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 № 3
по дисциплине «Компьютерные технологии моделирования и анализа данных»

ПОСТРОЕНИЕ ПОРТРЕТА И СБОРКА КОНЕЧНОЭЛЕМЕНТНОЙ МАТРИЦЫ

Вариант 1 ПММ-52 КУСАКИН АЛЕКСАНДР
ПММ-52 ЦИРКОВА АЛИНА
ПММ-53 БОРИСОВ ДМИТРИЙ

Преподаватели КОШКИНА ЮЛИЯ ИГОРЕВНА

Новосибирск, 2025

Цель работы

Реализовать алгоритм построения портрета и сборки конечноэлементной матрицы для разреженного строчного формата хранения. Протестировать написанную программу.

Задание

Написать подпрограмму построения портрета матрицы, возникающей при решении эллиптической краевой задачи методом конечных элементов и использованием базисных функций.

Реализовать алгоритм занесения локальных матриц и локальных векторов конечных элементов в глобальную матрицу и глобальный вектор конечноэлементной СЛАУ.

Проверить правильность формирования портрета и сборки конечноэлементной матрицы на тестовых задачах.

Теоретическая часть

Отобразим единичный квадрат $\Omega^E = \{(\xi, \eta) \mid 0 \le \xi \le 1, 0 \le \eta \le 1\}$ в четырехугольник Ω_kс вершинами $(\hat{x}_i, \hat{y}_i)$ с помощью следующих соотношений:

$$x = (1-\xi)(1-\eta)\hat{x}_1 + \xi(1-\eta)\hat{x}_2 + (1-\xi)\eta\hat{x}_3 + \xi\eta\hat{x}_4$$

$$y = (1-\xi)(1-\eta)\hat{y}_1 + \xi(1-\eta)\hat{y}_2 + (1-\xi)\eta\hat{y}_3 + \xi\eta\hat{y}_4$$

Будем использовать следующие обозначения:

$$\alpha_0 = (\hat{x}_2 - \hat{x}_1)(\hat{y}_3 - \hat{y}_1) - (\hat{y}_2 - \hat{y}_1)(\hat{x}_3 - \hat{x}_1)$$

$$\alpha_1 = (\hat{x}_2 - \hat{x}_1)(\hat{y}_4 - \hat{y}_3) - (\hat{y}_2 - \hat{y}_1)(\hat{x}_4 - \hat{x}_3)$$

$$\alpha_3 = (\hat{y}_3 - \hat{y}_1)(\hat{x}_4 - \hat{x}_2) - (\hat{x}_3 - \hat{x}_1)(\hat{y}_4 - \hat{y}_2)$$

$$\beta_1 = \hat{x}_3 - \hat{x}_1, \quad \beta_2 = \hat{x}_2 - \hat{x}_1, \quad \beta_3 = \hat{y}_3 - \hat{y}_1, \quad \beta_4 = \hat{y}_2 - \hat{y}_1,$$
$$\beta_5 = \hat{x}_1 - \hat{x}_2 - \hat{x}_3 + \hat{x}_4, \quad \beta_6 = \hat{y}_1 - \hat{y}_2 - \hat{y}_3 + \hat{y}_4.$$

С учетом этих обозначений, матрицы массы и жесткости примут вид:

$$\widehat{M}_{i,j} = \iint\limits_{\Omega_k} \hat{\psi}_i(x,y)\hat{\psi}_j(x,y)dxdy = \iint\limits_{\Omega^E} \hat{\psi}_i(\xi,\eta)\hat{\psi}_j(\xi,\eta)|J|d\xi d\eta$$

$$\widehat{G}_{i,j} = \iint\limits_{\Omega_k} \nabla\hat{\psi}_i(x,y)\cdot\nabla\hat{\psi}_j(x,y)dxdy = \text{sign}(\alpha_0)\iint\limits_{\Omega^E} \frac{1}{|J|} *$$
$$*\left(\left[\frac{\partial\hat{\psi}_i(\xi,\eta)}{\partial\xi}(\beta_6\xi+\beta_3) - \frac{\partial\hat{\psi}_i(\xi,\eta)}{\partial\eta}(\beta_6\eta+\beta_4)\right]\left[\frac{\partial\hat{\psi}_j(\xi,\eta)}{\partial\xi}(\beta_6\xi+\beta_3) - \frac{\partial\hat{\psi}_j(\xi,\eta)}{\partial\eta}(\beta_6\eta+\beta_4)\right] + \right.,$$
$$\left. + \left[\frac{\partial\hat{\psi}_i(\xi,\eta)}{\partial\eta}(\beta_5\eta+\beta_2) - \frac{\partial\hat{\psi}_i(\xi,\eta)}{\partial\xi}(\beta_5\xi+\beta_1)\right]\left[\frac{\partial\hat{\psi}_j(\xi,\eta)}{\partial\eta}(\beta_5\eta+\beta_2) - \frac{\partial\hat{\psi}_j(\xi,\eta)}{\partial\xi}(\beta_5\xi+\beta_1)\right]\right)d\xi d\eta$$

