**Часть 2**

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

**Вариант 23**

a11 = 0.5

2\*a12 = 0.5

A22 = 2.5

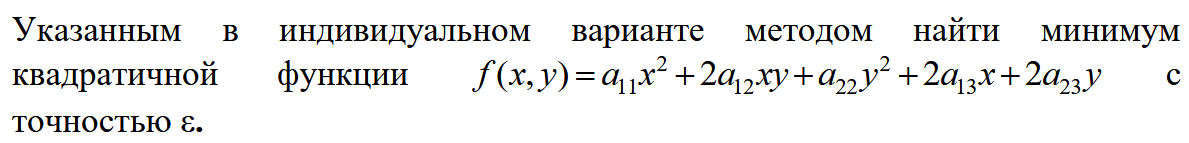
2\*a13 = 0

2\*a23 = -9.5

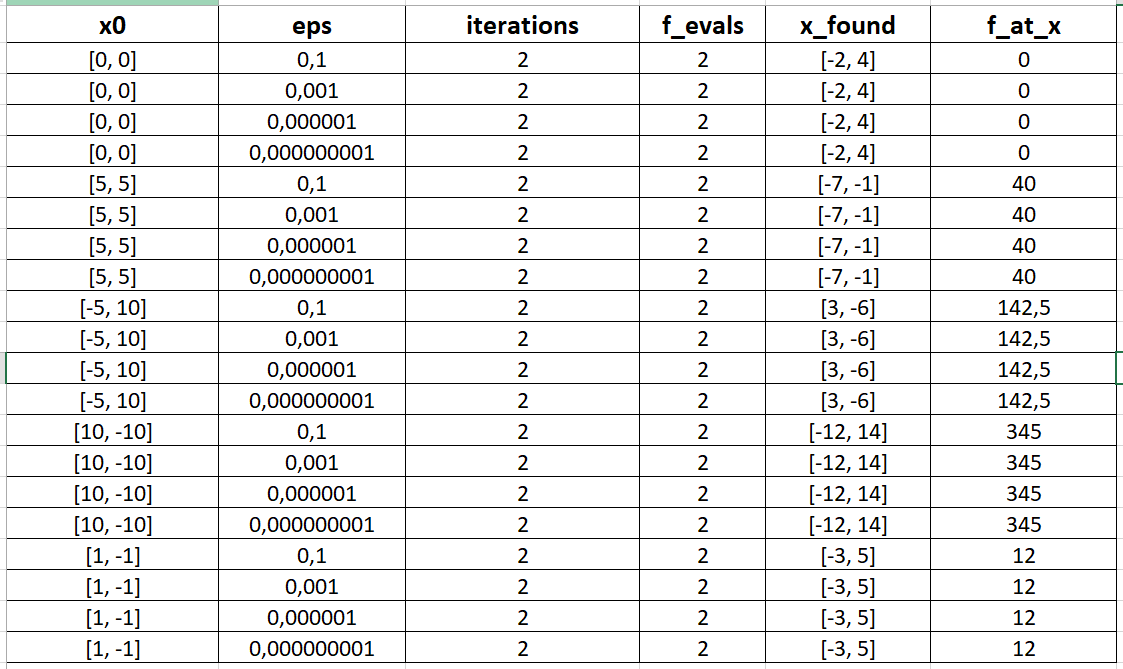
Метод сопряжённых направлений

**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.



**Таблицы с результатами проведённых исследований:**



**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.