Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
"ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Нигматуллин Роман Максимович

Лабораторная работа №9

Минимизация функций

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Общие функции для ЛР на Python

```python
import numpy as np
from typing import Callable


def deriv(func: Callable[[float], float],
          point: float,
          eps: float = 1e-5) -> float:
    return (func(point + eps) - func(point - eps)) / (2 * eps)


def deriv2(func: Callable[[float], float],
           point: float,
           eps: float = 1e-5) -> float:
    return (func(point + eps) - 2 * func(point) + func(point - eps)) / (eps ** 2)


def grad(f: Callable[[np.array], np.array],
         x: np.array,
```

```
         eps: float = 1e-5) -> np.array:
    dim = len(x)
    grad_vector = np.zeros((dim, ), dtype=np.double)
    for i in range(dim):
        delta = np.zeros(dim)
        delta[i] += eps
        grad_vector[i] = (f(x + delta) - f(x - delta)) / (eps * 2)
    return grad_vector


def hessian(f: Callable[[np.array], np.array],
            x: np.array,
            eps: float = 1e-5) -> np.array:
    dim = len(x)
    hess = np.zeros((dim, dim), dtype=np.double)
    for i in range(dim):
        i_d = np.zeros(dim)
        i_d[i] += eps
        for j in range(dim):
            j_d = np.zeros(dim)
            j_d[j] += eps
            hess[i, j] = (f(x - i_d - j_d) - f(x + i_d - j_d)
                          - f(x - i_d + j_d) + f(x + i_d + j_d)
                          ) / (4 * eps ** 2)
    return hess


def point_in_area(point, bbox):
    return bbox[0][0] < point[0] < bbox[1][0]\
           and bbox[0][1] < point[1] < bbox[1][1]


def newton_optimize_vec(func: Callable[[np.array], np.array],
                        bbox: tuple[np.array, np.array],
                        start: np.array,
                        eps: float = 1e-5,
                        minimize: bool = True) -> tuple[np.array, int]:
    x = start.astype(np.double)
    point_grad = grad(func, x)
    iter_cnt = 0
    while point_in_area(x, bbox) and np.linalg.norm(point_grad) > eps:
        step = np.linalg.inv(hessian(func, x)).dot(point_grad)
        x -= step if minimize else -step
        point_grad = grad(func, x)
        iter_cnt += 1
    x[0] = max(min(bbox[1][0], x[0]), bbox[0][0])
    x[1] = max(min(bbox[1][1], x[1]), bbox[0][1])
    return x, iter_cnt
```

```
def conjucate_grad_minimize(func: Callable[[np.array], np.array],
                            start: np.array,
                            eps: float = 1e-5) -> tuple[np.array, int]:
    x = start.astype(np.double)
    x_prev, h_prev = None, None
    iter_cnt = 0
    h = -grad(func, x)
    while np.linalg.norm(grad(func, x)) > eps:
        h = -grad(func, x)
        if h_prev is not None and x_prev is not None:
            denominator = np.linalg.norm(grad(func, x_prev))
            beta = (np.linalg.norm(grad(func, x)) / denominator) ** 2
            h += beta * h_prev
        alpha, _ = newton_optimize_scal(lambda a: func(x + a * h),
                                        interval=(-5, 5),
                                        start=0,
                                        eps=eps * 1e-2)
        x_prev = x.copy()
        x += alpha * h
        h_prev = h.copy()
        iter_cnt += 1

    return x, iter_cnt


def conjucate_grad_quadratic_minimize(func: Callable[[np.array], np.array],
                                      func_matrix: np.array,
                                      start: np.array,
                                      eps: float = 1e-5) -> tuple[np.array, int]:
    x = start.astype(np.double)
    h_prev = None
    iter_cnt = 0
    h = -grad(func, x)
    while np.linalg.norm(grad(func, x)) > eps:
        h = -grad(func, x)
        if h_prev is not None:
            denominator = ((func_matrix @ h_prev) @ h_prev)
            beta = ((func_matrix @ h_prev) @ grad(func, x)) / denominator
            h += beta * h_prev
        alpha, _ = newton_optimize_scal(lambda a: func(x + a * h),
                                        interval=(-10, 10),
                                        start=0,
                                        eps=eps * 1e-2)
        x += alpha * h
        h_prev = h.copy()
        iter_cnt += 1

    return x, iter_cnt
```

```python
def newton_optimize_scal(func: Callable,
                         interval: tuple[float, float],
                         start: float,
                         eps: float = 1e-5,
                         minimize: bool = True) -> tuple[float, int]:
    (a, b), x = interval, start
    cnt = 0
    while a <= x <= b and abs(deriv(func, x)) > eps:
        step = deriv(func, x) / deriv2(func, x)
        x += -step if minimize else step
        cnt += 1
    x = min(b, max(a, x))
    return x, cnt


def fibonacci(func: Callable,
              interval: tuple[float, float],
              eps: float = 1e-5,
              minimize: bool = True) -> tuple[float, int]:
    a, b = interval
    numbers, f = [1, 1], 2
    d = b - a
    while f <= (b - a) / eps:
        numbers.append(f)
        f = numbers[-1] + numbers[-2]
    n = len(numbers)
    for k in range(1, n):
        d *= numbers[n - k - 1] / numbers[n - k]
        x1 = b - d
        x2 = a + d
        cond = func(x1) <= func(x2) if minimize else func(x1) >= func(x2)
        if cond:
            b = x2
        else:
            a = x1
    return (a + b) / 2, n
```

