КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет вычислительной математики и кибернетики

Р.З. ДАУТОВ

ПРОГРАММИРОВАНИЕ МКЭ В MATLAB

Учебное пособие

Казань — 2010

УДК 519.3

Р.З. Даутов. Программирование МКЭ в MATLAB. 71 с.

В пособии излагаются основные этапы построения и программной реализации схем метода конечных элементов приближенного решения краевых задач для линейных эллиптических уравнений второго порядка.

Пособие рассчитано на студентов старших курсов и аспирантов, специализирующихся в области методов численного решения задач математической физики.

Научный редактор:
доктор физико-математических наук М.М. Карчевский

Рецензенты:
доктор физико-математических наук М.Р. Тимербаев
кандидат физико-математических наук Е.В. Стребков

Печатается по постановлению редакционно-издательского совета факультета ВМК Казанского университета

Оглавление

ПРЕДИСЛОВИЕ

В пособии излагаются основные этапы построения и программной реализации схем метода конечных элементов приближенного решения краевых задач для линейных эллиптических уравнений второго порядка. Пособие рассчитано на студентов старших курсов и аспирантов, специализирующихся в области методов численного решения задач математической физики и может рассматриваться как практическое дополнение к учебнику [2].

Предполагается, что читатели имеют некоторые навыки работы в системе MatLab. Фрагменты MatLab-программ в текст пособия вставлены непосредственно из редактора текстов MatLab и являются исполняемыми. Для этого был использован стилевой файл *mcode*. Он часто некорректно воспринимает кириллический текст, поэтому комментарии к программам написаны на английском.

Рукопись была внимательно прочитана М.М. Карчевским. Автор с благодарностью учел его замечания.

Замеченные ошибки, опечатки, а также комментарии и пожелания, просьба направлять по адресу *rdautov@ksu.ru*.

Глава 1

АЛГОРИТМИЧЕСКИЕ АСПЕКТЫ МЕТОДА КОНЕЧНЫХ ЭЛЕМЕНТОВ

§ 1. Определение МКЭ на основе лагранжевых элементов

1.1. Модельная задача.

Рассмотрим следующую модельную краевую задачу об определении функции $u = u(x)$ в плоской ограниченной области Ω с кусочно гладкой границей $\Gamma = \Gamma_0 \cup \Gamma_1$:

$$-\operatorname{div}(c(x)\,\nabla u) + b(x)\cdot\nabla u + a(x)u = f(x), \quad x = (x_1, x_2) \in \Omega, \quad (1.1)$$

$$u(x) = u_D(x),\ x \in \Gamma_0, \quad c(x)\,\nabla u\cdot\nu(x) + \sigma(x)u = \mu(x),\ x \in \Gamma_1. \quad (1.2)$$

Здесь Γ_0 (Γ_1) — не обязательно связное множество, может быть пустым (в этом случае соответствующее краевое условие опускается); Γ_0 считается замкнутым множеством; $\nu(x)$ — единичный вектор внешней нормали в точке $x \in \Gamma$; "$\cdot$" — скалярное произведение векторов,

$$\operatorname{div} b = \frac{\partial b_1}{\partial x_1} + \frac{\partial b_2}{\partial x_2}, \quad \nabla u = \Big(\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}\Big)^T.$$

Функции c, a, f, u_D, σ, μ, а также вектор-функция $b(x) = (b_1(x), b_2(x))^T$, предполагаются заданными и кусочно гладкими; решение задачи u и вектор-функция $c(x)\,\nabla u$ предполагаются непрерывными на Ω.

Метод конечных элементов исходит из обобщенной формулировки этой задачи.[1)] Она имеет следующий вид: найти функцию $u \in V$ такую, что для любого $v \in V^0$

$$\int_\Omega (c\nabla u\cdot\nabla v + b\cdot\nabla u v + a u v)\,dx + \int_{\Gamma_1} \sigma u v\,dx = \int_\Omega f v\,dx + \int_{\Gamma_1} \mu v\,dx\,. \quad (1.3)$$

[1)] она получается стандартной последовательностью действий: умножением уравнения на функцию $v \in V^0$, интегрированием полученного равенства по области Ω, применением формулы Остраградского-Гаусса $\int_\Omega \operatorname{div} bv\,dx = -\int_\Omega b\cdot\nabla v\,dx + \int_\Gamma b\cdot\nu v\,dx$, учетом краевых условий во внеынтегральном слагаемом.

Здесь V — аффинное множество функций, V^0 — пространство пробных функций ($H^1(\Omega)$ — пространство Соболева):

$$V = \{v \in H^1(\Omega) : \, v(x) = u_D(x), \; x \in \Gamma_0\},$$
$$V^0 = \{v \in H^1(\Omega) : \, v(x) = 0, \; x \in \Gamma_0\}.$$

1.2. Определение схемы МКЭ.

Пусть $\mathcal{T}_h$ — разбиение (триангуляция) области Ω на лагаранжевы конечные элементы τ одного типа (аффинно-эквивалентные, изопараметрические или криволинейные элементы); предполагается, что все граничные точки Γ_0 являются вершинами каких либо элементов.[1)] Введем обозначения: $\Omega_h = \bigcup\limits_{\tau \in \mathcal{T}_h} \tau$; Γ_0^h, Γ_1^h — части границы области Ω_h, соответствующие Γ_0 и Γ_1 (Γ_0^h замкнуто); $\omega_h = \bigcup\limits_{\tau \in \mathcal{T}_h} \omega_\tau$ — множество всех узлов интерполяции (сетка узлов в $\bar{\Omega}$), γ_h множество узлов интерполяции, принадлежащих Γ_0^h. Определим пространство конечных элементов (аппроксимацию $H^1(\Omega)$):

$$S_h = \{v \in C(\bar{\Omega}_h) \cap H^1(\Omega_h) : v\big|_\tau \in P_\tau \;\; \forall\, \tau \in \mathcal{T}_h\},$$

а также аппроксимации множества функций V и пространства V^0:

$$V_h = \{v \in S_h : \, v(x) = u_D(x), \; x \in \gamma_h\},$$
$$V_h^0 = \{v \in S_h : \, v(x) = 0, \; x \in \gamma_h\}.$$

Построение приближенного решения задачи (1.3) по методу конечных элементов сводится к отысканию такой функции $u_h \in V_h$, что для любого $v_h \in V_h^0$ имеет место равенство

$$\int\limits_{\Omega_h} (c \nabla u_h \cdot \nabla v_h + b \cdot \nabla u_h v_h + a u_h v_h) \, dx + \int\limits_{\Gamma_1^h} \sigma u_h v_h \, dx = \int\limits_{\Omega_h} f v_h \, dx + \int\limits_{\Gamma_1^h} \mu v_h \, dx. \quad (1.4)$$

[1)] под лагранжевым конечным элементом понимается тройка $(\tau, \omega_\tau, P_\tau)$, где τ — замкнутая область простой формы (обычно треугольник или четырехугольник, возможно криволинейные); ω_τ — множество точек на τ, называемых узлами интерполяции; P_τ — конечномерное пространство функций на τ такое, что любая функция из P_τ однозначно определяется через свои значения в точках ω_τ. Область τ также называют конечным элементом, а под h понимают максимальный диаметр элементов из $\mathcal{T}_h$.

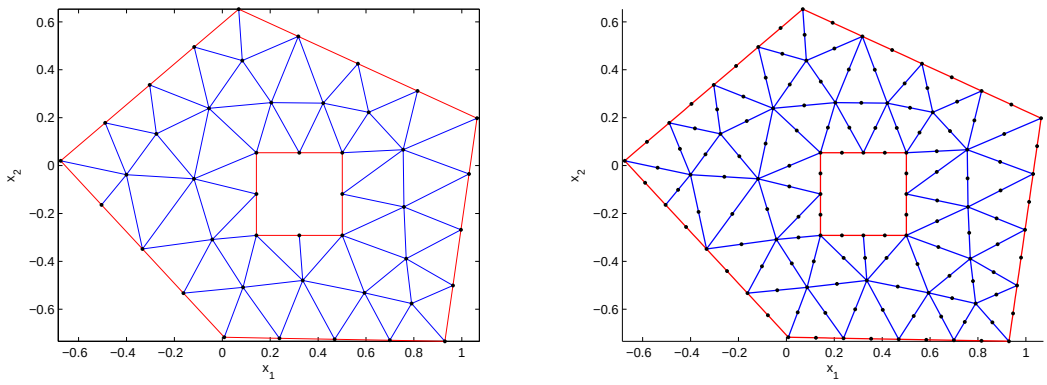

Рис. 1. Примеры триангуляции двусвязной многоугольной области. Конечные элементы — треугольники, узлы интерполяции выделены жирными точками. Сетку узлов на левом рисунке назовем P_1 сеткой, на правом — P_2 сеткой.

1.3. Примеры триангуляций области.

На левом рис. 1 приведен пример разбиения области на 3-х узловые конечные элементы. В этом случае элемент τ является одним из треугольников, ω_τ образуется его вершинами, $P_\tau = P_1 = \{p : p = c_1 + c_2 x_1 + c_3 x_2\}$ — множество полиномов первой степени. Отметим, что любые два элемента могут иметь общими либо целиком сторону (ребро), либо вершину. На каждом ребре располагаются 2 узла интерполяции.

На правом рис. 1 приведен пример разбиения области на 6-ти узловые конечные элементы: в ω_τ включается кроме вершин треугольника τ также средние точки его сторон (ребер), $P_\tau = P_2 = \{p : p = c_1 + c_2 x_1 + c_3 x_2 + c_4 x_1^2 + c_5 x_1 x_2 + c_6 x_2^2\}$ — множество полиномов второй степени; на каждом ребре располагаются 3 узла интерполяции. В обоих случаях область Ω_h совпадает с Ω, произвольная функция из P_τ однозначно определяется своими значениями в точках ω_τ.

1.4. Система МКЭ.

Равенства (1.4), определяющие схему МКЭ, сводятся к системе алгебраических уравнений. Для этого необходимо выбрать базис в S_h (размерность S_h равна числу узлов в ω_h). Базис Лагранжа, принятый в МКЭ, определяется следующим образом. Пронумеруем каким либо способом узлы ω_h от 1 до n_p (n_p зависит от h) и каждому узлу $a_i \in \omega_h$

поставим в соответствие базисную функцию $\varphi_i \in S_h$ так, что $\varphi_i(a_j) = \delta_{ij},\ j = 1, \ldots, n_p$.[1)] Значение произвольной функции $v_h \in S_h$ в узле $a_i \in \omega_h$ договоримся обозначать через v_i, а вектор $v = (v_i)_{i=1}^{n_p}$ будем называть вектором узловых параметров v_h. Тогда для любого $v_h \in S_h$ справедливо разложение

$$v_h(x) = \sum_{i=1}^{n_p} v_i \varphi_i(x), \quad x \in \bar{\Omega}_h.$$

Разобьем множество индексов $I = \{1, 2, \ldots, n_p\}$ на два подмножества:

$$i_d = \{i \in I :\ a_i \in \gamma_h\}, \quad i_n = \{i \in I :\ a_i \in \omega_h \setminus \gamma_h\}.$$

В узлах a_i, $i \in i_d$, задано краевое условие Дирихле ($u_h(a_i) = u_{D\,i}$, $u_{D\,i} = u_D(a_i)$) и решение задачи в этих узлах известно, $u_i = u_{D\,i}$, $i \in i_d$; необходимо определить значения u_h в точках с индексами из i_n. В $\bar{\Omega}_h$ справедливы разложения

$$u_h(x) = \sum_{i \in i_n} u_i \varphi_i(x) + \sum_{i \in i_d} u_{D\,i} \varphi_i(x), \quad v_h(x) = \sum_{i \in i_n} v_i \varphi_i(x), \tag{1.5}$$

для функций из V_h и V_h^0 соответственно. Определим также матрицу A (размера $n_p \times n_p$) и вектор Φ ($n_p \times 1$) с компонентами

$$a_{ij} = \int\limits_{\Omega_h} (c \nabla \varphi_j \cdot \nabla \varphi_i + b \cdot \nabla \varphi_j \varphi_i + a \varphi_j \varphi_i)\, dx + \int\limits_{\Gamma_1^h} \sigma \varphi_j \varphi_i\, dx, \tag{1.6}$$

$$\phi_i = \int\limits_{\Omega_h} f(x) \varphi_i\, dx + \int\limits_{\Gamma_1^h} \mu(x) \varphi_i\, dx, \tag{1.7}$$

Теперь, подставляя в (1.4) вместо u_h его разложение (1.5), а вместо v_h — поочередно φ_i, $i \in i_n$, придем к системе уравнений[1)]

$$\sum_{j \in i_n} a_{ij} u_j = F_i, \quad i \in i_n, \quad F_i = \phi_i - \sum_{j \in i_d} a_{ij} u_{D\,j}. \tag{1.8}$$

[1)] $\delta_{ij} = 1$ при $i = j$, иначе $\delta_{ij} = 0$. Базис Лагранжа имеет два важных свойства: если a_i не принадлежит некоторому конечному элементу (грани элемента), то φ_i тождественно равна нулю на этом элементе (на этой грани); т.о. диаметр области, на которой φ_i отлична от нуля, равен $2h$.