где $i, j = 1..4$, $|J| = \alpha_0 + \alpha_1\xi + \alpha_2\eta$.

Для перевода четырехугольника Ω_k в единичный квадрат Ω^E воспользуемся следующими соотношениями:

$$\xi = \frac{\beta_3(x-\hat{x}_1)-\beta_1(y-\hat{y}_1)}{\beta_2\beta_3-\beta_1\beta_4}, \quad \eta = \frac{\beta_2(y-\hat{y}_1)-\beta_4(x-\hat{x}_1)}{\beta_2\beta_3-\beta_1\beta_4} \text{ при } \alpha_1 = \alpha_2 = 0;$$

$$\xi = \frac{\alpha_2(x-\hat{x}_1)+\beta_1 w(x,y)}{\alpha_2\beta_2-\beta_5 w(x,y)}, \quad \eta = -\frac{w(x,y)}{\alpha_2} \quad \text{при } \alpha_1 = 0 \text{ и } \alpha_2 \neq 0;$$

$$\xi = \frac{w(x,y)}{\alpha_1}, \quad \eta = \frac{\alpha_1(y-\hat{y}_1)-\beta_4 w(x,y)}{\alpha_1\beta_3-\beta_6 w(x,y)} \quad \text{при } \alpha_1 \neq 0 \text{ и } \alpha_2 = 0;$$

$$\beta_5\alpha_2\eta^2 + (\alpha_2\beta_2 + \alpha_1\beta_1 + \beta_5 w(x,y))\eta + \alpha_1(\hat{x}_1 - x) + \beta w(x,y) = 0,$$

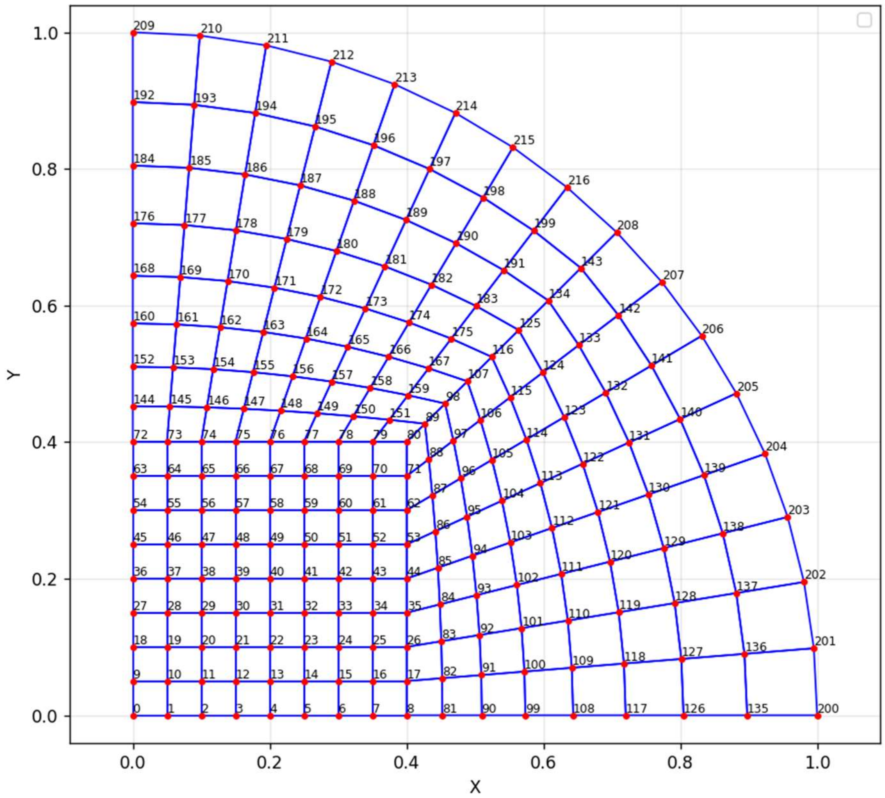
$$\xi = \frac{\alpha_2}{\alpha_1}\eta + \frac{w(x,y)}{\alpha_1} \text{ при } \alpha_1 \neq 0 \text{ и } \alpha_2 \neq 0;$$

где $w(x,y) = \beta_6(x-\hat{x}_1) - \beta_5(y-\hat{y}_1)$.

