МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | **Факультет прикладной математики и информатики**

Кафедра теоретической и прикладной информатики

Практическое задание № 4

по дисциплине «Компьютерные технологии моделирования и анализа данных»

# МЕТОДЫ ПОСТРОЕНИЯ ИЗОБРАЖЕНИЙ КОНЕЧНОЭЛЕМЕНТНЫХ РЕШЕНИЙ


Вариант 1 ПММ-52 КУСАКИН АЛЕКСАНДР

ПММ-52 ЦИРКОВА АЛИНА

ПММ-53 БОРИСОВ ДМИТРИЙ

Преподаватели КОШКИНА ЮЛИЯ ИГОРЕВНА


Новосибирск, 2025

**Цель работы**

Изучить методы построения изображений конечноэлементного решения при решении различных задач. Реализовать методы построения изображения решения для элементов высоких порядков.

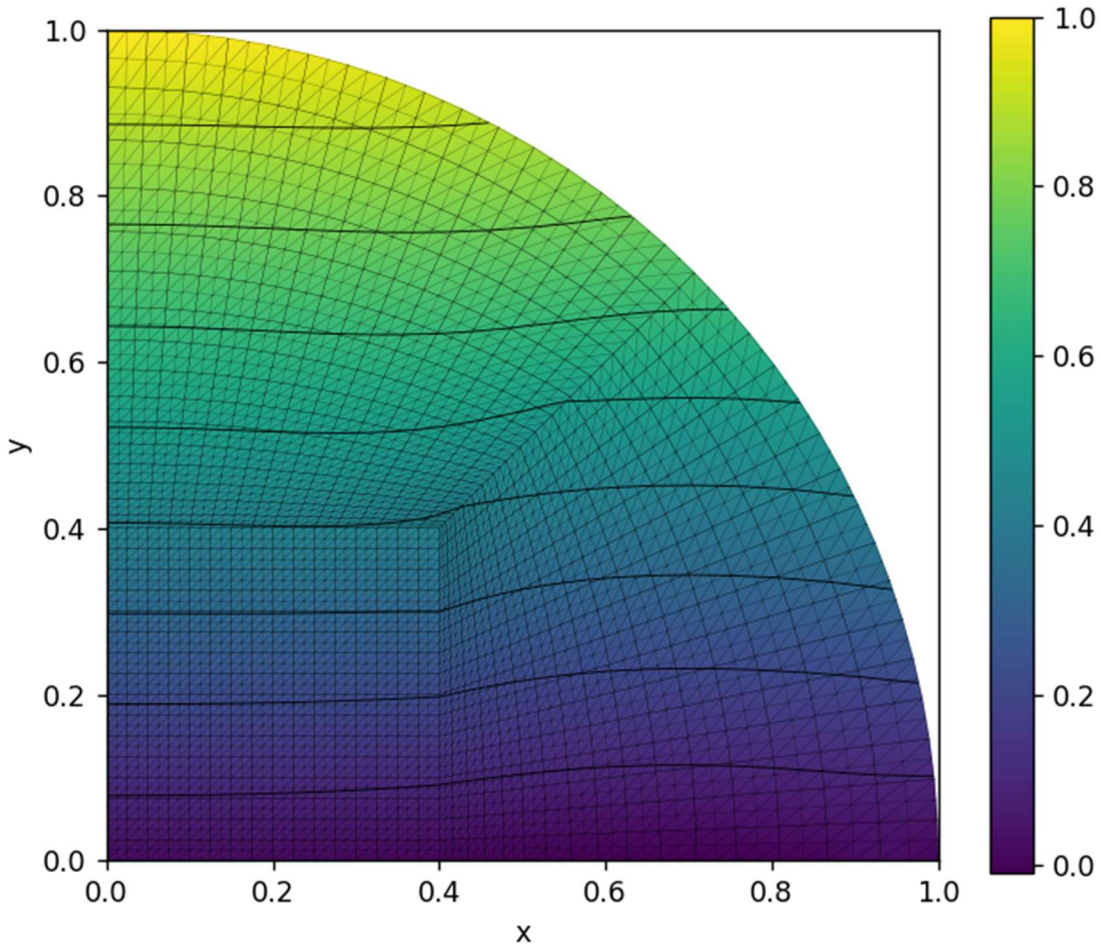
**Задание**

Сгенерировать двумерную конечноэлементную сетку из плоских четырехугольников. На основании программы рисования поля на треугольной сетке написать программу для сетки из четырехугольников, разбивая каждый четырехугольник на более мелкие четырехугольники, которые затем разбить на треугольники.