[1)] из системы (1.8) следуют равенства (1.4). Чтобы убедиться в этом достаточно умножить i-тое равенство в (1.8) на v_i и просуммировать по всем $i \in i_n$ и учесть определения (1.5), (1.6) и (1.7).

Решая эту систему находим неизвестные u_j, $j \in i_n$, а приближенное решение нашей задачи (1.3) — по первой формуле в (1.5).

Запишем систему (1.8) в матричном виде. Для этого нам нужно пронумеровать неизвестные по порядку. Пусть n_ω (n_d) — число элементов в i_n (i_d) и пусть $i_n = [i_1, i_2, \ldots, i_{n_\omega}]$, т.е. $i_n(k) = i_k$, $k = 1, 2, \ldots, i_{n_\omega}$ (аналогично, $i_d = [j_1, j_2, \ldots, j_{n_d}]$, $i_d(k) = j_k$, $k = 1, \ldots, i_{n_d}$). Образуем вектор столбец неизвестных u_0, матрицу K_0 и вектор F_0:

$$u_0 = \{u_{i_n(k)}\}_{k=1}^{n_\omega}, \quad K_0 = \{a_{i_n(k),\, i_n(l)}\}_{k,l=1}^{n_\omega}, \quad F_0 = \{F_{i_n(k)}\}_{k=1}^{n_\omega}.$$

Тогда система (1.8), очевидно, запишется в виде

$$K_0 u_0 = F_0. \tag{1.9}$$

Формулы (1.6), (1.7) дают некоторый способ вычисления элементов матрицы K_0 и вектора F_0, но ими непосредственно не пользуются при практических вычислениях, т.к. существует более удобный и более экономичный метод, который мы и рассмотрим. По традиции K_0 называют глобальной матрицей жесткости, F_0 — глобальным вектором сил.

§ 2. Алгоритм формирования системы МКЭ

Система алгебраических уравнений (1.9) полностью определяется матрицей A и вектором Ф. Поэтому сначала рассмотрим принятый в МКЭ способ их вычисления.

2.1. Алгоритм вычисления матрицы A и вектора Ф.

Из формулы (1.6) следует, что матрица A есть сумма двух квадратных матриц размерности $n_p \times n_p$: $A = K + H$,[1)]

$$K = \Big\{ \int\limits_{\Omega_h} (c\nabla\varphi_j \cdot \nabla\varphi_i + b \cdot \nabla\varphi_j\varphi_i + a\varphi_j\varphi_i)\, dx \Big\}, \quad H = \Big\{ \int\limits_{\Gamma_1^h} \sigma\varphi_j\varphi_i\, dx \Big\}.$$

[1)] отметим, что H — симметричная матрица; K является симметричной, если b есть тождественно нулевая вектор-функция.

Аналогично,

$$\Phi = F + G, \quad F = \Big\{ \int\limits_{\Omega_h} f\varphi_i \, dx \Big\}_{i=1}^{n_p}, \quad G = \Big\{ \int\limits_{\Gamma_1^h} \mu\varphi_i \, dx \Big\}_{i=1}^{n_p}.$$

Вычисление K и F. Согласно определению

$$K\,u \cdot v = \int\limits_{\Omega_h} (c\nabla u_h \cdot \nabla v_h + b \cdot \nabla u_h v_h + a u_h v_h)\, dx,$$

где $u, v \in R^{n_p}$ — векторы узловых параметров произвольно фиксированных функций $u_h, v_h \in S_h$. Пронумеруем все конечные элементы от 1 до n_t. Тогда

$$K\,u \cdot v = \sum_{\ell=1}^{n_t} \int\limits_{\tau_\ell} (c\nabla u_h \cdot \nabla v_h + b \cdot \nabla u_h v_h + a u_h v_h)\, dx. \tag{1.10}$$

Рассмотрим представление u_h и v_h на элементе $\tau_\ell \in \mathcal{T}_h$. Пусть на τ_ℓ имеется m_τ узлов интерполяции (m_τ одно и тоже для всех элементов) и пусть они перечислены в некотором порядке $a_{i_1}, a_{i_2}, \ldots, a_{i_{m_\tau}}$, одинаковом для всех элементов (локально пронумерованы). Зададим матрицу t размера $m_\tau \times n_t$, в ℓ-том столбце которого разместим номера узлов ℓ-го элемента $i_1, i_2, \ldots, i_{m_\tau}$; т.о. $t_{j\ell}$ определяет номер (индекс) j-го узла (в локальной нумерации) на элементе τ_ℓ.[1)] По определению базиса Лагранжа на элементе τ_ℓ имеют место разложения

$$u_h(x) = \sum_{\beta=1}^{m_\tau} u_{i_\beta}\varphi_{i_\beta}(x) = \sum_{\beta=1}^{m_\tau} u_{t_{\beta\ell}}\varphi_{t_{\beta\ell}}(x), \quad v_h(x) = \sum_{\alpha=1}^{m_\tau} v_{t_{\alpha\ell}}\varphi_{t_{\alpha\ell}}(x).$$

Подставляя эти разложения в (1.10), получим

$$K\,u \cdot v = \sum_{\ell=1}^{n_t} \sum_{\alpha,\beta=1}^{m_\tau} k_{\alpha\beta}^{\ell} u_{t_{\beta\ell}} v_{t_{\alpha\ell}}. \tag{1.11}$$

Образовавшаяся здесь матрица $K^\ell = \{k_{\alpha\beta}^\ell\}_{\alpha,\beta=1}^{m_\tau}$, где

$$k_{\alpha\beta}^{\ell} = \int\limits_{\tau_\ell} \left(c\,\nabla\varphi_{t_{\beta\ell}} \cdot \nabla\varphi_{t_{\alpha\ell}} + b \cdot \nabla\varphi_{t_{\beta\ell}}\varphi_{t_{\alpha\ell}} + a\,\varphi_{t_{\beta\ell}}\varphi_{t_{\alpha\ell}} \right) dx,$$

[1)] Матрицу t называют матрицей связности элементов, поскольку в ней хранится информация о том, какая группа узлов образуют тот или иной элемент.

называется матрицей жесткости элемента τ_ℓ (локальной матрицей жесткости). Преобразуем равенство (1.11). Введем в рассмотрение матрицу $\tilde{K}^\ell$ размера $n_p \times n_p$, состоящую из нулей, за исключением m_τ^2 элементов, которые определим так:

$$\tilde{K}^\ell_{t_{\alpha\ell},t_{\beta\ell}} = k^\ell_{\alpha\beta}, \quad \alpha,\beta = 1,\dots,m_\tau.$$

Тогда формула (1.11) примет вид

$$K\,u\cdot v = \sum_{\ell=1}^{n_t}\sum_{\alpha,\beta=1}^{m_\tau} \tilde{K}^\ell_{t_{\alpha\ell},t_{\beta\ell}} u_{t_{\beta\ell}} v_{t_{\alpha\ell}} = \sum_{\ell=1}^{n_t}\sum_{i,j=1}^{n_p} \tilde{K}^\ell_{ij} u_j v_i = \\ = \sum_{\ell=1}^{n_t} \tilde{K}^\ell y\cdot v = \Big(\sum_{\ell=1}^{n_t}\tilde{K}^\ell\Big)u\cdot v,$$

откуда в силу произвольности векторов u и v вытекает, что

$$K = \sum_{\ell=1}^{n_t} \tilde{K}^\ell.$$

Последнее равенство показывает, что глобальную матрицу жесткости можно получить суммированием по всем конечным элементам локальных матриц жесткости. Поскольку нули не имеет смысла суммировать, приходим к следующему алгоритму, известному как алгоритм сборки матрицы жесткости.

Алгоритм сборки матрицы жесткости (вклад элементов).

- Положить $K = 0$ (K — матрица размера $n_p \times n_p$).
- Для каждого $\ell = 1, 2, \dots, n_t$:
  - вычислить K^ℓ (размера $m_\tau \times m_\tau$).
  - для $\alpha,\beta = 1,\dots,m_\tau$ суммировать: $K_{t_{\alpha\ell},t_{\beta\ell}} = K_{t_{\alpha\ell},t_{\beta\ell}} + k^\ell_{\alpha\beta}$.

Аналогичные соображения приводят к алгоритму вычисления F.

Алгоритм сборки вектора сил (вклад элементов).

- Положить $F = 0$ (F — вектор столбец длины n_p).
- Для каждого $\ell = 1, 2, \ldots, n_t$:
  - вычислить F^ℓ (вектор длины m_τ).
  - для $\alpha = 1, \ldots, m_\tau$ суммировать: $F_{t_{\alpha\ell}} = F_{t_{\alpha\ell}} + F^\ell_\alpha$.

Здесь F^ℓ — вектор сил элемента τ_ℓ, имеет компоненты

$$F^\ell_\alpha = \int\limits_{\tau_\ell} f(x)\varphi_{t_{\alpha\ell}}(x)\,dx.$$

Вычисление H и G. Граница Γ_1^h области Ω_h является объединением граней e элементов из $\mathcal{T}_h$. Пусть $\Gamma_1^h = \cup_{k=1}^{n_e} e_k$, m_e — количество узлов интерполяции на каждой грани и пусть узлы e_k перечислены в некотором порядке $a_{i_1}, a_{i_2}, \ldots, a_{i_{m_e}}$, одинаковом для всех элементов. Зададим матрицу e размера $m_e \times n_e$, в k-том столбце которого разместим номера узлов k-го ребра $i_1, i_2, \ldots, i_{m_e}$; т.о. e_{jk} определяет номер (индекс) j-го узла (в локальной нумерации) на грани e_k. Поэтому на грани e_k

$$u_h(x) = \sum_{\alpha=1}^{m_e} u_{e_{\alpha k}}\varphi_{e_{\alpha k}}(x).$$

Точно так же, как при вычислении матрицы K, можем написать, что

$$H\,u \cdot v = \sum_{k=1}^{n_e} \int\limits_{e_k} \sigma(x)\,u_h(x)v_h(x)\,dx = \sum_{k=1}^{n_e}\sum_{\alpha,\beta=1}^{m_e} h^k_{\alpha\beta} u_{e_{\beta k}} v_{e_{\alpha k}},$$

где $H^k = \{h^k_{\alpha\beta}\}_{\alpha,\beta=1}^{m_e}$ — "матрица масс" грани e_k:

$$h^k_{\alpha\beta} = \int\limits_{e_k} \sigma(x)\,\varphi_{e_{\beta k}}\varphi_{e_{\alpha k}}\,dx.$$

Вводя расширенную матрицу $\tilde{H}^k$, как и ранее, получим:

$$H = \sum_{k=1}^{n_e} \tilde{H}^k.$$

Аналогично вычисляется и вектор G, при этом вводится "вектор сил" G^k грани e_k, определяемый компонентами

$$G^k_\alpha = \int\limits_{e_k} \mu(x)\, \varphi_{e_{\alpha k}}(x)\, dx.$$

Алгоритм сборки матрицы H и вектора G тот же, что и выше; H и G определяют вклады от граней элементов, принадлежащих Γ_1^h, в матрицу A и вектор Φ.

2.2. Алгоритм решения задачи.

На основе предыдущего, приходим к следующей последовательности действий для нахождения приближенного решения задачи (1.3):

1) определение геометрии области (области Ω и участков ее границы Γ_0 и Γ_1);

2) определение краевых условий;

3) построение триангуляции области; в частности, определение матриц связности элементов t и граничных ребер e;

4) формирование матрицы K и вектора F;

5) учет краевых условий: определение индексов узлов i_n и i_d; формирование H и G; построение K_0 и F_0;

6) решение системы уравнений $K_0 u_0 = F_0$; формирование вектора узловых параметров u решения u_h;

7) представления решения u_h в подходящем графическом виде.

Отметим, что способ задания области на 1-ом шаге важен только для 2-го и 3-го шага, поскольку он должен обеспечить нужную информацию в удобном виде для алгоритма триангуляции области. Выбор способа кодировки триангуляции является важным решением, поскольку он непосредственно влияет на шаги 4, 5, 7.

Для примера, рассмотрим краевую задачу в единичном круге Ω:

$$-\triangle u(x) = 1, \quad x \in \Omega, \quad u(x) = 0, \quad x \in \Gamma. \tag{1.12}$$

Она соответствует модельной задаче (1.1) при $c = 1$, $a = b = u_D = 0$, $f = 1$, $\Gamma_1 = \emptyset$.

Следующий код, на основе функций *pde toolbox*, решает (1.12), используя P_1 элементы. Шаги 4-6 реализованы в assempde. Результаты представлены на рис. 2.

```
g='circleg';        % set domain geometry
bc='circleb1';      % set the boundary conditions
c=1; a=0; f=1;      % set pde coefficients

[p,e,t]=initmesh(g,'Hmax',0.1); % set the mesh
u=-(p(1,:).^2+p(2,:).^2-1)/4;   % exact solution u=-(x_1^2+x_2^2)/4

uh=assempde(bc,p,e,t,c,a,f);    % assembles and solves the FEM system

figure;                         % plot the mesh
pdemesh(p,e,t), axis equal
xlabel('x_1'), ylabel('x_2')
title(['Number of mesh points np=' int2str(size(p,2))...
       ',  triangles nt=' int2str(size(t,2))])
figure;                         % plot the error.
pdesurf(p,t,u'-uh)
colormap('hsv')
xlabel('x_1'), ylabel('x_2'), zlabel('Error')
```

Далее мы достаточно подробно обсудим шаги 1-7 и продемонстрируем как их можно практически реализовать. В качестве системы программирования мы выбрали MatLab и предполагаем, что читатель знаком с основными принципами работы с ней. Кроме того в редких случаях (наиболее трудных для реализации), мы будем пользоваться программами (функциями) *pde toolbox* — расширения MatLab.