Тестирование

Сетка для тестовых задач:

Коэффициент разрядки = 1.1
Количество узлов = 9



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

Вычисление погрешности и порядка аппроксимации

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

$\sqrt{\frac{\parallel u - u^h \parallel}{\parallel u \parallel}}$	$\sqrt{\frac{\parallel u - u^{h/2} \parallel}{\parallel u \parallel}}$	$\sqrt{\frac{\parallel u - u^{h/4} \parallel}{\parallel u \parallel}}$	$\log_2 \sqrt{\frac{\parallel u - u^h \parallel}{\parallel u - u^{h/2} \parallel}}$	$\log_2 \sqrt{\frac{\parallel u - u^{h/2} \parallel}{\parallel u - u^{h/4} \parallel}}$
4,09E-02	3,70E-02	3,93E-02	1,21	1,13

Код

Построение портрета

def build_portrait(n_nodes: int, elements):
    row_to_cols = defaultdict(set)
 	
    for elem_nodes, _mat in elements:
        k = len(elem_nodes)
        for a in range(k):
            ia = elem_nodes[a]
            for b in range(a + 1, k):
                ib = elem_nodes[b]
                if ia == ib:
                    continue
                i_min = min(ia, ib)
                i_max = max(ia, ib)
                row_to_cols[i_min].add(i_max)
 	
    ig = [0] * (n_nodes + 1)
    jg: List[int] = []
 	
    nnz = 0
    for i in range(n_nodes):
        cols = row_to_cols.get(i, set())
        cols_sorted = sorted(cols)
        nnz += len(cols_sorted)
        ig[i + 1] = nnz
        jg.extend(cols_sorted)
 	
    return ig, jg

Вычисление локальной матрицы

def _element_geometry(elem_nodes: Sequence[int],
                      nodes_coords: Sequence[Tuple[float, float]]):
    pts = [(nodes_coords[nid][0], nodes_coords[nid][1], nid) for nid in elem_nodes]
 	
    xs = [p[0] for p in pts]
    ys = [p[1] for p in pts]
    min_x, max_x = min(xs), max(xs)
    min_y, max_y = min(ys), max(ys)
 	
    def closest(pt_list, x0, y0):
        return min(pt_list, key=lambda p: (abs(p[0] - x0) + abs(p[1] - y0)))
 	