2 Минимум унимодальной функции на отрезке (9.1.16)

2.1 Формулировка задачи

Методом Ньютона найти минимум и максимум унимодальной на отрезке $[a, b]$ функции $f(x)$ с точностью. Предусмотреть подсчет числа итераций, потребовавшихся для достижения заданной точности.

2.2 Вариант

$$f(x) = 3cos^2(x) - \sqrt{x}$$

$$[a, b] = [0, 3]$$

2.3 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt

from optimize import newton_optimize_scal


def f(t):
    return 3 * np.cos(t) ** 2 - np.sqrt(t)


interval = (0 + 1e-5, 3)

minima, iter_min = newton_optimize_scal(func=f,
                                        interval=interval,
                                        start=1.5,
                                        eps=1e-6,
                                        minimize=True)
maxima, iter_max = newton_optimize_scal(func=f,
                                        interval=interval,
                                        start=1.0,
                                        eps=1e-6,
                                        minimize=False)
right_maxima, right_iter_max = newton_optimize_scal(func=f,
                                                    interval=interval,
                                                    start=2.0,
                                                    eps=1e-6,
                                                    minimize=False)
t = np.linspace(start=interval[0], stop=interval[1], num=100)
y = f(t)

fig, ax = plt.subplots(figsize=(8, 5))
plt.plot(t, y, label='$f(x)$')
plt.scatter(minima, f(minima),
            s=70,
            color='red',
            label=f'$min$ $f(x)$, {iter_min} iterations')
plt.scatter(maxima, f(maxima),
            s=70,
            color='green',
            label=f'$max$ $f(x)$, {iter_max} iterations')
plt.scatter(right_maxima, f(right_maxima),
            s=70,
            color='green',
            label=f'$max$ $f(x)$, {right_iter_max} iterations')
plt.tight_layout()
```

```
plt.legend()
plt.grid()
plt.savefig('plots/newton.png', dpi=300)
```

2.4 Графики результирующих решений

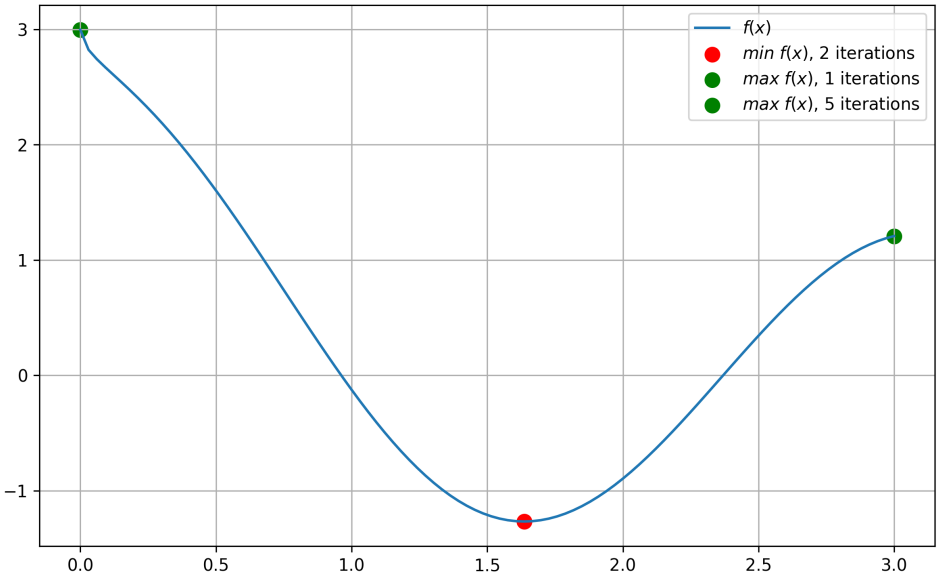

Рис. 1: Найденный минимум и число итераций метода Ньютона

3 Экстремумы функции методами деления (9.2.6)

3.1 Формулировка задачи

Указанным в индивидуальном варианте методом найти минимумы и максимумы функции $f(x)$ на отрезке $[x_1, x_2]$ с точностью $\varepsilon = 10^{-6}$. Предусмотреть подсчет числа итераций, потребовавшихся для достижения заданной точности.

3.2 Вариант

Метод - Фибоначчи.

$$f(x) = (t^2 - 3)/(t^2 + 2)$$

$$[x_1, x_2] = [-1, 4]$$

3.3 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt

from optimize import fibonacci
```

```python
def f(t):
    return (t ** 2 - 3) / (t ** 2 + 2)


interval = (-1, 4)

minima, iter_min = fibonacci(func=f, interval=interval, eps=1e-6)
maxima, iter_max = fibonacci(func=f, interval=interval, eps=1e-6, minimize=False)
t = np.linspace(start=interval[0], stop=interval[1], num=100)
y = f(t)

fig, ax = plt.subplots(figsize=(8, 5))
plt.plot(t, y, label='$f(x)$')
plt.scatter(minima, f(minima),
            s=70, color='red', label=f'$min$ $f(x)$, {iter_min} iterations'
            )
plt.scatter(maxima, f(maxima),
            s=70, color='green', label=f'$max$ $f(x)$, {iter_max} iterations'
            )
plt.tight_layout()
plt.legend()
plt.grid()
plt.savefig('plots/search_optimize.png', dpi=300)
```