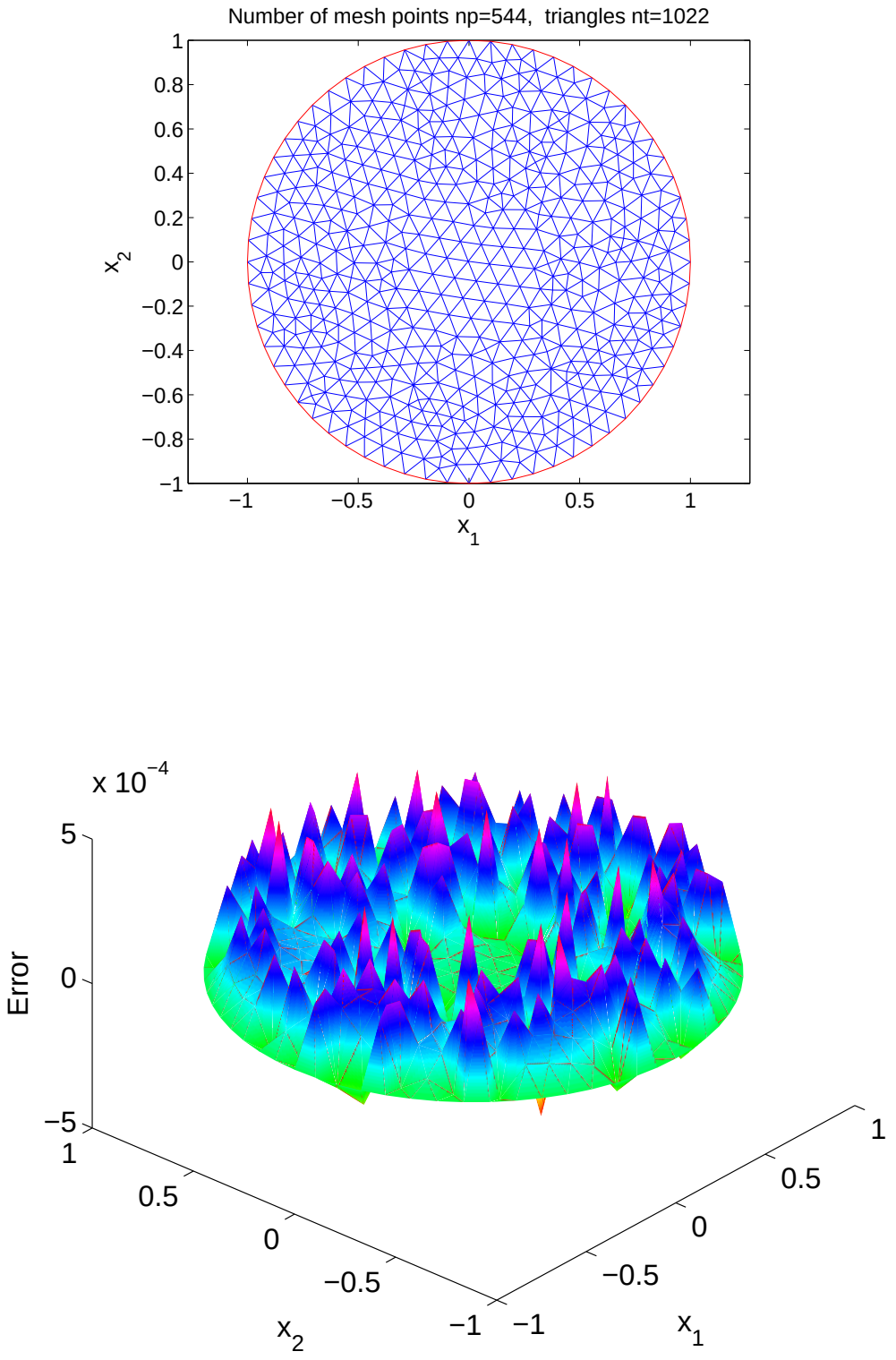

Рис. 2. P_1 триангуляция области и погрешность решения задачи (1.12) методом конечных элементов.

Глава 2

ПОСТРОЕНИЕ СЕТОК В MATLAB

§ 1. Создание и хранение разреженных матриц

В MatLab использование разреженных и плотных (полных) матриц фактически не различается, за исключением того, что

- разреженная матрица должна быть описана (создана) функцией sparse;
- имена некоторых функций для разреженных матриц могут быть другими.

Имеется три способа формирования разреженной матрицы.
1) Создание разреженной матрицы из полной. Например,

```
B =[1  3  0
    0  4  0
    2  0  5];   % B is full matrix
A = sparse(B) % A is sparse

A =  (1,1)        1
     (3,1)        2
     (1,2)        3
     (2,2)        4
     (3,3)        5
```

Для изображения разреженной матрицы A используется его так называемое координатное представление, когда указываются индексы и значения ненулевых элементов, которые перечисляются последовательно по столбцам. Таким образом A представляется на экране тремя массивами одинаковой длины (i, j) a.
2) Элементное наполнение матрицы. Предыдущую матрицу можно ввести также следующим образом.

```
A=sparse(3,3); % all elements of A equal 0.
A(1,1)=1; A(3,1)=2; A(1,2)=3; A(2,2)=4; A(3,3)=5;
```

3) Основной способ. В этом случае разреженная матрица получается из ее координатного представления. Например, следующие команды также создают матрицу A из предыдущих примеров.

```
i = [1  3  1  2  3];
j = [1  1  2  2  3];
a = [1  2  3  4  5];        % (i,j) a is the coordinate form of A.
A = sparse(i,j,a,3,3); % set A.
```

Здесь i, j могут быть как строками, так и столбцами с целыми положительными элементами. Более того, индексы не обязательно должны быть упорядочены, среди пар (i, j) могут быть совпадающие. В этом случае соответствующие им элементы a будут просуммированы при формировании A. Эта возможность позволяет формировать матрицу посредством накопления ее элементов. Например, A(1,1)=0.3+0.7=1. Тогда следующий код сформирует матрицу из предыдущих примеров

```
i = [1    1    3  1  2  3];
j = [1    1    1  2  2  3];
a = [0.3  0.7  2  3  4  5];
A = sparse(i,j,a,3,3);
```

Отметим, что a) i, j, a могут быть заданы как двумерные матрицы: в этом случае в функции sparse они будут восприниматься как вектор столбцы (как это принято в MatLab, согласно схеме хранения матриц по столбцам); b) аргумент a может быть скаляром: в этом случае он расширяется до вектора нужной длины с элементами равными этому скаляру.

Для больших разреженных матриц предпочтение необходимо отдать способу 3). Разница между 2) и 3) способом становится ясным, если разобраться с "внутренним" способом хранения разреженных матриц. Разреженные матрицы в MatLab хранятся в "упорядоченном столбцевом разреженном формате". А именно, $n \times n$ матрица A представляется тремя массивами (i, p, a), где в i хранятся строчные индексы ненулевых элементов, которые перечисляются по порядку по столбцам, в a хранятся сами ненулевые элементы, в p хранятся указатели на те позиции в i, с которых начинается новый столбец. Таким образом, упорядоченные по возрастанию строчные индексы ненулевых элементов в j-том столбце хранятся в $i\big(p(j) : p(j+1) - 1\big)$, а их значения в $a\big(p(j) : p(j+1) - 1\big)$; у пользователя не имеется доступа к этим массивам. Отметим, что массивы i, a те же, что и в координатной форме. Следующий код позволяет получить указатели p (в

предположении, что A не содержит нулевых столбцов).[1)]

```
[i,j,a] = find(A); % get the coordinate form of A (i,j have no copy)
p = find(diff([0;j;size(A,2)+1]))'

p =
     1     3     5     6
```

Проверим этот код на предыдущем примере:

```
for j = 1:n
    fprintf ('j=%d:\n      i      a\n', j) ;
    disp ([ i(p(j):p(j+1)-1) a(p(j):p(j+1)-1)])
end

j=1:
      i      a
      1      1
      3      2
j=2:
      i      a
      1      3
      2      4
j=3:
      i      a
      3      5
```

Разберем 2) способ, учитывая внутреннее хранение A. Рассмотрим код

```
A=sparse(n,n);   % 1)
. . . .
A(i1,j1)=a1;     % 2)
. . . .
A(i2,j2)=a2;     % 3)
. . . .
```

Команда 1) определяет матрицу A размера nxn, массивы (i, p, a), определяющие формат ее хранения, являются пустыми. На шаге 2) включается элемент a1, происходит формирование массивов (i, p, a). На 3-ем шаге включается новый элемент a2, что приводит к переформированию векторов (i, p, a): они меняют длину, выделяется память для них, части старых (i, p, a) копируются, вставляется новый элемент и т.д. Этот способ, как видим, оказывается дорогим, если $nnz(A)$ — число ненулевых элементов A, является большим числом.

Очевидный метод реализации 3) способа основывается на упорядочении триплета (i, j, a) по возрастанию индекса i, суммирования

[1)] если $x = [x_1, \ldots, x_n]$, то вектор $y = diff(x)$ имеет координаты $x_{i+1} - x_i$, $i = 1, \ldots, n-1$.

копий, и т.д. Наибольшую сложность здесь представляет процедура сортировки целочисленного массива, сложность остальных шагов порядка $nnz(A)$.

§ 2. Определение геометрии области, построение P_1 сеток

В *pde toolbox*, расширении MatLab, имеется функция initmesh, позволяющая строить треугольные сетки в достаточно сложных двумерных областях. Например, команды

```
[p,e,t]=initmesh('g','Hmax',0.1); pdemesh(p,e,t), axis equal
```

позволяют построить треугольную сетку в области, определяемой файлом геометрии g, и нарисовать ее в графическом окне. Максимальная сторона треугольников определяется параметром 'Hmax' и равна 0.1. Рассмотрим способы задания области.

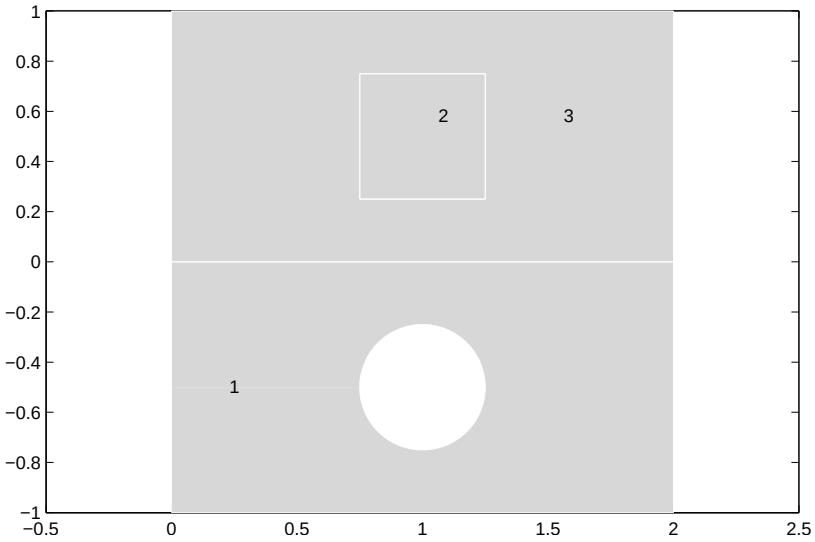

Рис. 1. Геометрия исходной области.

Для примера возьмем область, изображенную на рис. 1: она состоит из трех подобластей, отмеченных на рисунке цифрами 1,2,3 (метками).[1)] Как видим, области с метками 1,3 — прямоугольники одинакового размера, область 2 — квадрат, из области 1 удален круг радиуса 0.25. Эти области имеют границы, которые, в свою очередь, состоят из сегментов (прямолинейных или криволинейных отрезков).

[1)] В этих подобластях коэффициенты дифференциального уравнения могут принимать разные значения или определяться разными формулами

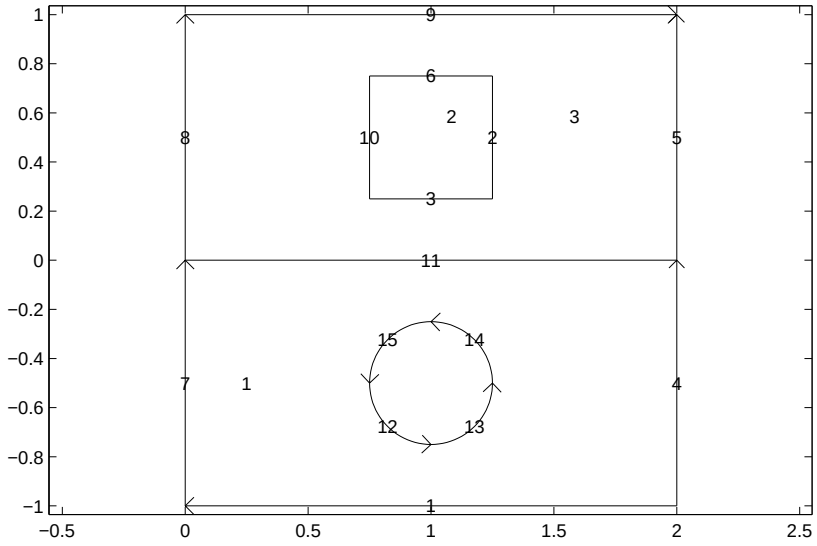

Рис. 2. Определение граничных сегментов и их номеров.