    bl = closest(pts, min_x, min_y)
    br = closest(pts, max_x, min_y)
    tr = closest(pts, max_x, max_y)
    tl = closest(pts, min_x, max_y)

```
    pts_std = [bl, br, tr, tl]
    x_std = [p[0] for p in pts_std]
    y_std = [p[1] for p in pts_std]
    nid_std = [p[2] for p in pts_std]
    return x_std, y_std, nid_std

def compute_local_stiffness_quad(elem_nodes: Sequence[int],
                                 nodes_coords: Sequence[Tuple[float, float]],
                                 lam: float = 1.0) -> List[List[float]]:
    assert len(elem_nodes) == 4

    x_std, y_std, nid_std = _element_geometry(elem_nodes, nodes_coords)

    a = math.sqrt(3.0 / 5.0)
    b = math.sqrt(1.0 / 5.0)

    w_corner = 25.0 / 324.0
    w_edge   = 40.0 / 324.0
    w_inner  = 64.0 / 324.0

    gauss_pts = [
        (-a, -a), ( a, -a), ( a,  a), (-a,  a),
        (-a,  0.0), ( a,  0.0), ( 0.0, -a), ( 0.0,  a),
        (-b, -b), ( b, -b), ( b,  b), (-b,  b),
    ]
    gauss_w = [
        w_corner, w_corner, w_corner, w_corner,
        w_edge,   w_edge,   w_edge,   w_edge,
        w_inner,  w_inner,  w_inner,  w_inner,
    ]

    K_std = [[0.0] * 4 for _ in range(4)]

    for (xi, eta), w in zip(gauss_pts, gauss_w):
        dN1_dxi  = -0.25 * (1 - eta)
        dN1_deta = -0.25 * (1 - xi)
        dN2_dxi  =  0.25 * (1 - eta)
        dN2_deta = -0.25 * (1 + xi)
        dN3_dxi  =  0.25 * (1 + eta)
        dN3_deta =  0.25 * (1 + xi)
        dN4_dxi  = -0.25 * (1 + eta)
        dN4_deta =  0.25 * (1 - xi)

        dN_dxi  = [dN1_dxi,  dN2_dxi,  dN3_dxi,  dN4_dxi]
        dN_deta = [dN1_deta, dN2_deta, dN3_deta, dN4_deta]

        dx_dxi = dx_deta = dy_dxi = dy_deta = 0.0
        for iN in range(4):
            dx_dxi  += dN_dxi[iN]  * x_std[iN]
```

```
        dx_deta += dN_deta[iN] * x_std[iN]
        dy_dxi  += dN_dxi[iN]  * y_std[iN]
        dy_deta += dN_deta[iN] * y_std[iN]

      J11 = dx_dxi
      J12 = dx_deta
      J21 = dy_dxi
      J22 = dy_deta
      detJ = J11 * J22 - J12 * J21

      invJ11 =  J22 / detJ
      invJ12 = -J12 / detJ
      invJ21 = -J21 / detJ
      invJ22 =  J11 / detJ

      dN_dx = [0.0] * 4
      dN_dy = [0.0] * 4
      for iN in range(4):
        dxi  = dN_dxi[iN]
        deta = dN_deta[iN]
        dN_dx[iN] = invJ11 * dxi + invJ12 * deta
        dN_dy[iN] = invJ21 * dxi + invJ22 * deta

      for iN in range(4):
        for jN in range(4):
          grad_dot = dN_dx[iN] * dN_dx[jN] + dN_dy[iN] * dN_dy[jN]
          K_std[iN][jN] += lam * grad_dot * detJ * w
  pos_in_std = {nid: i for i, nid in enumerate(nid_std)}
  K_elem = [[0.0] * 4 for _ in range(4)]
  for i_e, nid_i in enumerate(elem_nodes):
    i_s = pos_in_std[nid_i]
    for j_e, nid_j in enumerate(elem_nodes):
      j_s = pos_in_std[nid_j]
      K_elem[i_e][j_e] = K_std[i_s][j_s]

  return K_elem

def compute_local_mass_quad(elem_nodes: Sequence[int],
                  nodes_coords: Sequence[Tuple[float, float]]) -> List[List[float]]:
  assert len(elem_nodes) == 4

  x_std, y_std, nid_std = _element_geometry(elem_nodes, nodes_coords)

  a = math.sqrt(3.0 / 5.0)
  b = math.sqrt(1.0 / 5.0)

  w_corner = 25.0 / 324.0
  w_edge   = 40.0 / 324.0
  w_inner  = 64.0 / 324.0
```

```
gauss_pts = [
    (-a, -a), ( a, -a), ( a,  a), (-a,  a),
    (-a,  0.0), ( a,  0.0), ( 0.0, -a), ( 0.0,  a),
    (-b, -b), ( b, -b), ( b,  b), (-b,  b),
]
gauss_w = [
    w_corner, w_corner, w_corner, w_corner,
    w_edge,   w_edge,   w_edge,   w_edge,
    w_inner,  w_inner,  w_inner,  w_inner,
]

M_std = [[0.0] * 4 for _ in range(4)]

for (xi, eta), w in zip(gauss_pts, gauss_w):
    N1 = 0.25 * (1 - xi) * (1 - eta)
    N2 = 0.25 * (1 + xi) * (1 - eta)
    N3 = 0.25 * (1 + xi) * (1 + eta)
    N4 = 0.25 * (1 - xi) * (1 + eta)
    N = [N1, N2, N3, N4]

    dN1_dxi  = -0.25 * (1 - eta)
    dN1_deta = -0.25 * (1 - xi)
    dN2_dxi  =  0.25 * (1 - eta)
    dN2_deta = -0.25 * (1 + xi)
    dN3_dxi  =  0.25 * (1 + eta)
    dN3_deta =  0.25 * (1 + xi)
    dN4_dxi  = -0.25 * (1 + eta)
    dN4_deta =  0.25 * (1 - xi)

    dN_dxi  = [dN1_dxi,  dN2_dxi,  dN3_dxi,  dN4_dxi]
    dN_deta = [dN1_deta, dN2_deta, dN3_deta, dN4_deta]

    dx_dxi = dx_deta = dy_dxi = dy_deta = 0.0
    for iN in range(4):
        dx_dxi  += dN_dxi[iN]  * x_std[iN]
        dx_deta += dN_deta[iN] * x_std[iN]
        dy_dxi  += dN_dxi[iN]  * y_std[iN]
        dy_deta += dN_deta[iN] * y_std[iN]

    J11 = dx_dxi
    J12 = dx_deta
    J21 = dy_dxi
    J22 = dy_deta
    detJ = J11 * J22 - J12 * J21

    for iN in range(4):
        for jN in range(4):
            M_std[iN][jN] += N[iN] * N[jN] * detJ * w
```

```
    pos_in_std = {nid: i for i, nid in enumerate(nid_std)}
    M_elem = [[0.0] * 4 for _ in range(4)]
    for i_e, nid_i in enumerate(elem_nodes):
        i_s = pos_in_std[nid_i]
        for j_e, nid_j in enumerate(elem_nodes):
            j_s = pos_in_std[nid_j]
            M_elem[i_e][j_e] = M_std[i_s][j_s]

    return M_elem
```

