[:, np.newaxis] + h * np.eye(2)) - f(x[:, np.newaxis] -
    h * np.eye(2))) / (2 * h)

def gradient_descent_with_dichotomy(eps):
    a, b = 0.001, 10
    x = np.array([1, 2])
    p = [x]
    delta = eps
    count, k = 0, 0
    gradient_in_x = np.array(grad(x))
    while (b - a) / 2 >= eps:
        merge = (a + b) / 2
        x1 = merge - delta
        x2 = merge + delta
        point_1 = x - x1 * gradient_in_x
        point_2 = x - x2 * gradient_in_x
        f1, f2 = f(point_1), f(point_2)
        count = count + 2
        min_point = x - merge * gradient_in_x
        p.append(min_point)
        if f1 <= f2:
```

```
            b = x1
        else:
            a = x2
    min_point = x - (a + b) / 2 * gradient_in_x
    p.append(min_point)
    print("number of gradient evaluations: ", 1)
    print("number of function evaluations: ", count)
    print(" x and y :", min_point, " min:", f(min_point))
    t = np.linspace(-10, 10, 100)
    x_1, y_1 = np.meshgrid(t, t)
    x = np.array(p)
    plt.plot(x[:, 0], x[:, 1], '-o')
    plt.contour(x_1, y_1, f([x_1, y_1]), levels=sorted([f(p) for p in
  x]))
    ax = plt.figure().add_subplot(projection='3d')
    ax.plot_surface(x_1, y_1, f([x_1, y_1]))
    plt.show()
    return (a + b) / 2

def main():
    gradient_descent_with_dichotomy(1e-5)

if __name__ == "__main__":
    main()
```

1.5. Одномерный поиск с учетом условий Вольфе

Задано: начальное приближение функции, ее градиент.
Чтобы найти новое приближение $x_{k+1} = x_k + \alpha_k p_k$ необходимо найти α, удовлетворяющее следующим условиям Вольфе:
1) $f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k$
2) $f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k$
Константы выбираются следующим образом: $0 < c_1 < c_2 < 1$

Листинг 1.4: Одномерный поиск с учетом условий Вольфе

```
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams["figure.figsize"] = (4, 4)
def f(x):
    return x ** 2 + 2 * x

def diff(x):
    return 2 * x + 2

def line_search(x, p, count):
    alpha = 1
    c_1 = 1e-4
    c_2 = 0.9
```

```
    for m in range(1, 100):
        alpha = alpha * 0.5
        if f(x + alpha * p) <= f(x) + c_1 * alpha * diff(x) * p and \
                diff(x + alpha * p) * p >= c_2 * diff(x) * p:
            break
        count += 2
    return alpha, count

def simple_gradient_descents(x, eps):
    p = [x]
    count = 0
    while True:
        alpha, count = line_search(x, -diff(x), count)
        x1 = x - alpha * (diff(x))
        p.append(x1)
        count = count + 1
        if abs(f(x) - f(x1)) >= eps:
            x = x1
        else:
            print("min:", x, " f(min):", f(x))
            print("number of gradient evaluations: ", count)
            x = np.array(p)
            plt.plot(x)
            plt.show()
            break

def main():
    simple_gradient_descents(5, 1e-5)

if __name__ == "__main__":
    main()
```

1.6. Генератор случайных квадратичных функций n переменных с числом обусловленности k

Генерация квадратичной функции с помощью случайной квадратной матрицы, диагональной матрицы с максимумом равным числу обусловленности. $x^T Ax$

Листинг 1.5: Генератор

```
def getFbyArg(args):
    return args.T @ quadratic_form @ args


def gradByQuad(x):
    return (quadratic_form + quadratic_form.T) @ x

def generate(n, k):
```

```
    matrix = np.random.rand(n, n)
    orthonormal, _ = np.linalg.qr(matrix)
    m = np.random.uniform(1, k, n)
    diag_matrix = np.diagflat(m)
    diag_matrix[0][0] = 1
    diag_matrix[n - 1][n - 1] = k
    inverse = orthonormal.T
    return orthonormal @ diag_matrix @ inverse
```

1.7. Исследование зависимости числа итераций T(n, k), необходимых градиентному спуску для сходимости в зависимости от размерности пространства 2 n 103 и числа обусловленности оптимизируемой функции 1 k 103

Листинг 1.6: Tester