3.4 Графики результирующих решений

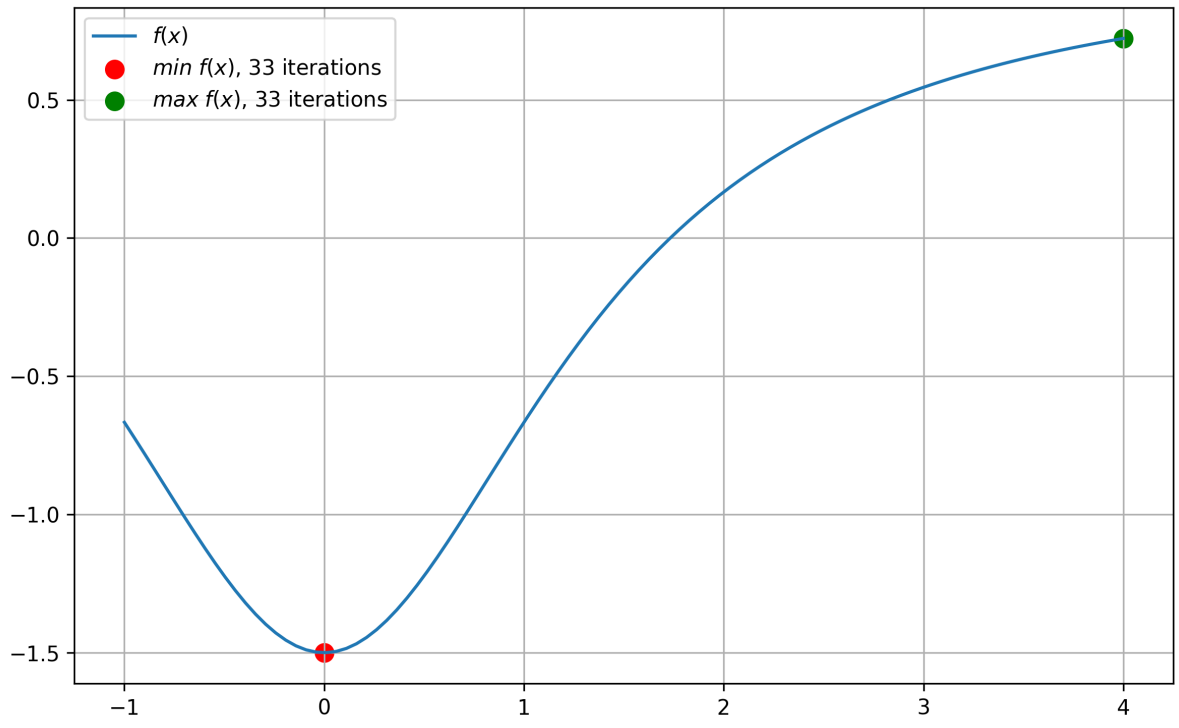

Рис. 2: Найденный минимум и число итераций метода Ньютона

4 Метод Ньютона для функции двух переменных (9.5.16)