Различают внутренние сегменты (на рисунке они выделены белым цветом) и граничные (образующие границу всей области).[2)] Сегменты не могут пересекаться, и могут иметь лишь общие граничные точки. Замкнутые сегменты не допускаются; сегменты снабжаются номером (меткой).

Таким образом, прямоугольник задается минимум четырьмя сегментами, круг — двумя. Границы малых по размеру областей (в сравнении с размерами остальных) рекомендуется разбивать на 3 и более сегментов. На рис. 2 указано разбиение границ областей на сегменты. Всего получилось 15 сегментов, 5 из них — внутренние (с номерами 2, 3, 6, 10, 11). Предполагается, что каждый сегмент задается в параметрическом виде

$$x = x(s), \quad y = y(s), \quad s \in [s_0, s_1],$$

где s_0 и s_1 начальное и конечное значение параметра соответственно; при обходе сегмента от начальной точки (соответствующей s_0) до конечной точки (соответствующей s_1) слева и справа всегда будут две подобласти, которые он частично разделяет (считая дополнение ко всей области, которое имеет метку 0). Например, при обходе сегмента 5 в направлении стрелки слева остается область 3, справа — 0; при обходе сегмента 9 наоборот — слева 0, справа — 3.

[2)] На каждом сегменте можно определить в дальнейшем свое краевое условие.

Имеется две возможности задания геометрии:

1) при помощи матрицы (если все сегменты являются отрезками прямых или сегментами окружностей или эллипсов);

2) при помощи m.файла.

1 способ. Следующая матрица, формируемая функцией gtm, определяет геометрию области для нашего примера. Матрица содержит 15 столбцов (по числу сегментов) и каждый столбец определяет 1 сегмент. Первый элемент в столбце определяет тип сегмента (2 — отрезок прямой, 4 — эллипса); 2, 3 (4, 5) строки определяют x (y) координаты начальной и конечной точки соответственно; 6, 7 строки — номер области слева и справа по направлению обхода. Для отрезков прямых следующие строки не нужны и задаются нулями; для эллипса 8, 9 строки определяют x и y координаты центра эллипса, а 10, 11 строки — его x и y полуоси (для окружности они совпадают и равны радиусу окружности); 12 строка определяет угол поворота эллипса вокруг центра против часовой стрелки (в радианах).[1)]

```
function g=gtm
% geometry matrix
g=[2     2     2  2  2     2  2  2  2     2  2     4     4     4     4
   2  1.25  1.25  2  2  0.75  0  0  0  0.75  0  0.75     1  1.25     1
   0  1.25  0.75  2  2  1.25  0  0  2  0.75  2     1  1.25     1  0.75
  -1  0.75  0.25 -1  0  0.75 -1  0  1  0.25  0  -0.5 -0.75  -0.5 -0.25
  -1  0.25  0.25  0  1  0.75  0  1  1  0.75  0 -0.75  -0.5 -0.25  -0.5
   0     3     3  1  3     3  0  0  0     3  3     0     0     0     0
   1     2     2  0  0     2  1  3  3     2  1     1     1     1     1
   0     0     0  0  0     0  0  0  0     0  0     1     1     1     1
   0     0     0  0  0     0  0  0  0     0  0  -0.5  -0.5  -0.5  -0.5
   0     0     0  0  0     0  0  0  0     0  0  0.25  0.25  0.25  0.25
   0     0     0  0  0     0  0  0  0     0  0  0.25  0.25  0.25  0.25
   0     0     0  0  0     0  0  0  0     0  0     0     0     0     0 ];
```

Следующие команды строит сетку, представленную на рис. 3:

```
g=gtm;
[p,e,t]=initmesh(g,'Hmax',0.15);
pdemesh(p,e,t), axis equal
```

2 способ. К рис. 3 приводят также команды

```
g='gtf';
[p,e,t]=initmesh(g,'Hmax',0.15);
pdemesh(p,e,t), axis equal
```

[1)] если все сегменты прямые, то матрица может иметь лишь 7 строк.

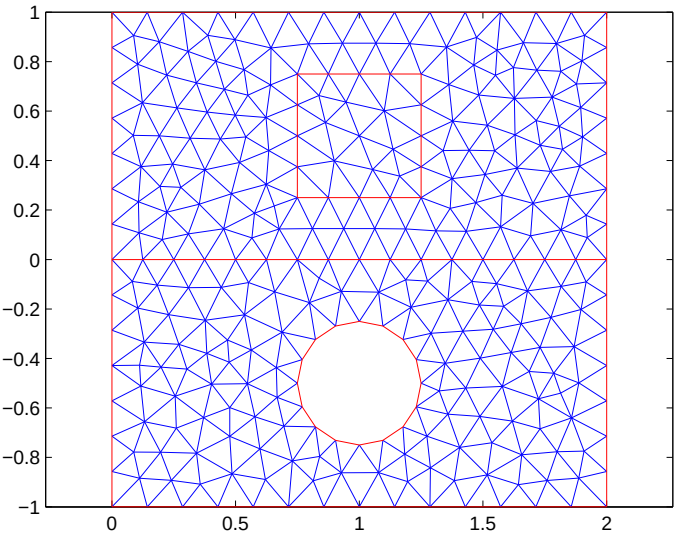

Рис. 3

где функция gtf.m определяет ту же геометрию, что и выше:

```
function [x,y]=gtf(bs,s)
% GTF Gives geometry data for the gtf PDE model
%
%   NE=GTF gives the number of boundary segment
%   D=GTF(BS) gives a matrix with one column for each boundary segment
%    specified in BS.
%   Row 1 contains the start parameter value.
%   Row 2 contains the end parameter value.
%   Row 3 contains the number of the left hand region.
%   Row 4 contains the number of the right hand region.
%
%   [X,Y]=GTF(BS,S) gives coordinates of boundary points. BS specifies the
%   boundary segments and S the corresponding parameter values. BS may be
%   a scalar.
%----------- Geometry data ----------------------------------
% rows d: s_0, s_1, left hand region, right hand region
d=[ 0   0   0   0   0   0   0   0   0   0   0    pi   3/2*pi  0     pi/2
    1   1   1   1   1   1   1   1   1   1   1  3/2*pi 2*pi  pi/2  pi
    0   3   3   1   3   3   0   0   0   3   3    0      0      0     0
    1   2   2   0   0   2   1   3   3   2   1    1      1      1     1 ];
% coordinates of start and end point's of line segments
xy=[2 1.25 1.25  2 2 0.75  0  0  0 0.75  0   % x coord. of start point
    0 1.25 0.75  2 2 1.25  0  0  2 0.75  2   % x coord. of end point
   -1 0.75 0.25 -1 0 0.75 -1  0  1 0.25  0   % y coord. of start point
   -1 0.25 0.25  0 1 0.75  0  1  1 0.75  0]; % y coord. of end point.
%----------------- STANDARD code ----------------------------
nbs=size(d,2);                % number of boundary segments
if nargin==0, x=nbs; return; end
if nargin==1, x=d(:,bs); return; end
x=zeros(size(s)); y=zeros(size(s)); [m,n]=size(bs);
if m==1 && n==1,bs=bs*ones(size(s)); end
%------------------------------------------------------------
```

```
if ~isempty(s),
  for is=1:11 % line segments
    i=find(bs==is);
    if ~isempty(i)
      x(i)=xy(1,is)+(xy(2,is)-xy(1,is))*s(i);
      y(i)=xy(3,is)+(xy(4,is)-xy(3,is))*s(i);
    end
  end
  for is=12:15 % circle segments
    i=find(bs==is);
    if ~isempty(i)
      xc=1; yc=-0.5; rc=0.25;
      x(i)= rc*cos(s(i))+xc;
      y(i)= rc*sin(s(i))+yc;
    end
  end
end
```

В начальных комментариях указаны способы вызова функции (эти способы вызова используются функциями *pde toolbox*; они реализованы в выделенном стандартном коде, который не нужно менять). Определение матрицы d является обязательным: в нем должно быть столько столбцов, сколько сегментов образует геометрию области. В нашем случае 15. Для каждого сегмента столбец содержит последовательно значение параметров s_0, s_1, метку областей слева и справа от сегмента. В матрице xy определены данные для прямолинейных сегментов: 1, 2 (3, 4) строки определяют x (y) координаты начальной и конечной точки сегмента соответственно. Далее, после стандартного кода, определяется каждый сегмент в параметрической форме.

Аналогично можно определить достаточно сложные по форме области. Тем не менее, необходимо сделать одно замечание по поводу параметризации сегментов, поскольку одну и ту же кривую можно параметризовать множеством способов. В отношении построения сеток наиболее удачным является выбор в качестве s параметра длины дуги сегмента (или пропорционального ему параметра, как например, выбор полярного угла в случае окружности). Однако такой выбор не всегда осуществим практически. В этом случае можно поступить следующим образом.

Рассмотрим область в виде эллипса с центром в начале координат и полуосями $a = 1, b = 0.1$. Напишем файл геометрии gell0 как и выше и построим сетку. Получим рис. 4:

```
[p,e,t]=initmesh('gell0','Hmax',0.1); pdemesh(p,e,t), axis equal
```

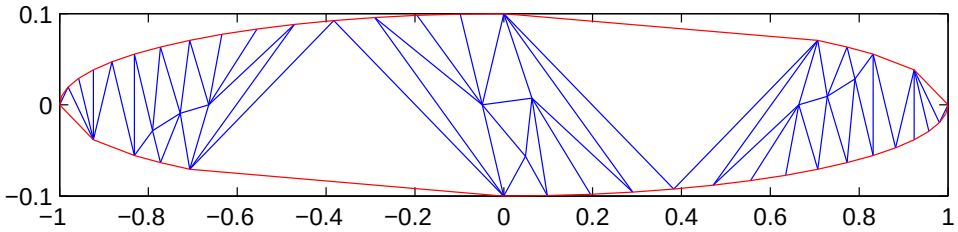

Рис. 4. Неудачная параметризация граничных сегментов.

```
function [x,y]=gell0(bs,s)
d=[   pi     3/2*pi   0      pi/2
    3/2*pi   2*pi     pi/2   pi
      1        1       1      1
      0        0       0      0  ];
%---------- STANDARD code ----------------------------
nbs=size(d,2);          % number of boundary segments
if nargin==0, x=nbs; return; end
if nargin==1, x=d(:,bs); return; end
x=zeros(size(s)); y=zeros(size(s)); [m,n]=size(bs);
if m==1 && n==1,bs=bs*ones(size(s)); end
%-----------------------------------------------------
if ~isempty(s),
  for is=1:4 % ellipse segments
    i=find(bs==is);
    if ~isempty(i)
      a=1; b=0.1;
      x(i)= a*cos(s(i));
      y(i)= b*sin(s(i));
    end
  end
end
```

Как видим, сетка весьма некачественная; это объясняется тем, что функция initmesh не может распределить равномерно узлы на границе области; угловая параметризация в данном случае не помогает в этом. Тем не менее, этой параметризацией можно воспользоваться, но со следующей модификацией функции gell0:

```
function [x,y]=gell1(bs,s)
d=[   pi     3/2*pi   0      pi/2
    3/2*pi   2*pi     pi/2   pi
      1        1       1      1
      0        0       0      0  ];
%---------- STANDARD code ----------------------------
nbs=size(d,2);          % number of boundary segments
if nargin==0, x=nbs; return; end
if nargin==1, x=d(:,bs); return; end
x=zeros(size(s)); y=zeros(size(s)); [m,n]=size(bs);
if m==1 && n==1,bs=bs*ones(size(s)); end
%-----------------------------------------------------
```

```
if ~isempty(s),
  for is=1:4 % ellipse segments
    i=find(bs==is);
    if ~isempty(i)
      a=1; b=0.1;
      % select 100 point's on segment is
      t=linspace(d(1,is),d(2,is),100);
      xt=a*cos(t);
      yt=b*sin(t);
      % use pdearcl to make the parameter s proportional to arc length
      t=pdearcl(t,[xt;yt],s(i),d(1,is),d(2,is));
      x(i)= a*cos(t);
      y(i)= b*sin(t);
    end
  end
end
```

Команды

```
[p,e,t]=initmesh('gell1','Hmax',0.1);
pdemesh(p,e,t), axis equal
```

приводят теперь к подходящей сетке, изображенной на рис.5. Функция

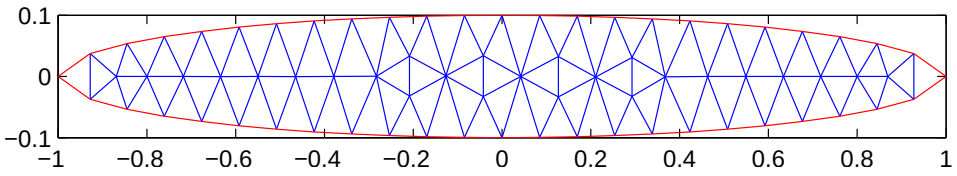

Рис. 5. Неудачная параметризация исправляется функцией pdearcl.

gell1 отличается от gell0 тем, что сначала выбираются 100 точек (xt,yt) на сегменте, которые следующей командой равномерно распределяются на нем по длине дуги (можно брать меньше или больше 100 точек, в зависимости от длины сегмента). Функция pdearcl расположена в *pde toolbox*.