```
def tester():
    dimensions = []
    condition_number = []
    iterations = []
    for i in range(2, 10):
        for j in range(1, 100, 20):
            dimensions.append(i)
            condition_number.append(j)
            global quadratic_form
            quadratic_form = generate(i, j)
            count = gradient_descent_with_dichotomy(1e-5, i)
            iterations.append(count)
    index = []
    for i in range(len(dimensions)):
        index.append(i)
    data = {'index': index,
            'n': dimensions,
            'k': condition_number,
            'count': iterations
            }
    pd.set_option('display.max_rows', None)
    pd.set_option('display.max_columns', None)
    results_df = pd.DataFrame(data)
    pivoted_df = results_df.pivot_table(index='index', values=['n', 'k', 'count'])

    pivoted_df.to_csv('results.csv')
```

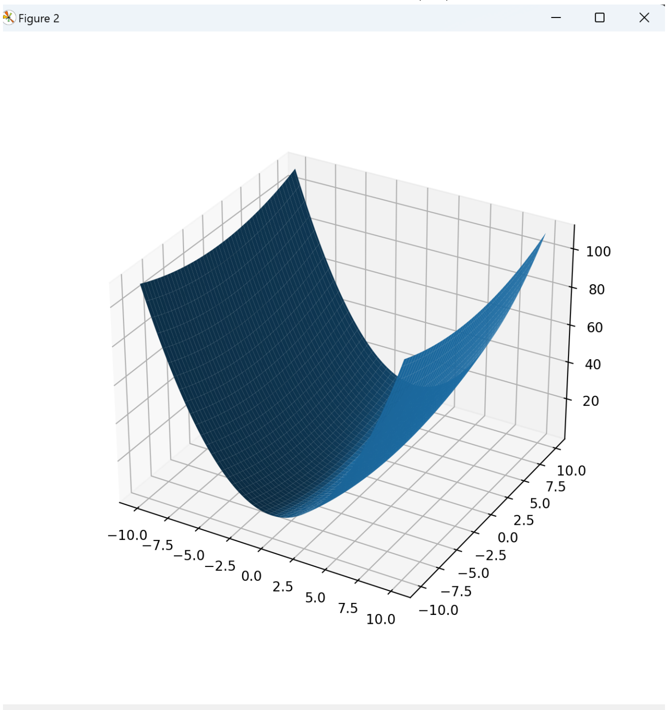
Вывод: зависимость числа операций градиентного спуска от числа обусловленности прямая

1.8. Полученные результаты и их анализ

Рассмотрим несколько функций, на примере которых работа методов будет отличаться.

Функция $x^2 + 0.1 \cdot y^2$
У данной функции при $\varepsilon = 1e - 2$ (оптимальном шаге для большинства функций) плохая сходимость, но при подборе шага (например 0.8) он сходится. Он проигрывает градиентному спуску с применением метода дихотомии по количеству вычислений градиента. Тем не менее он выгоднее с точки зрения количества вычислений функции.

Рис. : График функции: $f(x) = x^2 + 0.1 \cdot y^2$



Функция $x^2 + 2 \cdot y^2 - 4 \cdot x + 4 \cdot y + 3$

У данной функции точно также при подборе оптимального шага метод градиентного спуска с постоянным шагом выгоднее с точки зрения количества вычислений функции, но не градиента.

Сходимость градиентного спуска главным образом зависит от выбора шага. Его выбор не очевиден в силу отсутствия изначальной информации о минимизируемой функции. Если его сделать малым, то метод будет сходиться медленно, а его увеличение может привести к расходимости. Также на сходимость влияет масштабтрование осей, часто необходимо увеличение для ее рассмотрения.

От начального приближения может зависеть количество вычислений градиента и функции, но это не отражается на сходимости и вычисленном значении минимума функции.

Рис. : Градиентный спуск на основе дихотомии для $f(x) = x^2 + 0.1 \cdot y^2$

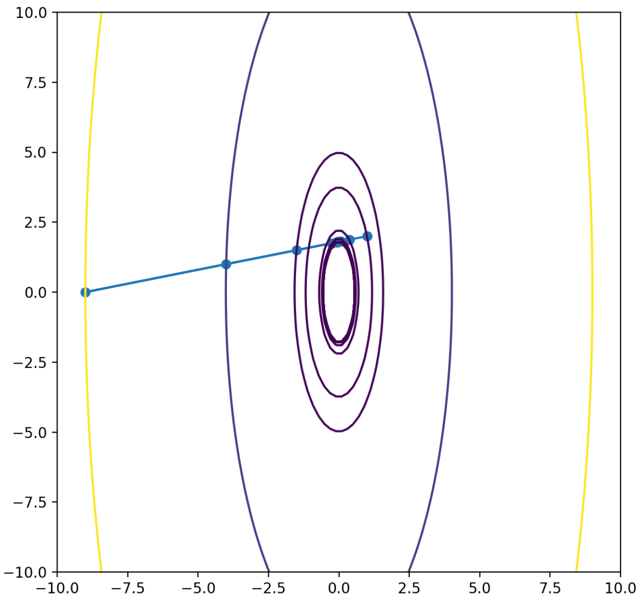
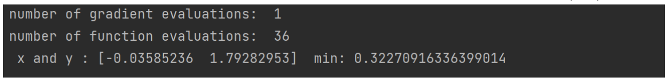


Рис. : Результат работы с дихотомией для $f(x) = x^2 + 0.1 \cdot y^2$