4.1 Формулировка задачи

Найти минимум функции 2-х переменных $f(x, y)$ с точностью с указанной точностью $\varepsilon = 10^{-6}$ на прямоугольнике $[x_1, x_2] \times [y_1, y_2]$. Порядок решения:

1. Задать указанную в варианте функцию $f(x, y)$.
2. Построить графики функции и поверхностей уровня $f(x, y)$.
3. По графикам найти точки начального приближения к точкам экстремума.
4. Найти экстремумы функции с заданной точностью.

4.2 Вариант

$$f(x, y) = x^2 + 2y^2 - 4sin(x) - sin(y)$$

$$[x_1, x_2] = [-2, 4]$$

$$[y_1, y_2] = [-2, 4]$$

4.3 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt

from optimize import newton_optimize_vec


def f(x):
    return x[0] ** 2 + 2 * x[1] ** 2 - 4 * np.sin(x[0]) - np.sin(x[1])


def ff(x, y):
    return x ** 2 + 2 * y ** 2 - 4 * np.sin(x) - np.sin(y)


bbox = (np.array([-2, -2]), np.array([4, 4]))

nx, ny = (50, 50)
x = np.linspace(bbox[0][0], bbox[1][0], nx)
y = np.linspace(bbox[0][1], bbox[1][1], ny)
xv, yv = np.meshgrid(x, y)
zv = ff(xv, yv)

plt.subplots(figsize=(7, 6))
cset = plt.contourf(x, y, zv)
plt.axis('scaled')
```

```
plt.colorbar(cset)
plt.savefig('plots/newton2d_levels.png', dpi=300)

print('Enter starting points in format < x y >:')
start = np.array(list(map(float, input().split())))
minima, iter_min = newton_optimize_vec(func=f, bbox=bbox, start=start,
                                       eps=1e-6, minimize=True)
maxima, iter_max = newton_optimize_vec(func=f, bbox=bbox, start=start,
                                       eps=1e-6, minimize=False)

fig, ax = plt.subplots(figsize=(7, 6))
cset = plt.contour(x, y, zv)
plt.axis('scaled')
plt.colorbar(cset)
plt.scatter(minima[0], minima[1],
            s=70,
            color='red',
            label=f'$min$ $f(x)$, {iter_min} iterations')
plt.scatter(maxima[0], maxima[1],
            s=70,
            color='green',
            label=f'$max$ $f(x)$, {iter_max} iterations')
plt.tight_layout()
plt.legend()
plt.savefig('plots/newton2d.png', dpi=300)
```

4.4 Вывод программы

Вводится значение для начальной точки, подобранное по первому графику функции.