Занесение локальной матрицы в глоабльную

```
def add_local_matrix(
    di: List[float],
    ggl: List[float],
    ggu: List[float],
    ig: Sequence[int],
    jg: Sequence[int],
    L: Sequence[int],
    A_loc: Sequence[Sequence[float]],
):
    k = len(L)

    for a in range(k):
        i_glob = L[a]
        di[i_glob] += A_loc[a][a]

    for a in range(k):
        i_glob = L[a]
        row_start = ig[i_glob]
        row_end = ig[i_glob + 1]

        for b in range(a + 1, k):
            j_glob = L[b]

            pos = -1
            for p in range(row_start, row_end):
                if jg[p] == j_glob:
                    pos = p
                    break

            if pos == -1:
                print(f"ERROR: пара ({i_glob}, {j_glob}) не найдена в портрете")
                continue

            ggu[pos] += A_loc[a][b]
            ggl[pos] += A_loc[b][a]
def compute_local_rhs_quad(elem_nodes: Sequence[int],
```

```
                     nodes_coords: Sequence[Tuple[float, float]]) -> List[float]:
    assert len(elem_nodes) == 4

    x_std, y_std, nid_std = _element_geometry(elem_nodes, nodes_coords)

    a = math.sqrt(3.0 / 5.0)
    b = math.sqrt(1.0 / 5.0)

    w_corner = 25.0 / 324.0
    w_edge   = 40.0 / 324.0
    w_inner  = 64.0 / 324.0

    gauss_pts = [
        (-a, -a), ( a, -a), ( a,  a), (-a,  a),
        (-a,  0.0), ( a,  0.0), ( 0.0, -a), ( 0.0,  a),
        (-b, -b), ( b, -b), ( b,  b), (-b,  b),
    ]
    gauss_w = [
        w_corner, w_corner, w_corner, w_corner,
        w_edge,   w_edge,   w_edge,   w_edge,
        w_inner,  w_inner,  w_inner,  w_inner,
    ]

    b_std = [0.0] * 4

    for (xi, eta), w in zip(gauss_pts, gauss_w):
        N1 = 0.25 * (1 - xi) * (1 - eta)
        N2 = 0.25 * (1 + xi) * (1 - eta)
        N3 = 0.25 * (1 + xi) * (1 + eta)
        N4 = 0.25 * (1 - xi) * (1 + eta)
        N = [N1, N2, N3, N4]

        x = sum(N[i] * x_std[i] for i in range(4))
        y = sum(N[i] * y_std[i] for i in range(4))

        dN1_dxi  = -0.25 * (1 - eta)
        dN1_deta = -0.25 * (1 - xi)
        dN2_dxi  =  0.25 * (1 - eta)
        dN2_deta = -0.25 * (1 + xi)
        dN3_dxi  =  0.25 * (1 + eta)
        dN3_deta =  0.25 * (1 + xi)
        dN4_dxi  = -0.25 * (1 + eta)
        dN4_deta =  0.25 * (1 - xi)

        dN_dxi  = [dN1_dxi,  dN2_dxi,  dN3_dxi,  dN4_dxi]
        dN_deta = [dN1_deta, dN2_deta, dN3_deta, dN4_deta]

        dx_dxi = dx_deta = dy_dxi = dy_deta = 0.0
        for iN in range(4):
```

```
            dx_dxi  += dN_dxi[iN]  * x_std[iN]
            dx_deta += dN_deta[iN] * x_std[iN]
            dy_dxi  += dN_dxi[iN]  * y_std[iN]
            dy_deta += dN_deta[iN] * y_std[iN]

        J11 = dx_dxi
        J12 = dx_deta
        J21 = dy_dxi
        J22 = dy_deta
        detJ = J11 * J22 - J12 * J21

        f_val = f_rhs(x, y)

        for iN in range(4):
            b_std[iN] += f_val * N[iN] * detJ * w

    pts_std = list(zip(x_std, y_std, nid_std, range(4)))
    nid_to_std_index = {p[2]: p[3] for p in pts_std}

    b_elem = [0.0] * 4
    for i_e, nid in enumerate(elem_nodes):
        s_idx = nid_to_std_index[nid]
        b_elem[i_e] = b_std[s_idx]

    return b_elem
```