```
number of gradient evaluations:  1
number of function evaluations:  36
 x and y : [-0.03585236  1.79282953]  min: 0.32270916336399014
```

Можно заметить, что одномерный поиск с учетом условий Вольфе оптимален с точки зрения количества вычислений градиента, также он быстро находит минимум рассматриваемой функции.

Рис. : Без дихотомии $f(x) = x^2 + 0.1 \cdot y^2$

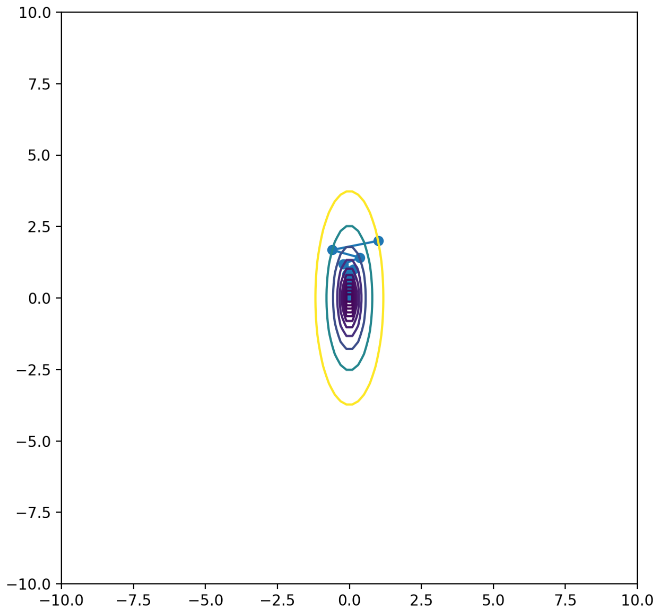


Рис. : Результат работы без дихотомии $f(x) = x^2 + 0.1 \cdot y^2$

