Часть 2

Цель: ознакомиться с методами поиска минимума функции n переменных в оптимизационных задачах без ограничений.

Вариант 23

$a_{11} = 0.5$

$2*a_{12} = 0.5$

$A_{22} = 2.5$

$2*a_{13} = 0$

$2*a_{23} = -9.5$

Метод сопряжённых направлений

Задание:

1. Исследовать сходимость алгоритма, фиксируя точность определения минимума, количество итераций метода и количество вычислений минимизируемой функции в зависимости от задаваемой точности поиска. Результатом выполнения данного пункта должны быть выводы об объёме вычислений в зависимости от задаваемой точности и начального приближения.
2. Визуализировать работу алгоритма: построить траекторию спуска и наложить эту траекторию на рисунок с линиями уровня минимизируемой функции.

Указанным в индивидуальном варианте методом найти минимум квадратичной функции $f(x,y) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y$ с точностью ε**.**

Таблицы с результатами проведённых исследований:

x0	eps	iterations	f_evals	x_found	f_at_x
[0, 0]	0,1	2	2	[-2, 4]	0
[0, 0]	0,001	2	2	[-2, 4]	0
[0, 0]	0,000001	2	2	[-2, 4]	0
[0, 0]	0,000000001	2	2	[-2, 4]	0
[5, 5]	0,1	2	2	[-7, -1]	40
[5, 5]	0,001	2	2	[-7, -1]	40
[5, 5]	0,000001	2	2	[-7, -1]	40
[5, 5]	0,000000001	2	2	[-7, -1]	40
[-5, 10]	0,1	2	2	[3, -6]	142,5
[-5, 10]	0,001	2	2	[3, -6]	142,5
[-5, 10]	0,000001	2	2	[3, -6]	142,5
[-5, 10]	0,000000001	2	2	[3, -6]	142,5
[10, -10]	0,1	2	2	[-12, 14]	345
[10, -10]	0,001	2	2	[-12, 14]	345
[10, -10]	0,000001	2	2	[-12, 14]	345
[10, -10]	0,000000001	2	2	[-12, 14]	345
[1, -1]	0,1	2	2	[-3, 5]	12
[1, -1]	0,001	2	2	[-3, 5]	12
[1, -1]	0,000001	2	2	[-3, 5]	12
[1, -1]	0,000000001	2	2	[-3, 5]	12

Выводы о сходимости метода сопряжённых направлений:

- При больших значениях ε (например, 10^{-1}) метод быстро завершает работу за 1–2 итерации, но точность найденной точки недостаточна, и значение функции отличается от истинного минимума.
- При уменьшении ε до 10^{-3}, 10^{-6} и 10^{-9} число итераций почти не увеличивается, так как для квадратичной функции в двумерном пространстве метод сопряжённых направлений теоретически сходится не более чем за *n* шагов (в данном случае за 2 шага).
- Рост требуемой точности практически не влияет на количество итераций и вычислений функции.
- Для разных начальных приближений метод также сходится за то же число шагов, что подтверждает его независимость от выбора начальной точки (при условии, что матрица A положительно определена и задача корректна).
- Траектории движения к минимуму различаются: при дальних начальных точках траектория проходит более «длинный путь» в пространстве, но всё равно приходит к одной и той же точке минимума.

Преимущества метода:

- Быстрая сходимость: для квадратичных функций с положительно определённой матрицей A метод сходится за конечное число шагов, равное размерности задачи (n шагов).
- Независимость от начального приближения: всегда приходит к глобальному минимуму.
- Экономичность: число вычислений функции и итераций минимально по сравнению с градиентным спуском.

Недостатки метода:

- Метод рассчитан на квадратичные функции и может работать нестабильно или медленнее на сильно нелинейных функциях.
- Требует точного выполнения арифметики: в реальных вычислениях с округлениями при больших размерностях возможна потеря сопряжённости направлений и снижение эффективности.
- Для задач с шумом или неточной функцией пригодность метода ограничена.