§ 3. Кодировка сетки

Рассмотрим геометрию области, изображенной слева на рис. 6, а также треугольную сетку на ней (рис. справа). Файл геометрии 'lshapeg' этой области находится в *pde toolbox*; на левом рисунке указана номера граничных сегментов (всего их 10, из них 2 внутренних) и номера подобластей (их 3). Правый рисунок строят команды

```
[p,e,t]=initmesh('lshapeg','Hmax',inf); plotmeshP1(p,e,t,[1,1,1],12);
```

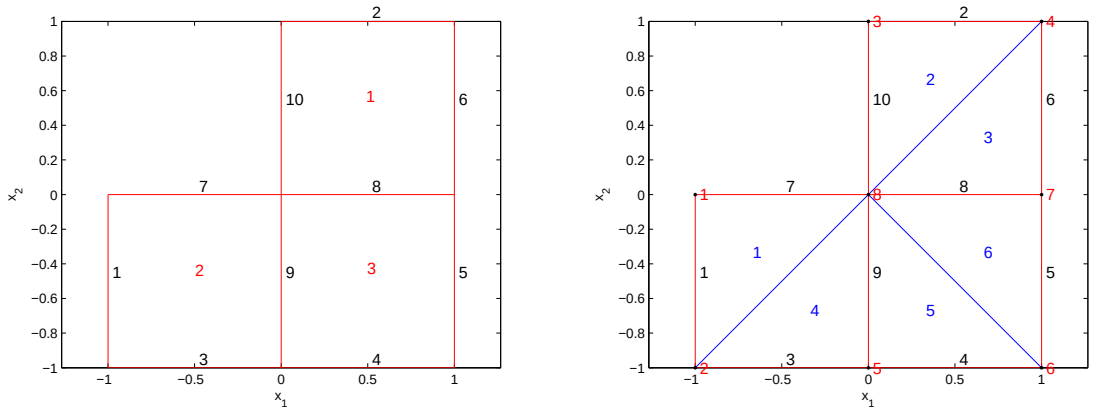

Рис. 6. L-образная область (слева) и его грубая триангуляция (справа).

Текст функции plotmeshP1 приведен ниже.

Укажем способ кодировки сетки в *pde toolbox* в матрицах (p, e, t):

- в p, размером $2 \times np$, хранятся координаты узлов сетки (вершин элементов): в $p(1, i)$ $(p(2, i))$ хранится x (y) координата i-го узла;
- в e, размером $7 \times ne$, пакуется информация о ребрах элементов, которые попали на границы подобластей (на внутренние и граничные сегменты).[1)] Точнее, в $e(1 : 2, k)$ располагаются номера вершин ребра с номером k; в $e(3 : 4, k)$ — соответствующие им значения параметра параметризации сегмента s; в $e(5, k)$ — номер сегмента, которому принадлежит ребро, а в $e(6 : 7, k)$ — номера подобластей слева и справа от ребра, если следовать параметризации. Напомним, что дополнение ко всей области имеет номер 0;
- в t, размером $4 \times nt$, хранятся номера вершин элемента и метка подобласти, которой он принадлежит. А именно, в $t(1 : 3, j)$ располагаются номера вершин j-го элемента, перечисленные в порядке обхода против часовой стрелки, в $t(4, j)$ — номер подобласти.

Для сетки на рис. 6 эти матрицы имеют следующий вид:

```
p =[-1 -1   0   1   0   1   1   0
    0 -1   1   1  -1  -1   0   0];

t =[1   4   7   2   5   6
    2   3   4   5   6   7
    8   8   8   8   8   8
    2   1   1   2   3   3];
```

[1)] В нашем случае это все сегменты с номерами 1 — 10 на левом рис. 6.

```
e =[1   3   2   5   6   7   1   8   5   8
    2   4   5   6   7   4   8   7   8   3
    0   0   0   0   0   0   0   0   0   0
    1   1   1   1   1   1   1   1   1   1
    1   2   3   4   5   6   7   8   9  10
    2   0   2   3   3   1   0   1   2   0
    0   1   0   0   0   0   2   3   3   1];
```

Рисунки 6 были получены при помощи функции plotmeshP1, в которой использованы MatLab-функции plot и text. Напомним, что функция

```
text(x,y,'abc','FontSize',fs,'Color','r')
```

выводит на график текстовую строку abc, начиная с точки с координатами x, y (остальные параметры необязательны).

```
function h=plotmeshP1(p,e,t,opt,fs)
% PLOTMESHP1: draw mesh (p,e,t) and
% opt(1)==1 --> draw points labels
% opt(2)==1 --> draw boundary edge labels
% opt(3)==1 --> draw triangle labels
% fs = FontSize

np=size(p,2); ne=size(e,2);  nt=size(t,2);

h=figure;
pdemesh(p,e,t); axis equal; hold on
plot(p(1,:),p(2,:),'.k','MarkerSize',8)
xlabel('x_1'), ylabel('x_2')

if opt(1)==1  % draw mesh-point labels
  for i=1:np
    text(p(1,i),p(2,i),[' ' int2str(i)],'FontSize',fs,'Color','r');
  end
end

if opt(2)==1  % draw boundary sdl
  for it=1:ne
    i=e(1,it); j=e(2,it);
    xx=(p(1,i)+p(1,j))/2; yy=(p(2,i)+p(2,j))/2;
    text(xx,yy+0.05,[' ' int2str(e(5,it))],'FontSize',fs,'Color','k');
  end
end

if opt(3)==1 % draw element labels
  for i=1:nt
    I=t(1:3,i); xy=sum(p(:,I)')/3;
    text(xy(1),xy(2),int2str(i),'FontSize',fs-1,'Color','b');
  end
end
```

Рассмотренный выше способ построения и кодировки треугольных сеток (назовем их P_1 сетками) позволяет реализовать метод конечных элементов для приближенного решения краевых задач для дифференциальных уравнений в частных производных второго порядка в двумерных областях. Точнее говоря, простейший метод, основанный на конформных линейных треугольных элементах (или, короче, P_1 элементах).[1)] Набор различного по характеру функций, позволяющий достигнуть этой цели, представлен в *pde tooldox*. Далее мы рассмотрим другой способ кодировки P_1 сеток, а также P_2 сетки.

§ 4. Сопряженная кодировка сетки

Рассмотренная выше кодировка сетки, представленная массивами (p, e, t), ориентирована на алгоритмы, в которых конечные элементы (треугольники) обрабатываются независимо друг от друга. Однако, имея только массивы (p, e, t), трудно, например, ответить на такие вопросы:

1) какие элементы содержат данный узел?

2) какие элементы имеют общее ребро с данным элементом? Является ли ребро элемента граничным и какому сегменту он принадлежит?

Чтобы быстро решать подобные задачи, необходимо дополнить массивы (p, e, t) другими.

Рассмотрим 1-ую задачу. Матрица t, называемая также матрицей связности элементов, является компактным способом представления следующей разреженной матрицы T размера $np \times nt$ с 3-мя ненулевыми элементами в каждом столбце: в j столбце располагается 1 в строках с номерами $t(1:3, j)$ (соответствующим номерам вершин j-го элемента). Не трудно заметить, что в i-той строке T единица находится только в тех позициях, которые соответствуют номерам элементов, имеющим узел i своей вершиной. Поэтому, если мы получим разреженный столбцевой формат хранения T' (транспонированной к T; она определяется лишь векторами (it, pt), т.к. ненулевые элементы

[1)] в этом случае на каждом элементе приближенная функция является алгебраическим полиномом 1-ой степени и определяется однозначно своими значениями в вершинах элемента.

T' равны 1), то сможем легко решить 1-ую задачу. Это можно сделать при помощи функции

```
function [it,pt]=trant(p,t)
% TRANT gives the transpose to the connectivity matrix t in
% sparse-odered-column format
np=size(p,2); nt=size(t,2);

j=[1:nt;1:nt;1:nt];
T=sparse(t(1:3,:),j,1,np,nt); % domain connectivity matrix
[it,j]=find(T'); % get the coordinate form of T'
pt=find(diff([0;j;np+1]))';
```

Протестируем эту функцию на сетке, изображенной слева на рис. 6; матрица T в этом случае имеет вид

```
T=[1   0   0   0   0   0
   1   0   0   1   0   0
   0   1   0   0   0   0
   0   1   1   0   0   0
   0   0   0   1   1   0
   0   0   0   0   1   1
   0   0   1   0   0   1
   1   1   1   1   1   1]
```

а результатом выполнения команд

```
[it,pt]=trant(p,t);
for i = 1:size(p,2)
  disp([i, it(pt(i):pt(i+1)-1)'])
end
```

будут следующие числа:

```
i       it(pt(i):pt(i+1)-1)

1       1
2       1   4
3       2
4       2   3
5       4   5
6       5   6
7       3   6
8       1   2   3   4   5   6
```

Они показывают, что i-тый узел сетки действительно является вершиной треугольников с номерами $it(pt(i) : pt(i+1)-1)$.

Для решения задач, указанных выше *во 2-ом пункте* и аналогичных им, введем два дополнительных массива ee и te. Определим их следующим образом:

- в ee, размером $7 \times K$, пакуется информация о всех ребрах элементов (а не только попавших на границы подобластей, как в e), K — число всех ребер. Точнее, в $ee(1:2,k)$ располагаются номера вершин ребра с номером k; в $e(3:4,k)$ — начальное и конечное значение параметра s на ребре k, если ребро k лежит на любом сегменте, иначе $e(3:4,k) = [0;0]$; в $ee(5,k)$ хранится либо номер сегмента, которому принадлежит ребро, либо 0; $e(6:7,k)$ — номера треугольников слева и справа от ребра (если смотреть от первой вершины ребра на вторую); соответствующее значение $ee(j,k) = 0$, если ребро граничное;
- в te, размером $4 \times nt$, хранятся номера ребер элемента и метка подобласти, которой он принадлежит. А именно, в $t(1:3,j)$ располагаются номера ребер элемента с номером j, в $t(4,j)$ — номер подобласти.

Три матрицы (p, ee, te) содержат в себе всю информацию о сетке и определяют альтернативную к (p, e, t) кодировку сетки; далее будем называть ее сопряженной P_1 сеткой. Такая кодировка оказывается полезной, когда независимо приходится обходить как элементы, так и ребра. Следующая функция вычисляет эти матрицы.

```
function [ee,te]=adjmeshP1(p,e,t)
% ADJMESHP1: gives an edge oriented mesh data.
%  P1-mesh (p,e,t) as in matlab function [p,e,t]=initmesh(g).
%  ee : 7xK, matrix of edges; every 2 mesh points i,j (or j,i)
%       form 1 edge; every edge belongs to 2 triangle,
%       or to 1 triangle (boundary edge); K - number of edges.
%       ee(1,i)  = index of the 1-st point on edge i,
%       ee(2,i)  = index of the 2-nd point on edge i.
%       ee(3:4,i)= start and end parameter, if i lies on border
%                  or boundary segment; else ee(3:4,i)=[0;0].
%       ee(5,i)  = border or boundary segment index, if edge i lies
%                  on segment; else ee(5,i)=0.
%       ee(6:7,i)= indices of left hand and right hand triangles;
%                  ee(6,i)=0 or ee(7,i)=0 for boundary edge.
%
%  te : 4xnt, triangle matrix, composed by edge indices;
%       te(1:3,i)= indices of edges on element i.
%       te(4,:) = sdl numbers

ne=size(e,2); nt=size(t,2);

% set ep = first 2 rows of the ee
tt=(t(1:3,:))';
ep = [tt(:,[1,2]); tt(:,[2,3]); tt(:,[3,1])]; % all edges - not unique
```

```
[ep,~,j]=unique(sort(ep,2),'rows');            % unique mesh edges
ep=ep';  j=j';  nep=size(ep,2);  % ep is sorted: 1 row and each column

% set te
mt = 1:nt;
te = [j(mt); j(mt+nt); j(mt+2*nt)]; % elements edges
te = [te; t(4,:)];

% set et = 6,7 rows of ee (first attempt)
et =zeros(2,nep); % edge2element connectivity matrix
for k=1:nt
  for j=1:3
    it=te(j,k);
    if   et(1,it)==0, et(1,it)=k;
    else et(2,it)=k;  end
  end
end

% sort first 2 rows of e as in ep
for j=1:ne
  if e(1,j)>e(2,j);
    c=e(1,j); e(1,j)=e(2,j); e(2,j)=c;
    c=e(3,j); e(3,j)=e(4,j); e(4,j)=c;
    c=e(6,j); e(6,j)=e(7,j); e(7,j)=c;
  end
end
[~,ib,it] = intersect(ep',e(1:2,:)','rows');

% set 3:5 rows of ee
ee=[ep;zeros(5,nep)];
ee(3:5,ib)=e(3:5,it);

% set 6,7 rows of ee
for ii=1:nep  % ii=edge
  t1=et(1,ii); t2=et(2,ii);  % neighbours triangles
  i=ep(1,ii); j=ep(2,ii); k=setdiff(tt(t1,:), [i j]); % t1 vortex indices
  % oriented area of triangle (i,j,k)
  d12 = p(:,j)-p(:,i);
  d13 = p(:,k)-p(:,i);
  A = d12(1,:).*d13(2,:)-d12(2,:).*d13(1,:); % A/2=area
  if A>0, ee(6,ii)=t1; ee(7,ii)=t2;
  else    ee(6,ii)=t2; ee(7,ii)=t1; end
end
```