```
number of gradient evaluations:  54
number of function evaluations:  27
[6.14094221e-07 1.51652651e-02]   0.000023
```

Рис. : График функции: $f(x) = x^2 + 2 \cdot y^2 - 4 \cdot x + 4 \cdot y + 3$

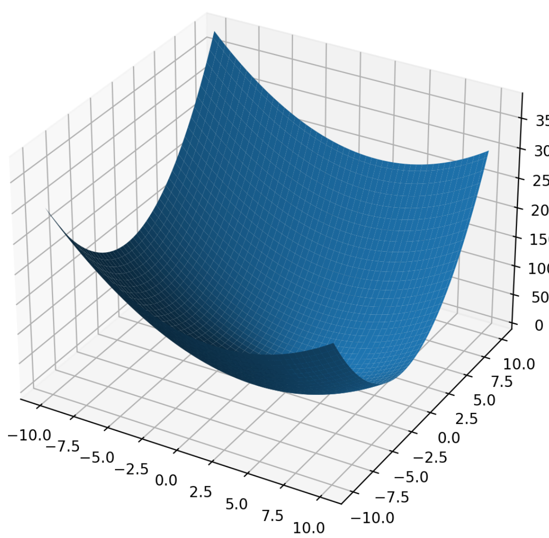


Рис. : Градиентный спуск на основе дихотомии для $f(x) = x^2 + 2 \cdot y^2 - 4 \cdot x + 4 \cdot y + 3$

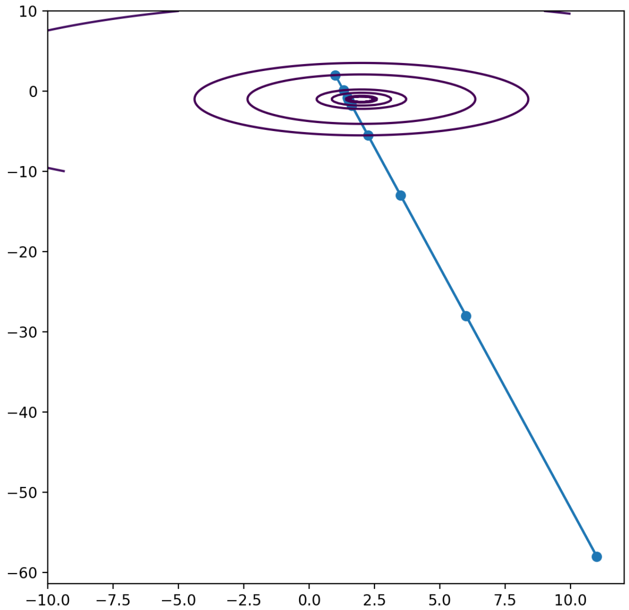


Рис. : Результат работы с дихотомией для $f(x) = x^2 + 2 \cdot y^2 - 4 \cdot x + 4 \cdot y + 3$

```
number of gradient evaluations:  1
number of function evaluations:  36
 x and y : [ 1.50688199 -1.04129195]  min: -2.7534245795907992
```

Рис. : Без дихотомии $f(x) = x^2 + 2 \cdot y^2 - 4 \cdot x + 4 \cdot y + 3$

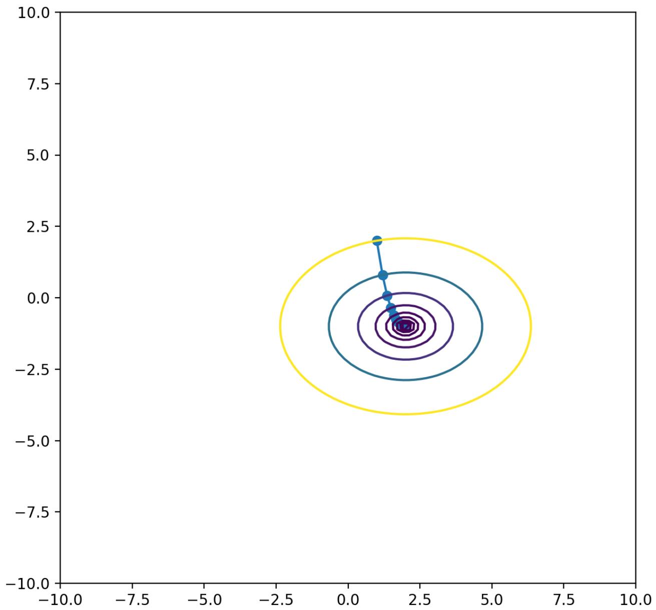


Рис. : Результат работы без дихотомии $f(x) = x^2 + 2 \cdot y^2 - 4 \cdot x + 4 \cdot y + 3$