**Тестирование**

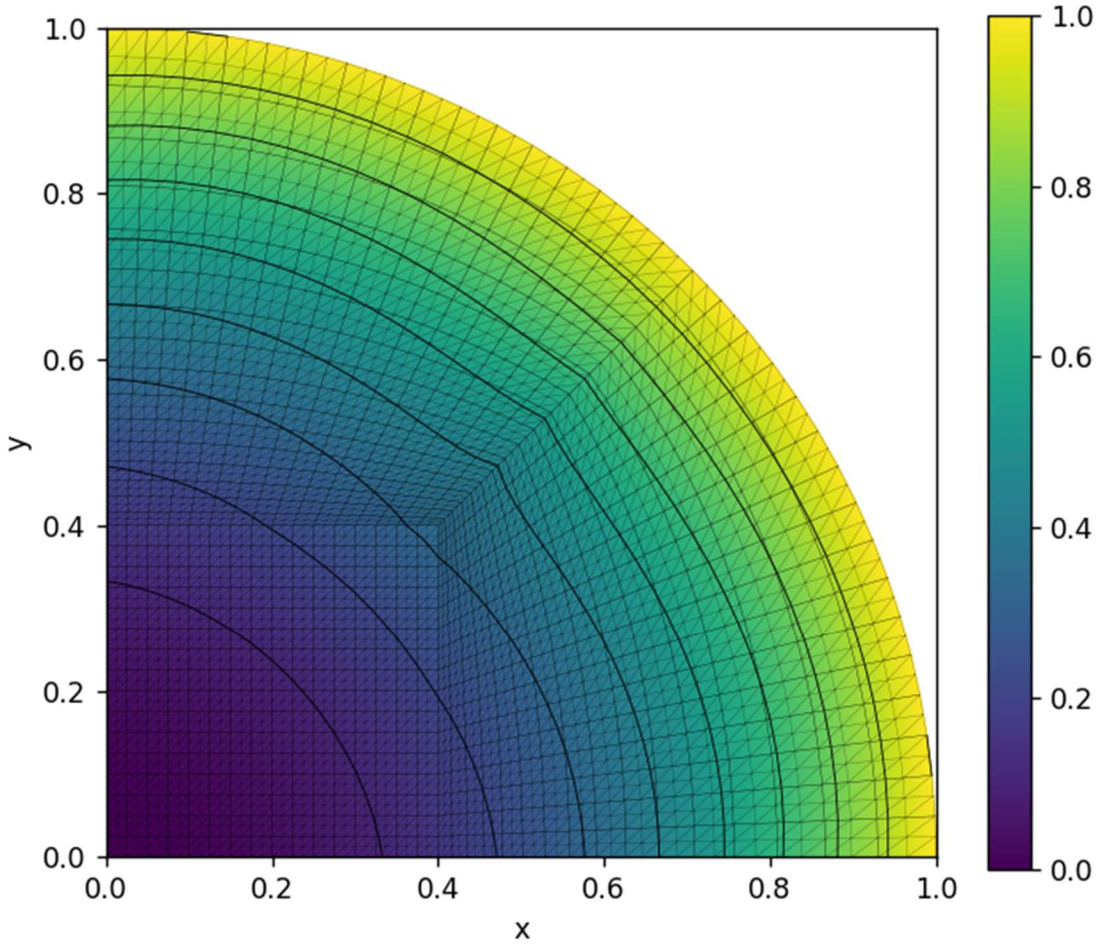
1. *Линейная функция*

Уравнение: $-\Delta u + u = y$
Аналитическое решение: $u = y$
Краевые условия: 1го рода на дуге, 2го рода на осях
Максимальная погрешность: 3.728e-02