Для примера выполним следующие команды:

```
[p,e,t]=initmesh('lshapeg','hmax',inf);
[ee,te]=adjmeshP1(p,e,t);
plotadjmeshP1(p,e,t,ee,12);
```

где функция рисования имеет следующий вид:

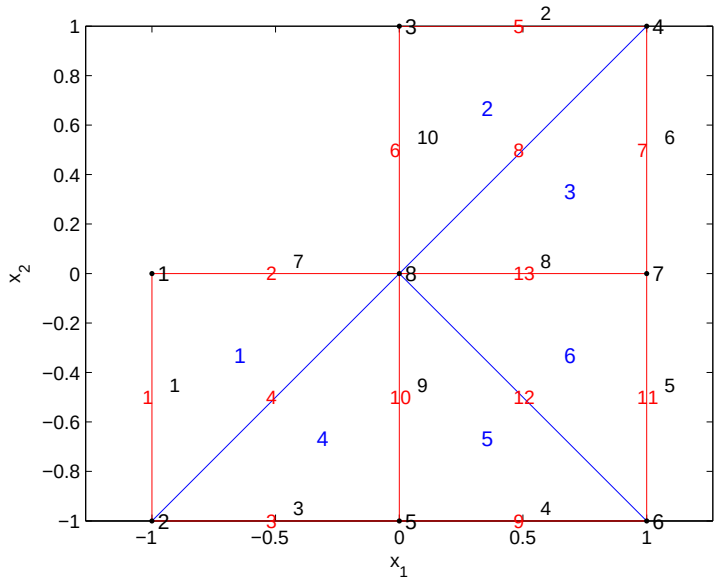

Рис. 7. Сопряженная P_1 сетка.

```
function h=plotadjmeshP1(p,e,t,ee,fs)
% PLOTADJMESHP1: draw adjoint mesh, element and edge labels

h=figure; pdemesh(p,e,t); axis equal; hold on
plot(p(1,:),p(2,:),'.k','MarkerSize',8); xlabel('x_1'), ylabel('x_2')

for i=1:size(p,2)
  text(p(1,i),p(2,i),[' ' int2str(i)],'FontSize',fs-1,'Color','k');
end

for it=1:size(ee,2) % draw edge labels
  i=ee(1,it); j=ee(2,it);
  text((p(1,i)+p(1,j))/2-0.06,(p(2,i)+p(2,j))/2,[' ' int2str(it)],...
      'FontSize',fs-2,'Color','r');
  if ee(5,it), text((p(1,i)+p(1,j))/2+0.05,(p(2,i)+p(2,j))/2+0.05,...
      [' ' int2str(ee(5,it))],'FontSize',fs-2,'Color','k');
  end
end

for i=1:size(t,2) % draw  element labels
  I=t(1:3,i);
  text(sum(p(1,I))/3,sum(p(2,I))/3,int2str(i),'FontSize',fs-1,'Color','b');
end
```

В результате получим рис. 7, на котором подписаны номера вершин, элементов и ребер; граничные ребра маркированы двумя числами — номером ребра и сегмента (чуть в стороне, мельче шрифт).

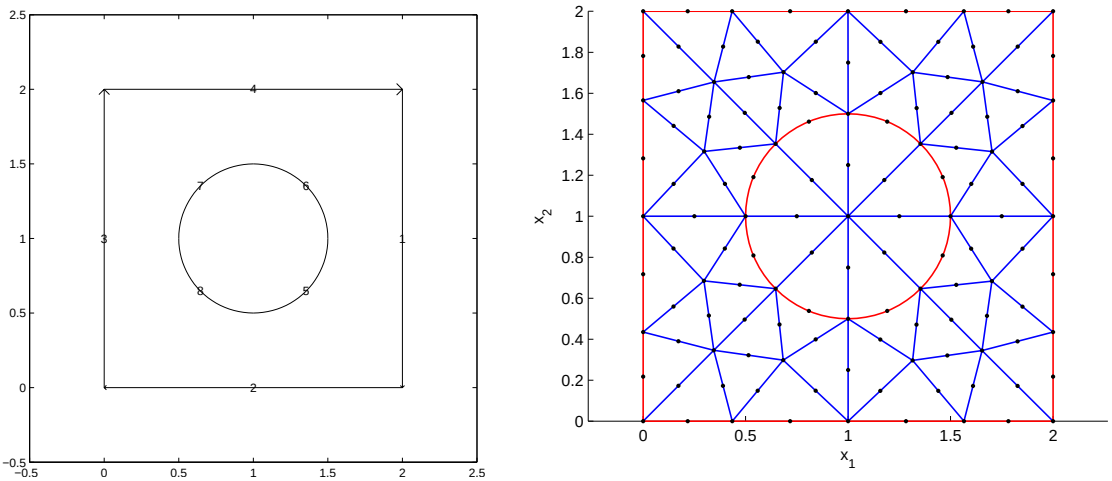

Рис. 8. Геометрия области и P_2 сетка на ней.

§ 5. P_2 сетки

На каждом треугольнике триангуляции области выделим 6 узлов — вершины треугольников и середины их сторон; предполагается, что если ребро элемента опирается на граничный сегмент (внутренний или внешний), то средняя точка сдвигается на этот сегмент согласно его парамеризации. Полученную сетку узлов назовем P_2 сеткой.

На рис. 8 приведен пример P_2 сетки. Геометрия области определяется следующей матрицей

```
g =[2   2   2   2     4     4     4     4
   2   2   0   0     1   1.5     1   0.5
   2   0   0   2   1.5     1   0.5     1
   2   0   0   2   0.5     1   1.5     1
   0   0   2   2     1   1.5     1   0.5
   0   0   0   0     2     2     2     2
   1   1   1   1     1     1     1     1
   0   0   0   0     1     1     1     1
   0   0   0   0     1     1     1     1
   0   0   0   0   0.5   0.5   0.5   0.5
   0   0   0   0   0.5   0.5   0.5   0.5
   0   0   0   0     0     0     0     0]
```

Такая сетка может быть получена из соответствующей P_1 сетки введением новых точек и сдвигом средних приграничных узлов. Их кодировка может быть сделана аналогичной кодировке P_1 сеток при помощи 3-х матриц $(p2, e2, t2)$, где $p2$ определяет координаты всех точек сетки, $e2$ задает ребра на границах подобластей, матрица $t2$

определяет связность элементов, а ее столбцы содержат номера узлов на элементе и номер подобласти. Заметим, что каждый новый узел (по сравнению с P_1 сеткой), связан со своим ребром; поэтому при нумерации новых узлов оказывается полезной реберная (сопряженная) кодировка P_1 сетки. Сопряженная кодировка P2-сеток определяется так же, как и в случае P_1 сеток.

Следующая функция позволяет реализовать сказанное выше; с ее помощью можно построить как P_2 сетку, так и ее сопряженную. Она в достаточной мере самодокументирована; при ее чтении полезно пользоваться справочной информацией MatLab по поводу незнакомых Вам функций (возможно таких, как unique, intersect, setdiff, pdeigeom).

```
function [p2,e2,t2,ee,te]=P1meshToP2(g,p,e,t)
%  P1MESHTTOP2: convert P1-mesh (p,e,t) to P2-mesh
%
%  [p2,e2,t2]=P1meshToP2(g,p,e,t) <-- gives P2-mesh
%  [p2,e2,t2,ee,te]=P1meshToP2(g,p,e,t) <-- gives P2 & adjoint P2-mesh (ee,te)
%
%  p,e,t: P1-mesh as in MatLab function [p,e,t]=initmesh(g).
%  p2   : 2xnp2, matrix of nodes coordinates
%  t2   : 7xnt,  matrix of triangles connectivity:
%         t2(1:3,:)=t(1:3,:) are vertex indices,
%         t2(4,:)=point indices between t2(1,:) and t2(2,:),
%         t2(5,:)=point indices between t2(2,:) and t2(3,:),
%         t2(6,:)=point indices between t2(3,:) and t2(1,:),
%         t2(7,:) - element sdl numbers
%  e2   : 7xne,  matrix of border and boundary edges:
%         e2(1:2,:) = vortex indices
%         e2(3,:)   = edge midpoint indices
%         e2(4:5,:) = start and end parameter values
%         e2(6,:)   = segment indices
%         e2(7:8,:)= indices of left hand and right hand subdomain
%
%  ee   : 8xK, matrix of edges; K - number of edges.
%        ee(1,i)  = index of the 1-st point on the edge i,
%        ee(2,i)  = index of the 2-nd point on the edge i.
%        ee(3,i)  = index of the midpoint on the edge i.
%        ee(4:5,i)= start and end parameter values, if the edge i lies on
%                   border or boundary segment; else ee(4:5,i)=[0;0].
%        ee(6,i)  =  border or boundary segment index, if the edge i lies
%                   on segment; else ee(6,i)=0
%        ee(7:8,i)= indices of left hand and right hand triangles.
%
%  te   : 4xnt,  matrix of triangles, composed by edge indices;
%        te(1:3,i)= indices of edges on element i.
%        te(4,:)  = sdl numbers
```

```
np=size(p,2); ne=size(e,2); nt=size(t,2);

%------ set elements of edge oriented mesh data ----------
% matrices ep, te, et are the same as in function adjmeshP1
%---------------------------------------------------------
% set ep = first 2 rows of ee
tt=(t(1:3,:))';
ep = [tt(:,[1,2]); tt(:,[2,3]); tt(:,[3,1])]; % all edges - not unique
[ep,~,j]=unique(sort(ep,2),'rows');           % unique mesh edges
ep=ep'; j=j'; nep=size(ep,2); % ep is sorted: 1 row and each column

% set te
mt = 1:nt;
te = [j(mt); j(mt+nt); j(mt+2*nt)]; % elements edges

% set midpoint indices in elements connectivity matrix t2.
t2=[t(1:3,:); np+te; t(4,:)]; % midpoint <--> edge

% set et = 7,8 rows of ee (first attempt)
et =zeros(2,nep); % edge2element connectivity matrix
for k=1:nt
  for j=1:3
    it=te(j,k);
    if   et(1,it)==0, et(1,it)=k;
    else et(2,it)=k;  end
  end
end

% sort first 2 rows of e as in ep
for j=1:ne
  if e(1,j)>e(2,j);
    c=e(1,j); e(1,j)=e(2,j); e(2,j)=c;
    c=e(3,j); e(3,j)=e(4,j); e(4,j)=c;
    c=e(6,j); e(6,j)=e(7,j); e(7,j)=c;
  end
end
[~,ib,it] = intersect(ep',e(1:2,:)','rows');

% set midpoint indices in boundary connectivity matrix e2.
e2=[e(1:2,:); zeros(1,ne); e(3:7,:)];
te = [te; t(4,:)];

% set 3:6 rows of ee
ee=[ep;np+(1:nep);zeros(5,nep)]; ee(4:6,ib)=e2(4:6,it);

e2(3,it)=ee(3,ib);

% set 7,8 rows of ee
for ii=1:nep  % ii=edge
  it1=et(1,ii); it2=et(2,ii);  % neighbours triangles
  i=ep(1,ii); j=ep(2,ii); k=setdiff(tt(it1,:), [i j]); % t1 vortex indices
  % oriented area of triangle (i,j,k)
  d12 = p(:,j)-p(:,i);
  d13 = p(:,k)-p(:,i);
```

```
  A = d12(1,:).*d13(2,:)-d12(2,:).*d13(1,:); % A/2=area
  if A>0, ee(7,ii)=it1; ee(8,ii)=it2;
  else    ee(7,ii)=it2; ee(8,ii)=it1; end
end

% set midpoint coordinates.
nep=size(ep,2); p2=[p zeros(2,nep)];
for i=1:nep
  if ee(6,i)==0
    i1=ee(1,i); i2=ee(2,i);
    p2(:,np+i)=(p(:,i1)+p(:,i2))/2;
  else
    % set boundary edge "midpoint" coordinates
    [x,y]=pdeigeom(g,ee(6,i),0.5*(ee(4,i)+ee(5,i)));
    p2(:,np+i)=[x;y];
  end
end
```