```
number of gradient evaluations:  48
number of function evaluations:  24
[ 1.99622211 -0.99999147]   -2.999986
```

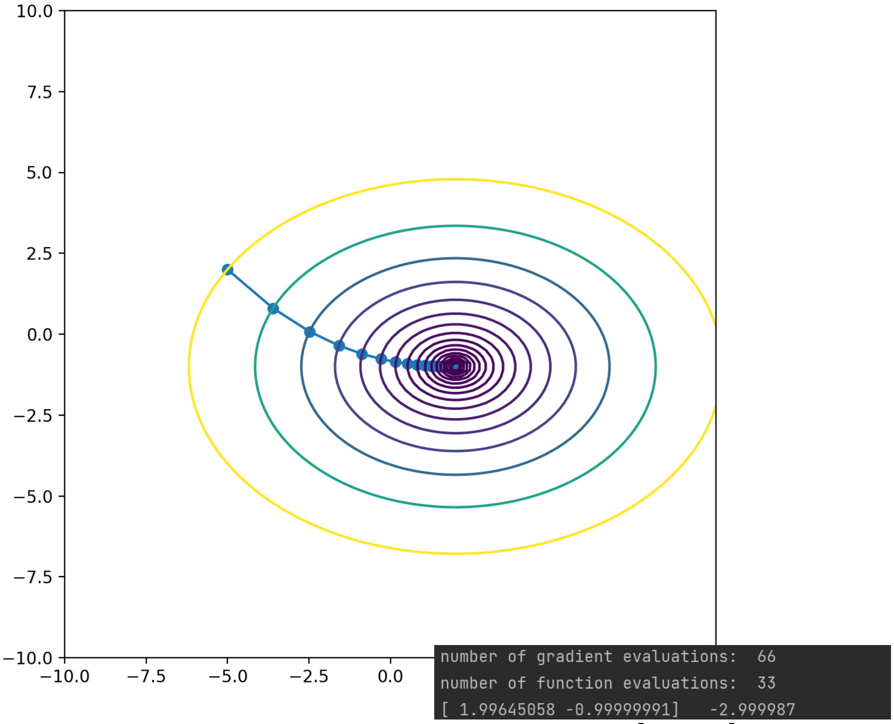


Рис. : Начальное приближение $[-5; 2]$

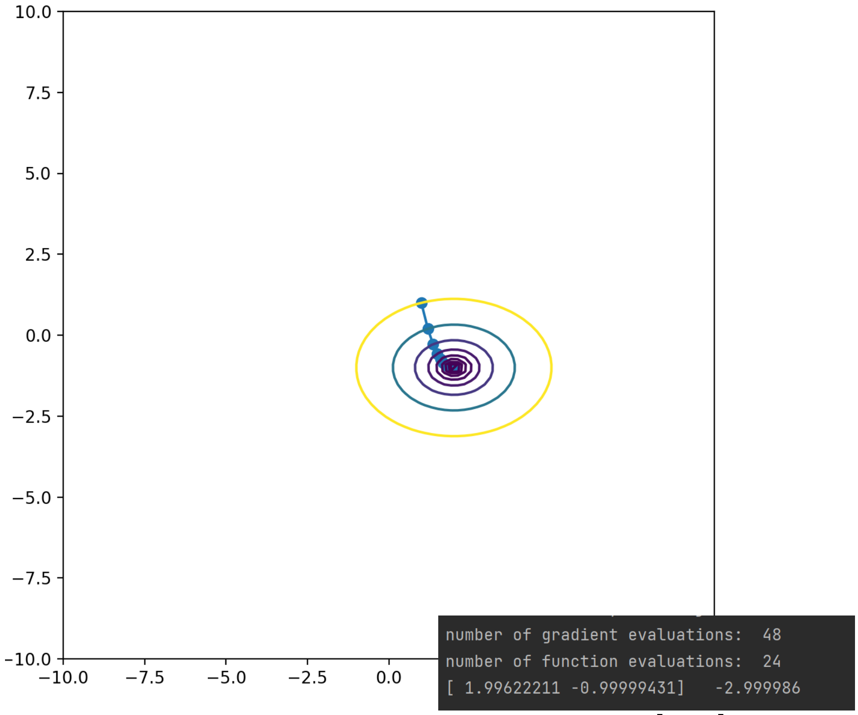
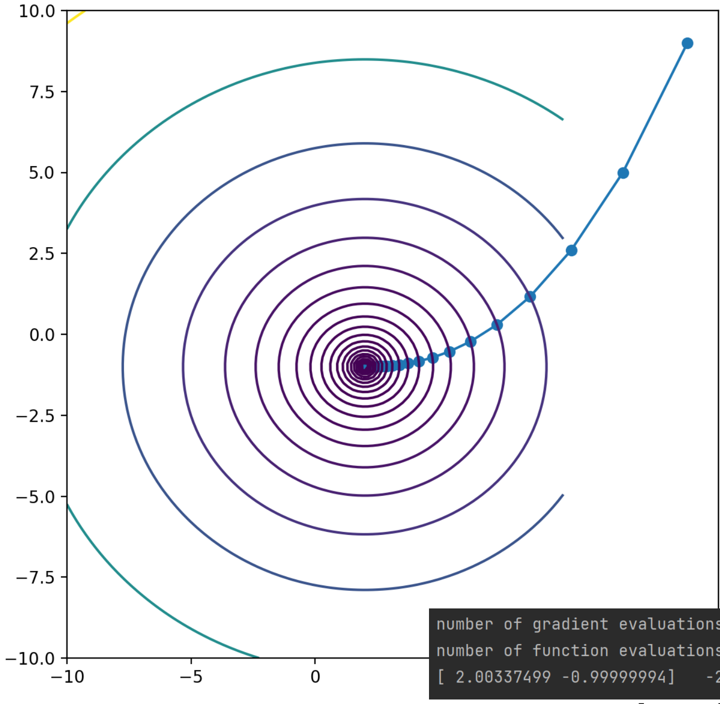


Рис. : Начальное приближение $[1; 1]$