Краевые условия

```
def add_neumann_boundary_contributions(b: List[float],
                                       elements,
                                       nodes_coords,
                                       mask_neumann,
                                       eps_axis=1e-8):
    for elem_nodes, _mat in elements:
        edges = [
            (elem_nodes[0], elem_nodes[1]),
            (elem_nodes[1], elem_nodes[2]),
            (elem_nodes[2], elem_nodes[3]),
            (elem_nodes[3], elem_nodes[0]),
        ]
        for ni, nj in edges:
            if not (mask_neumann[ni] and mask_neumann[nj]):
                continue

            x1, y1 = nodes_coords[ni]
            x2, y2 = nodes_coords[nj]

            on_x_axis = abs(y1) < eps_axis and abs(y2) < eps_axis
            on_y_axis = abs(x1) < eps_axis and abs(x2) < eps_axis

            if not (on_x_axis or on_y_axis):
```

```
            continue

        if on_x_axis:
            g_value = -1.0
        else:
            g_value = 0.0

        add_neumann_on_edge(b, ni, nj, nodes_coords, g_value)

def apply_dirichlet_conditions(di, ggl, ggu, ig, jg, b,
                               nodes_coords,
                               mask_dirichlet):
    n = len(di)
    for i in range(n):
        if not mask_dirichlet[i]:
            continue

        x, y = nodes_coords[i]
        u_val = u_exact(x, y)

        for p in range(ig[i], ig[i + 1]):
            ggu[p] = 0.0

        for row in range(n):
            if row == i:
                continue
            for p in range(ig[row], ig[row + 1]):
                if jg[p] == i:
                    ggl[p] = 0.0

        di[i] = 1.0
        b[i] = u_val
```

Сборка глобальной матрицы

```
    for elem_nodes, _mat in elements:
        K_geom = compute_local_stiffness_quad(elem_nodes, nodes, lam=1.0)
        M_geom = compute_local_mass_quad(elem_nodes, nodes)
        b_loc  = compute_local_rhs_quad(elem_nodes, nodes)

        L = sorted(elem_nodes)
        index_in_L = {node: idx for idx, node in enumerate(L)}
        k = len(elem_nodes)

        A_loc = [[0.0] * k for _ in range(k)]
        for i_loc, node_i in enumerate(elem_nodes):
            i_new = index_in_L[node_i]
            for j_loc, node_j in enumerate(elem_nodes):
                j_new = index_in_L[node_j]
                A_loc[i_new][j_new] = (
```

```
            K_geom[i_loc][j_loc] + M_geom[i_loc][j_loc]
        )

    b_reordered = [0.0] * k
    for i_loc, node_i in enumerate(elem_nodes):
        i_new = index_in_L[node_i]
        b_reordered[i_new] = b_loc[i_loc]

    add_local_matrix(di, ggl, ggu, ig, jg, L, A_loc)

    for a in range(k):
        i_glob = L[a]
        b[i_glob] += b_reordered[a]

mask_dirichlet, mask_neumann = build_boundary_masks(nodes, radius=1.0)
add_neumann_boundary_contributions(b, elements, nodes, mask_neumann)
apply_dirichlet_conditions(di, ggl, ggu, ig, jg, b, nodes, mask_dirichlet)

n = len(di)
A = np.zeros((n, n))
for i in range(n):
    A[i, i] = di[i]
    for p in range(ig[i], ig[i + 1]):
        j = jg[p]
        A[i, j] = ggu[p]
        A[j, i] = ggl[p]

u = np.linalg.solve(A, np.array(b))
```