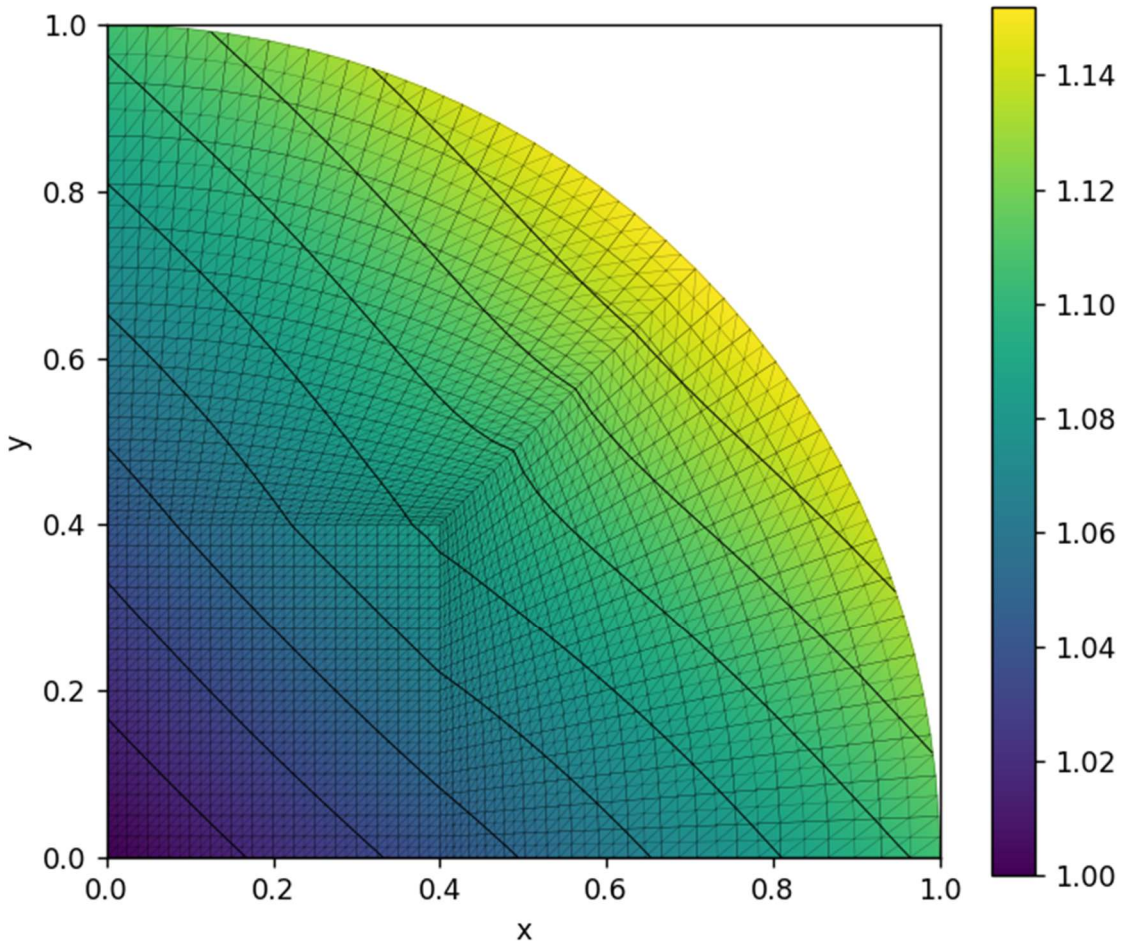
2. *Полином*

Уравнение: $-\Delta u + u = -4 + x^2 + y^2 - xy$
Аналитическое решение: $u = x^2 + y^2 - xy$
Краевые условия: 1го рода на всех границах
Погрешность: 4.271e-02



*3. Экспоненциальная функция*

Уравнение: $-\Delta u + u = -0.02e^{0.1(x+y)} + e^{0.1(x+y)}$
Аналитическое решение: $u = e^{0.1(x+y)}$
Краевые условия: 1го рода на всех границах
Погрешность: 4.841e-03