```
number of gradient evaluations:  72
number of function evaluations:  36
[ 2.00337499 -0.99999994]   -2.999989
```

Рис. : Начальное приближение $[15; 9]$

Рис. : одномерный поиск с учетом условий Вольфе $f(x) = x^2 + 2 \cdot x$

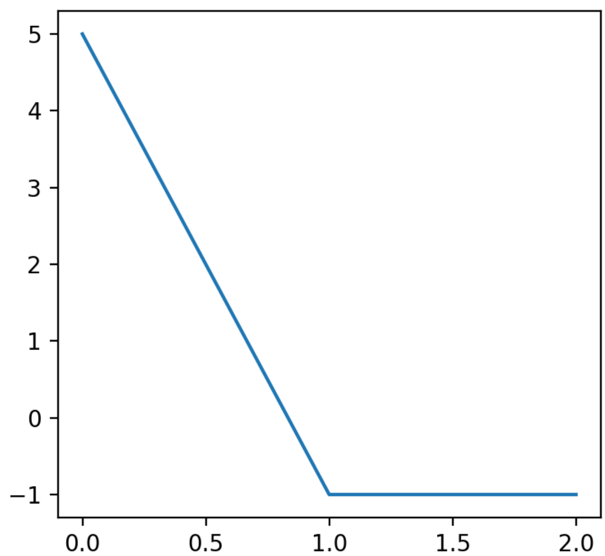


Рис. : одномерный поиск с учетом условий Вольфе $f(x) = x^2 + 2 \cdot x$

```
min: -1.0  f(min): -1.0
number of gradient evaluations:  2
```

index	count	k	n
0	16311	1	2
1	7553	21	2
2	1726	41	2
3	11495	61	2
4	10982	81	2
5	23265	1	3
6	16197	21	3
7	12271	41	3
8	11921	61	3
9	13345	81	3
10	24653	1	4
11	19651	21	4
12	15287	41	4
13	9755	61	4
14	14142	81	4
15	30053	1	5
16	16854	21	5
17	4720	41	5
18	17228	61	5
19	14698	81	5
20	30162	1	6
21	9692	21	6
22	26096	41	6
23	18594	61	6
24	16624	81	6
25	31074	1	7
26	27139	21	7
27	27155	41	7
28	24021	61	7
29	9347	81	7
30	31870	1	8
31	13889	21	8
32	32414	41	8
33	28918	61	8
34	19015	81	8
35	32578	1	9
36	29445	21	9
37	25624	41	9
38	33066	61	9
39	31809	81	9