Для рисования P_2 сеток можно использовать функцию plotmeshP2; она проста для понимания и легко может быть модифицирована.

```
function h=plotmeshP2(g,p,e,t,ee,opt,fs)
% PLOTMESHP2: draw P2-mesh (p,e,t) and
% opt(1)==1 --> draw point labels
% opt(2)==1 --> draw triangle labels
% opt(3)==1 --> draw border and boundary edge segment labels
% opt(4)==1 --> draw edge labels
% fs = FontSize

np=size(p,2);  nt=size(t,2); ne=size(e,2); nee=size(ee,2);
h=figure; axis equal; hold on, xlabel('x_1'), ylabel('x_2')

% get internal (not border or boundary) edges (ii) and plot
ii = setdiff(ee(1:2,:)',e(1:2,:)','rows');
X=[p(1,ii(:,1));p(1,ii(:,2))]; Y=[p(2,ii(:,1));p(2,ii(:,2))];
line(X,Y,'Color','b','LineWidth',1);

% plot border and boundary edges
npe=10; % number of point on an edge
X=[]; Y=[]; for i=1:ne,
  s=linspace(e(4,i),e(5,i),npe);
  [x1,y1]=pdeigeom(g,e(6,i),s);
  X=[X x1 NaN]; Y=[Y y1 NaN];
end
plot(X,Y,'Color','r','LineWidth',1);

% mark points
plot(p(1,:),p(2,:),'.k','MarkerSize',8)

if opt(1)==1  % draw points labels
  for i=1:np
    text(p(1,i),p(2,i),[' ' int2str(i)],'FontSize',fs,'Color','k');
  end
end
```

```
if opt(2)==1  % draw element labels
  for i=1:nt
    I=t(1:3,i);
    text(sum(p(1,I))/3,sum(p(2,I))/3,int2str(i),'FontSize',fs-1,'Color','r');
  end
end

if opt(3)==1    % draw border and boundary edge segment labels
  for i=1:nee
    k=ee(3,i);
    text(p(1,k)-0.05,p(2,k)+0.02,[' ' int2str(ee(6,i))],...
         'FontSize',fs-2,'Color','g');
  end
end

if opt(4)==1    % draw  edge labels
  for i=1:nee
    k=ee(3,i);
    text(p(1,k),p(2,k)+0.05,[' ' int2str(i)],'FontSize',fs-2,'Color','b');
  end
end
```

Для теста рассмотрим L-образную область, изображенную на рис. 6. С помощью команд

```
g='lshapeg';
[p,e,t]=initmesh(g,'hmax',inf);
[p2,e2,t2,ee,te]=P1meshP2(g,p,e,t);
plotmeshP2(g,p2,e2,t2,ee,[1,1,0,1],12);
```

получим рис. 9, на котором подписаны номера вершин, элементов и ребер (помельче шрифт). Получены следующие массивы, определяющие кодировку:

```
e2=[1   3   2   5   6   4   1   7   5   3
    2   4   5   6   7   7   8   8   8   8
    9  13  11  17  19  15  10  21  18  14
    0   0   0   0   0   1   0   1   0   1
    1   1   1   1   1   0   1   0   1   0
    1   2   3   4   5   6   7   8   9  10
    2   0   2   3   3   0   0   3   2   1
    0   1   0   0   0   1   2   1   3   0]

t2=[ 1   4   7   2   5   6
     2   3   4   5   6   7
     8   8   8   8   8   8
     9  13  15  11  17  19
    12  14  16  18  20  21
    10  16  21  12  18  20
     2   1   1   2   3   3];

ee=[1   1   2   2   3   3   4   4   5   5   6   6   7
    2   8   5   8   4   8   7   8   6   8   7   8   8
    9  10  11  12  13  14  15  16  17  18  19  20  21
    0   0   0   0   0   1   1   0   0   0   0   0   1
    1   1   1   0   1   0   0   0   1   1   1   0   0
    1   7   3   0   2  10   6   0   4   9   5   0   8
    1   0   4   1   0   2   0   3   5   4   6   5   6
    0   1   0   4   2   0   3   2   0   5   0   6   3]

te=[1  5   7   3   9  11
    4  6   8  10  12  13
    2  8  13   4  10  12
    2  1   1   2   3   3]
```

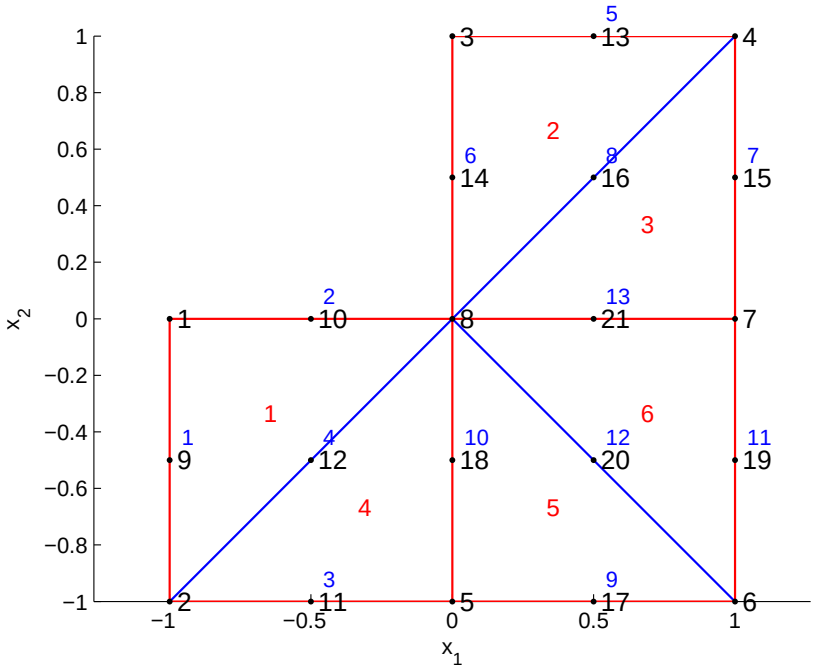

Рис. 9. P_2 сетка в L-образной области.

Глава 3

ПРОГРАММИРОВАНИЕ СБОРКИ МАТРИЦ МКЭ В MATLAB

§ 1. Программирование рассылки элементов

1.1. Рассылка элементов локальных матриц жесткости.

Для примера рассмотрим сборку глобальной матрицы жесткости для линейных 3-х узловых конечных элементов для конечно-элементной области, образованной разбиением сторон квадрата на nx частей. Будем считать, что все локальные матрицы жесткости одинаковы, чтобы оценить расходы непосредственно на рассылку элементов. Рассмотрим 5 способов сборки для различных сеток и фиксируем процессорное время их выполнения.

Конечно-элементное разбиение (триангуляция области) представляется тремя массивами (p, e, t) как это принято в *pde toolbox*. Для нас важно, что в $t(1:3, it)$ хранятся глобальные номера узлов на элементе с номером it.

1-ый алгоритм (bad1), естественный, написанный согласно определению алгоритма сборки глобальной матрицы жесткости. Так программировать сборку матриц в MATLAB не рекомендуется.

```
function K=assembabad1(p,t)
np=size(p,2);    % number of mesh points
nt=size(t,2);    % number of finite elements
dofe=3;          % number of mesh points on finite element

K =sparse(np,np);      % global stiffness matrix
Kt=ones(dofe,dofe);    % local stiffness matrix
for it=1:nt
  I=t(1:dofe,it);      % global point indices on element it
  for i=1:dofe, for j=1:dofe  % Assemble global stiffness matrix
      ii=I(i); jj=I(j);
      K(ii,jj)=K(ii,jj)+Kt(i,j);
  end, end
end
```

2-ый алгоритм (bad2), векторизованная версия предыдущего. Для совсем небольших сеток неплохой способ.

```
function K=assembabad2(p,t) np=size(p,2); nt=size(t,2); dofe=3;
K =sparse(np,np);       % global stiffness matrix
Kt=ones(dofe,dofe);     % local stiffness matrix
for it=1:nt
  I=t(1:dofe,it);       % global point indices on element it
  K(I,I)=K(I,I)+Kt;     % assemble global stiffness matrix
end
```

3-ый алгоритм (best1), основанный на сборке матрицы в координатной форме.

```
function K=assembabest1(p,t)
np=size(p,2); nt=size(t,2); dofe=3;
dofe2=dofe^2;    % number of elements of local stiffness matrix
m=dofe2*nt;      % number of stiffness matrices elements on all f.e.

% coordinates representation of global matrix K
i=zeros(m,1); j=zeros(m,1); a=ones(m,1);

Kt=rand(dofe,dofe);     % local stiffness matrix
for it=1:nt
  I=t(1:dofe,it);       % global point indices on element it

  % assemble global stiffness matrix in coordinate form (i,j,a)
  ii=repmat(I(:),1,dofe);  jj=ii';
  l=dofe2*(it-1)+(1:dofe2);
  i(l)=ii(:);
  j(l)=jj(:);
  a(l)=Kt(:);
end
K=sparse(i,j,a,np,np); % global stiffness matrix
```

Дадим некоторые пояснения. Для любой двумерной матрицы B, команда B=B(:) превращает ее в вектор столбец длины $numel(B)$, согласно способу хранения матриц по столбцам. Элемент $Kt(i,j)$ должен быть добавлен в позицию $(ii(i,j), jj(i,j))$ матрицы K. Команды

```
i(l)=ii(:); j(l)=jj(:); a(l)=Kt(:);
```

преобразуют матрицы ii, jj, Kt в столбцы и сохраняют их в соответствующих позициях векторов i, j, a.

4-ый алгоритм (best2), основанный на векторизованной сборке локальных матриц жесткости.

```
function K=assembabest2(p,t) np=size(p,2); nt=size(t,2); dofe=3;

it1=t(1,:); it2=t(2,:); it3=t(3,:); %itk=indices of local point k

% kij(k)= Kt(i,j) on finite element k
k12=ones(nt,1); K=  sparse(it1,it2,k12,np,np);
```

```
k13=ones(nt,1); K=K+sparse(it1,it3,k13,np,np);
k23=ones(nt,1); K=K+sparse(it2,it3,k23,np,np);

% replace next 3 lines to K=K+K.'; for symmetric matrix

k21=ones(nt,1); K=K+sparse(it2,it1,k21,np,np);
k31=ones(nt,1); K=K+sparse(it3,it1,k31,np,np);
k32=ones(nt,1); K=K+sparse(it3,it2,k32,np,np);

k11=ones(nt,1); K=K+sparse(it1,it1,k11,np,np);
k22=ones(nt,1); K=K+sparse(it2,it2,k22,np,np);
k33=ones(nt,1); K=K+sparse(it3,it3,k33,np,np);
```

В этой функции предполагается, что мы можем векторизовать сборку элементов локальных матриц жесткости. Для симметричной матрицы достаточно заменить среднюю тройку операторов на $K = K + K.'$. Именно такой стиль программирования принят в *pde toolbox*. Он не требует циклов и экономичен по затрачиваемой памяти (под все kij надо выделить один вектор).

Конечно, можно не думать о векторизации вычисления локальных матриц, жертвуя дополнительной оперативной памятью. Например, следующая версия сравнима по времени работы с предыдущей и даже экономичнее по требуемой памяти, чем best1.

```
function K=assembabest3(p,t) np=size(p,2); nt=size(t,2); dofe=3;

it1=t(1,:); it2=t(2,:); it3=t(3,:); %itk=indices of local point k
Kt=ones(nt,dofe,dofe);               % all local stiffness matrices

K=  sparse(it1,it2,Kt(:,1,2),np,np);
K=K+sparse(it1,it3,Kt(:,1,3),np,np);
K=K+sparse(it2,it3,Kt(:,2,3),np,np);

% replace next 3lines to K=K+K.'; for symmetric matrix

K=K+sparse(it2,it1,Kt(:,2,1),np,np);
K=K+sparse(it3,it1,Kt(:,3,1),np,np);
K=K+sparse(it3,it2,Kt(:,3,2),np,np);

K=K+sparse(it1,it1,Kt(:,1,1),np,np);
K=K+sparse(it2,it2,Kt(:,2,2),np,np);
K=K+sparse(it3,it3,Kt(:,3,3),np,np);
```

Следующая программа позволяет протестировать эффективность этих версий программы сборки.

```
function mainTestAssembK
% testing assembling functions.
clc
```

```
tbad1=[]; tbad2=[]; tbest1=[]; tbest2=[]; tbest3=[]; % assembling time
nt=[]; np=[];

for nx=[10 50 100 200 400]
  % set a regular (nx+1)*(nx+1) mesh on rectangular domain
  [p,~,t]=poimesh('squareg',nx,nx);
  np=[np size(p,2)];    % number of mesh points
  nt=[nt size(t,2)];    % number of finite elements

  tic, assembabad1(p,t);  tbad1=[tbad1  toc];
  tic, assembabad2(p,t);  tbad2=[tbad2  toc];
  tic, assembabest1(p,t);tbest1=[tbest1 toc];
  tic, assembabest2(p,t);tbest2=[tbest2 toc];
  tic, assembabest3(p,t);tbest3=[tbest3 toc];
end
disp('    np     nt    bad1      bad2     best1     best2    best3')
disp([np' nt' tbad1' tbad2' tbest1' tbest2' tbest3'])
```

В результате выполнения этой программы получена следующая таблица.

np	nt	bad1	bad2	best1	best2	best3
121	200	0.03125	0.015625	0	0	0
2601	5000	1.6719	1.1719	0.20313	0.015625	0.03125
10201	20000	23.516	22.828	0.79688	0.09375	0.09375
40401	80000	381.06	385.97	3.2344	0.45313	0.46875
160801	320000	*	*	13.125	2.0469	2.0938

В последних пяти столбцах указано время выполнения соответствующей программы в секундах CPU (поэтому при новом выполнении программы эти цифры могут незначительно измениться).