**Код**

```
import sys
from typing import List, Tuple, Dict
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.tri as mtri

def read_mesh(filename: str) -> Tuple[List[Tuple[float, float]], List[List[int]]]:
    with open(filename, "r", encoding="utf-8") as f:
        lines = [line.strip() for line in f if line.strip()]

    i = 0
    if lines[i] != "NODES":
        raise ValueError("Ожидалась секция NODES")
    i += 1

    n_nodes = int(lines[i])
    i += 1

    nodes: List[Tuple[float, float]] = []
    for _ in range(n_nodes):
        parts = lines[i].split()
        x = float(parts[1])
        y = float(parts[2])
        nodes.append((x, y))
        i += 1

    if lines[i] != "ELEMENTS":
        raise ValueError("Ожидалась секция ELEMENTS")
    i += 1

    n_elems = int(lines[i])
    i += 1

    quads: List[List[int]] = []
    for _ in range(n_elems):
        parts = lines[i].split()
        n1 = int(parts[1])
        n2 = int(parts[2])
        n3 = int(parts[3])
        n4 = int(parts[4])
        quads.append([n1, n2, n3, n4])
        i += 1

    if len(nodes) != n_nodes:
        raise ValueError("Несовпадение числа узлов при чтении")
    if len(quads) != n_elems:
        raise ValueError("Несовпадение числа элементов при чтении")
```

```
    return nodes, quads

def read_solution_table(filename: str) -> Dict[int, float]:
    sol: Dict[int, float] = {}

    with open(filename, "r", encoding="utf-8") as f:
        header = f.readline()
        if not header:
            raise ValueError("Пустой файл решения")

        for line in f:
            line = line.strip()
            if not line:
                continue
            parts = line.split()
            if len(parts) < 5:
                continue
            i_node = int(parts[0])
            u_n = float(parts[3])
            sol[i_node] = u_n

    return sol

def quads_to_tris(elements: List[List[int]]) -> List[Tuple[int, int, int]]:
    tris: List[Tuple[int, int, int]] = []
    for quad in elements:
        n1, n2, n3, n4 = quad
        tris.append((n1, n2, n3))
        tris.append((n1, n3, n4))
    return tris

def refine_quads_with_u(
    nodes: List[Tuple[float, float]],
    quads: List[List[int]],
    u_coarse: np.ndarray,
    nx: int = 2,
    ny: int = 2,
) -> Tuple[List[Tuple[float, float]], np.ndarray, List[List[int]]]:

    new_nodes: List[Tuple[float, float]] = []
    new_u_values: List[float] = []
    new_quads: List[List[int]] = []

    for quad in quads:
        n1, n2, n3, n4 = quad
        x1, y1 = nodes[n1]
        x2, y2 = nodes[n2]
        x3, y3 = nodes[n3]
```

```
        x4, y4 = nodes[n4]

        u1 = u_coarse[n1]
        u2 = u_coarse[n2]
        u3 = u_coarse[n3]
        u4 = u_coarse[n4]

        local_idx = [[-1] * (nx + 1) for _ in range(ny + 1)]

        for j in range(ny + 1):
            t = j / ny
            for i in range(nx + 1):
                s = i / nx

                N1 = (1 - s) * (1 - t)
                N2 = s * (1 - t)
                N3 = s * t
                N4 = (1 - s) * t

                x = N1 * x1 + N2 * x2 + N3 * x3 + N4 * x4
                y = N1 * y1 + N2 * y2 + N3 * y3 + N4 * y4
                u_new = N1 * u1 + N2 * u2 + N3 * u3 + N4 * u4

                gid = len(new_nodes)
                new_nodes.append((x, y))
                new_u_values.append(u_new)
                local_idx[j][i] = gid

        for j in range(ny):
            for i in range(nx):
                k1 = local_idx[j][i]
                k2 = local_idx[j][i + 1]
                k3 = local_idx[j + 1][i + 1]
                k4 = local_idx[j + 1][i]
                new_quads.append([k1, k2, k3, k4])

    new_u = np.array(new_u_values, dtype=float)
    return new_nodes, new_u, new_quads

def build_u_on_coarse(nodes: List[Tuple[float, float]],
                      sol: Dict[int, float]) -> np.ndarray:
    n = len(nodes)
    u = np.zeros(n, dtype=float)
    for i in range(n):
        u[i] = sol[i]
    return u

def main():
    mesh_file = "radial_mesh.txt"
```

```
    solution_file = "solution_exp.txt"

    if len(sys.argv) >= 2:
        mesh_file = sys.argv[1]
    if len(sys.argv) >= 3:
        solution_file = sys.argv[2]

    nx_ref, ny_ref = 2, 2

    nodes, quads = read_mesh(mesh_file)
    sol = read_solution_table(solution_file)

    u_coarse = build_u_on_coarse(nodes, sol)

    ref_nodes, u_refined, ref_quads = refine_quads_with_u(
        nodes, quads, u_coarse, nx=nx_ref, ny=ny_ref
    )

    x = np.array([p[0] for p in ref_nodes])
    y = np.array([p[1] for p in ref_nodes])

    tris = quads_to_tris(ref_quads)
    triangles = np.array(tris, dtype=int)
    triang = mtri.Triangulation(x, y, triangles)

    fig, ax = plt.subplots(figsize=(6, 5))

    tpc = ax.tripcolor(triang, u_refined, shading="gouraud", cmap="viridis")
    fig.colorbar(tpc, ax=ax)

    n_levels = 10
    levels = np.linspace(u_refined.min(), u_refined.max(), n_levels)
    ax.tricontour(triang, u_refined, levels=levels, colors="k", linewidths=0.5)

    ax.triplot(triang, color="black", linewidth=0.2, alpha=0.5)

    ax.set_aspect("equal")
    ax.set_xlabel("x")
    ax.set_ylabel("y")

    plt.tight_layout()
    plt.show()

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