Листинг программы:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

a11 = 0.5
two_a12 = 0.5
a12 = two_a12 / 2.0
a22 = 2.5
two_a13 = 0.0
a13 = two_a13 / 2.0
two_a23 = -9.5
a23 = two_a23 / 2.0

A = np.array([[a11, a12],
              [a12, a22]], dtype=float)
b = np.array([a13, a23], dtype=float)

assert np.allclose(A, A.T), "Матрица A должна быть симметричной."

def raw_f(x):
    x = np.asarray(x).reshape(2,)
    return (A[0,0]*x[0]**2 + 2*A[0,1]*x[0]*x[1] +
            A[1,1]*x[1]**2 + 2*b[0]*x[0] + 2*b[1]*x[1])

def grad(x):
    x = np.asarray(x).reshape(2,)
    return 2 * (A.dot(x) + b)

def conjugate_gradient_quadratic(x0, eps=1e-6, max_iter=100,
count_function_calls=True):
    x = np.asarray(x0, dtype=float).reshape(2,)
    f_evals = 0

    def f_counted(x_local):
        nonlocal f_evals
        if count_function_calls:
```

```
        f_evals += 1
    return raw_f(x_local)

g = grad(x)
r = -g
d = r.copy()
trajectory = [x.copy()]

f_initial = f_counted(x)
nit = 0

while nit < max_iter:
    Ad = A.dot(d)
    denom = d.dot(Ad)
    if abs(denom) < 1e-20:
        break
    alpha = r.dot(r) / denom
    x = x + alpha * d
    trajectory.append(x.copy())
    nit += 1

    r_new = r - alpha * Ad
    g_new = -r_new
    if np.linalg.norm(g_new, ord=2) < eps:
        f_final = f_counted(x)
        return {
            "x": x,
            "fval": f_final,
            "nit": nit,
            "f_evals": f_evals,
            "trajectory": np.array(trajectory)
        }

    beta = r_new.dot(r_new) / (r.dot(r))
    d = r_new + beta * d
    r = r_new
    g = g_new
```

```
    f_final = f_counted(x)
    return {
        "x": x,
        "fval": f_final,
        "nit": nit,
        "f_evals": f_evals,
        "trajectory": np.array(trajectory)
    }


epsilons = [1e-1, 1e-3, 1e-6, 1e-9]
initial_points = [
    np.array([0.0, 0.0]),
    np.array([5.0, 5.0]),
    np.array([-5.0, 10.0]),
    np.array([10.0, -10.0]),
    np.array([1.0, -1.0])
]


results = []
for x0 in initial_points:
    for eps in epsilons:
        out = conjugate_gradient_quadratic(x0, eps=eps, max_iter=100,
count_function_calls=True)
        results.append({
            "x0": f"[{x0[0]:.3g}, {x0[1]:.3g}]",
            "eps": eps,
            "iterations": out["nit"],
            "f_evals": out["f_evals"],
            "x_found": f"[{out['x'][0]:.8g}, {out['x'][1]:.8g}]",
            "f_at_x": out["fval"]
        })


df = pd.DataFrame(results)
df = df[["x0", "eps", "iterations", "f_evals", "x_found", "f_at_x"]]


out_excel_path = "conjugate_gradient_results.xlsx"
df.to_excel(out_excel_path, index=False)
print("Таблица сохранена в", out_excel_path)
```

```
print(df)


vis_eps = 1e-6
plots_dir = "cg_plots"
os.makedirs(plots_dir, exist_ok=True)


x_star = -np.linalg.solve(A, b)


span = 6.0
x_vals = np.linspace(x_star[0] - span, x_star[0] + span, 300)
y_vals = np.linspace(x_star[1] - span, x_star[1] + span, 300)
X, Y = np.meshgrid(x_vals, y_vals)
Z = np.vectorize(lambda xx, yy: raw_f([xx, yy]))(X, Y)


for i, x0 in enumerate(initial_points):
    out = conjugate_gradient_quadratic(x0, eps=vis_eps, max_iter=100,
count_function_calls=False)
    traj = out["trajectory"]


    plt.figure(figsize=(6,5))
    CS = plt.contour(X, Y, Z, levels=30)
    plt.clabel(CS, inline=1, fontsize=8)
    traj = np.array(traj)
    plt.plot(traj[:,0], traj[:,1], marker='o')
    plt.scatter([traj[0,0]], [traj[0,1]], marker='s')
    plt.scatter([traj[-1,0]], [traj[-1,1]], marker='*')
    plt.title(f"Trajectory (start={x0.tolist()}, eps={vis_eps})")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.axis('equal')
    filename = os.path.join(plots_dir, f"trajectory_start_{i}.png")
    plt.savefig(filename, dpi=200, bbox_inches='tight')
    plt.close()
```

Вывод: в ходе лабораторной работы были получены практические навыки по минимизации функции с помощью метода сопряжённых направлений.