```
Enter starting points in format < x y >:
0 1
```

4.5 Графики результирующих решений

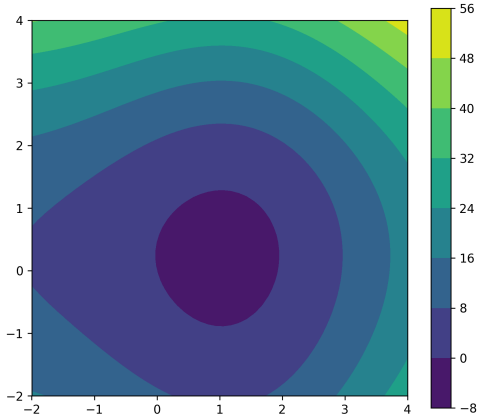

Рис. 3: Значения функции на сетке значений в виде уровней

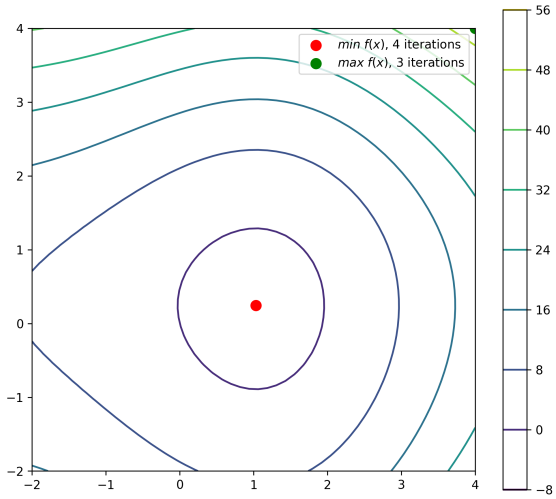

Рис. 4: Найденные экстремумы методом Ньютона

5 Метод первого порядка для функции двух переменных (9.6.16)

5.1 Формулировка задачи

Указанным в индивидуальном варианте методом найти минимум квадратичной функции с точностью $\varepsilon = 10^{-6}$.

$$f(x, y) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y$$

Для решения задачи одномерной минимизации использовать метод Ньютона. Построить график функции f. Предусмотреть подсчет числа итераций, потребовавшихся для достижения заданной точности.

5.2 Вариант

Метод - сопряженных градиентов.

$$a_{11} = 0.5; 2a_{12} = -0.5; a_{22} = 2.5; 2a_{13} = -2.5; 2a_{23} = -3.5$$

5.3 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt

from optimize import conjucate_grad_minimize as conjgrad
from optimize import conjucate_grad_quadratic_minimize as conjgrad_quadr


def get_quad(A, b):

    def f_arr(x):
        return x @ A @ x + b @ x
```

```
    def f_var(x, y):
        return (A[0][0] * x ** 2
                + 2 * A[0][1] * x * y
                + A[1][1] * y ** 2
                + b[0] * x
                + b[1] * y)

    return f_arr, f_var


A = np.array([
    [0.5, -0.25],
    [-0.25, 2.5]
])
b = np.array([-2.5, -3.5])
start = np.array([0, 0])
f, ff = get_quad(A, b)

nx, ny = (50, 50)
x = np.linspace(-5, 5, nx)
y = np.linspace(-5, 5, ny)
xv, yv = np.meshgrid(x, y)
zv = ff(xv, yv)

minima, iter_min = conjgrad(func=f, start=start, eps=1e-6)
minima_q, iter_min_q = conjgrad_quadr(func=f, func_matrix=A, start=start, eps=1e-6)
print(f'min(f(x)) = {minima}, {iter_min} iterations with general formula')
print(f'min(f(x)) = {minima_q}, {iter_min_q} iterations with quadratic formula')

plt.subplots(figsize=(8, 6))
cset = plt.contourf(x, y, zv, cmap='gray')
plt.scatter(minima[0], minima[1],
            s=70,
            color='red',
            label=f'$min$ $f(x)$, {iter_min} iterations')
plt.axis('scaled')
plt.colorbar(cset)
plt.tight_layout()
plt.legend()
plt.savefig('plots/conjucate_grad.png', dpi=300)
```

5.4 Вывод программы

```
min(f(x)) = [3. 1.], 2 iterations with general formula
min(f(x)) = [3. 1.], 2 iterations with quadratic formula
```

5.5 Графики результирующих решений

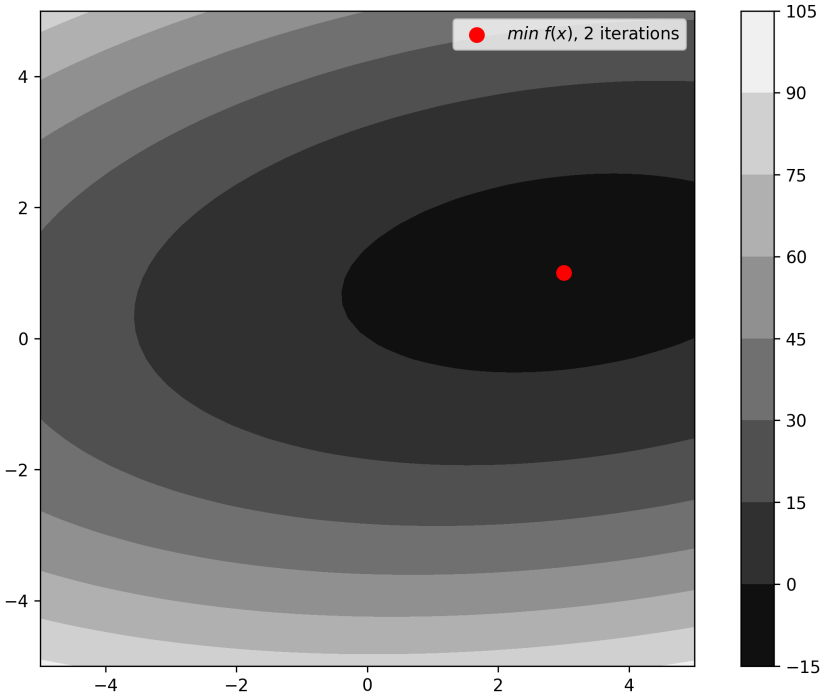

Рис. 5: Найденные экстремумы методом Ньютона