В следующей таблице приведено относительное время выполнения программ:

```
tbad1/np^2= [21.  2.5  2.3  2.3     ]*1e-7
tbest1/np = [0   7.8  7.8  8.0  8.2]*1e-5
tbest2/np = [0   0.6  0.9  1.1  1.3]*1e-5
tbest3/np = [0   1.2  0.9  1.2  1.3]*1e-5
```

Из этой таблицы с большим основанием можно заключить, что время выполнения tbad1 программы bad1 практически пропорционально квадрату числа узлов (элементов), тогда как время выполнения best1, best2, best3 пропорционально числу узлов (элементов); bad2 существенно короче bad1, но на подробных сетках не отличается от нее по времени выполнения; best2, best3 не содержат циклов, поэтому почти на порядок быстрее best1.

Таким образом, можно рекомендовать стиль программирования

принятый в best2 или best3, в зависимости от возможности векторизовать вычисление локальных матриц жесткости.

Для произвольных конечных элементов можно рекомендовать приведенную ниже функцию в предположении, что для вычисления локальной матрицы жесткости на элементе с номером it имеется функция $e = locsm(it, p, t)$.

```
function K=assemba(dofe,p,t)
% template for matrix-assembling function for dofe-point's FEM
np=size(p,2);    % number of mesh points
nt=size(t,2);    % number of finite elements
Kt=zeros(dofe,dofe,nt);  % all local stiffness matrices
for it=1:nt
  Kt(it,:,:)=locsm(it,p,t); % local stiff.matr. on element it
end

K=sparse(np,np); % global stiffness matrix
for i=1:dofe, for j=1:dofe
  K=K+sparse(t(i,:),t(j,:),Kt(:,i,j),np,np);
end, end
```

1.2. Рассылка элементов локальных векторов сил.

Программирование сборки глобального вектора сил осуществляется аналогично, за исключением того, что теперь собирается вектор, а не разреженная матрица. Можно рекомендовать три способа программирования, в зависимости от возможности векторизации вычисления компонент вектора сил. Эти способы представлены ниже.

```
% testing FEM assembling functions
tbest1=[]; tbest2=[];  tbest3=[];% assembling times
nt=[]; np=[];

for nx=[10 50 100 200 400]
  % set the regular (nx+1)*(nx+1) mesh on the domain
  [p,~,t]=poimesh('squareg',nx,nx);
  np=[np size(p,2)];    nt=[nt size(t,2)];

  tic; F=assembFbest1(p,t);  tbest1=[tbest1 toc];
  tic; F=assembFbest2(p,t);  tbest2=[tbest2 toc];
  tic; F=assembFbest3(p,t);  tbest3=[tbest3 toc];
end
disp('          np          nt        best1     best2     tbest3')
disp([np' nt' tbest1' tbest2' tbest3' ])

function F=assembFbest1(p,t)
```

```
np=size(p,2); nt=size(t,2);
it1=t(1,:); it2=t(2,:); it3=t(3,:);

%fj(k)= Ft(j) on the finite element k, Ft the local force vector
f1=ones(nt,1); F=  sparse(it1,1,f1,np,1);
f2=ones(nt,1); F=F+sparse(it2,1,f2,np,1);
f3=ones(nt,1); F=F+sparse(it3,1,f3,np,1);

function F=assembFbest2(p,t)
np=size(p,2); nt=size(t,2);
it1=t(1,:); it2=t(2,:); it3=t(3,:);

Ft=ones(nt,3); % all local force vectors

F=  sparse(it1,1,Ft(:,1),np,1);
F=F+sparse(it2,1,Ft(:,2),np,1);
F=F+sparse(it3,1,Ft(:,3),np,1);

function F=assembFbest3(p,t)
np=size(p,2); nt=size(t,2); dofe=3;
F=zeros(np,1);       % global force vector
for it=1:nt
  I=t(1:dofe,it);
  Ft=ones(dofe,1);  % calculate the local force vector
  F(I)=F(I)+Ft;
end
```

В результате выполнения этого кода получена следующая таблица

np	nt	best1	best2	tbest3
121	200	0	0	0
2601	5000	0	0	0.09375
10201	20000	0.046875	0.0625	0.29688
40401	80000	0.23438	0.21875	1.2656
160801	320000	1.0781	1.1094	5.0625

Из нее следует, что расходы по рассылке элементов по третьему способу примерно в 5 раз больше.

§ 2. Формирование системы МКЭ для P_1 элементов

Рассмотрим вопросы формирования глобальной и локальной матрицы жесткости и вектора сил, а также вопросы, касающиеся учета краевых условий.

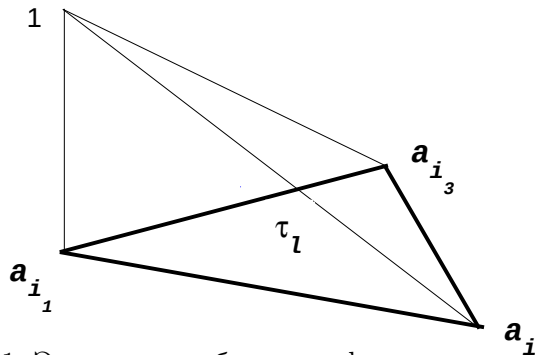

Рис. 1. Элемент τ_ℓ и базисная функция φ_{i_1}.

2.1. Расчетные формулы для P_1 элементов.

Рассмотрим задачу вычисления матрицы жесткости элемента $(\tau_\ell, \omega_{\tau_\ell}, P_1)$. Компоненты матрицы K^ℓ с номерами α, $\beta = 1, 2, 3$, определяются формулами

$$k^\ell_{\alpha\beta} = \int\limits_{\tau_\ell} \left(c\,\nabla\varphi_{i_\beta} \cdot \nabla\varphi_{i_\alpha} + b \cdot \nabla\varphi_{i_\beta}\varphi_{i_\alpha} + a\,\varphi_{i_\beta}\varphi_{i_\alpha} \right) dx.$$

Здесь i_1, i_2, i_3 — номера вершин элемента τ_ℓ; φ_{i_1}, φ_{i_2}, φ_{i_3} — базисные функции, соответствующие узлам с этими номерами (см. рис. 1).[1)] Для вычисления интеграла нужно иметь формулы для базисных функций и их градиентов и некоторый численный метод вычисления интеграла.

Для P_1 элементов используют два способа вычисления интеграла (согласно теории МКЭ; мы не рассматриваем возможность вычисления слагаемых интеграла по разным формулам). В 1-ом способе коэффициенты c, b, a на элементе полагаются постоянными и равными своим значениям в центре тяжести элемента $x_\tau = (a_{i_1} + a_{i_2} + a_{i_3})/3$. После чего интегралы вычисляются точно (такой способ принят в *pde toolbox*). Второй способ заключается в использовании квадратурной формулы с одним узлом x_τ, которая является точной на полиномах из P_1.[2)] В этом случае приходим к простой формуле:

$$k^\ell_{\alpha\beta} = |\tau_\ell| \left(c\,\nabla\varphi_{i_\beta} \cdot \nabla\varphi_{i_\alpha} + b \cdot \nabla\varphi_{i_\beta}\varphi_{i_\alpha} + a\,\varphi_{i_\beta}\varphi_{i_\alpha} \right)(x_\tau),$$

[1)] они являются аффинными функциями и удовлетворяют условию $\varphi_{i_\ell}(a_{i_k}) = \delta_{k\ell}$.

[2)] то есть $\int\limits_{\tau_\ell} p\,dx = |\tau_\ell|\, p(x_\tau)$ для любых $p \in P_1$.

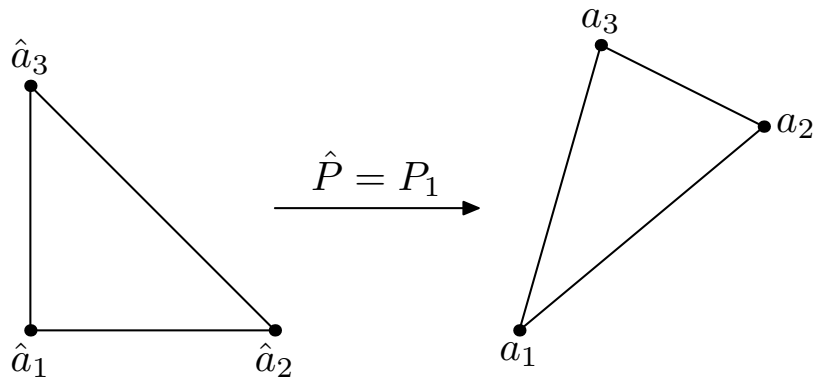

Рис. 2. Базисный элемент $\hat{\tau}$ (слева) и P_1 элемент τ (справа).

где $|\tau_\ell|$ есть площадь τ_ℓ.

Для получения формул для базисных функций можно воспользоваться следующим приемом, пригодным и для более сложных треугольных элементов с прямолинейными сторонами (например, для 6-ти узловых P_2 элементов; см. далее). Идея заключается в том, чтобы рассмотреть подобный τ_ℓ "канонический" (базисный) элемент $\hat{\tau}$, в координатах $\hat{x} = (\hat{x}_1, \hat{x}_2)$, для которого базисные функции просто определяются. Тогда базисные функции на τ_ℓ можно получить преобразованием координат.

С этой целью рассмотрим рис. 2, на котором изображен базисный элемент с вершинами $\hat{a}_1 = (0,0)$, $\hat{a}_2 = (1,0)$, $\hat{a}_3 = (0,1)$ (слева) и произвольный 3-х узловой P_1 элемент τ (справа) с вершинами a_1, a_2 и a_3 (для сокращения записи используем обозначения τ, a_α вместо τ_ℓ, a_{i_α}). Легко проверяется, что функции

$$\hat{\varphi}_1(\hat{x}) = 1 - \hat{x}_1 - x_2, \quad \hat{\varphi}_2(\hat{x}) = \hat{x}_1, \quad \hat{\varphi}_3(\hat{x}) = \hat{x}_2,$$

являются базисными (они являются аффинными функциями, $\hat{\varphi}_i(\hat{a}_j) = \delta_{ij}$). Также нетрудно видеть, что формула

$$x = a_1\hat{\varphi}_1 + a_2\hat{\varphi}_2 + a_3\hat{\varphi}_3 = a_1 + \hat{x}_1(a_2 - a_1) + \hat{x}_2(a_3 - a_1) \qquad (3.1)$$

задает преобразование элемента $\hat{\tau}$ на τ. Действительно, это отображение является аффинным и вершины $\hat{\tau}$ преобразуются в вершины τ, причем $(0,0) \to a_1$, $(1,0) \to a_2$, $(0,1) \to a_3$, т. е. отображение сохраняет ориентацию базисного элемента. Пусть

$$B_\tau = \begin{pmatrix} (a_2 - a_1)_1 & (a_3 - a_1)_1 \\ (a_2 - a_1)_2 & (a_3 - a_1)_2 \end{pmatrix}, \quad a_1 = \begin{pmatrix} (a_1)_1 \\ (a_1)_2 \end{pmatrix},$$

тогда

$$x = B_\tau \hat{x} + a_1 \ : \hat{\tau} \to \tau. \tag{3.2}$$

Якобиан этого отображения $J_\tau = \det(B_\tau) > 0$, и, следовательно, равен $2|\tau|$ — удвоенной площади элемента τ, поскольку

$$|\tau| = \int\limits_\tau d\,x = \int\limits_{\hat{\tau}} \det(B_\tau) d\,\hat{x} = |\hat{\tau}| \det(B_\tau) = \frac{1}{2} J_\tau. \tag{3.3}$$

Обратное отображение имеет вид $\hat{x} = B_\tau^{-1}(x - a_1)$, причем

$$B_\tau^{-1} = \frac{1}{\det(B_\tau)} \begin{pmatrix} (a_3 - a_1)_2 & -(a_3 - a_1)_1 \\ -(a_2 - a_1)_2 & (a_2 - a_1)_1 \end{pmatrix} =: \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}. \tag{3.4}$$

Следовательно,

$$\hat{x}_1 = d(x_1 - (a_1)_1) - b(x_2 - (a_1)_2), \quad \hat{x}_2 = -c(x_1 - (a_1)_1) + a(x_2 - (a_1)_2).$$

Базисные функция $\varphi_i(x)$ на элементе τ получаются заменой переменных из базисных функций на элементе $\hat{\tau}$ по формуле $\varphi_i(x) = \hat{\varphi}(B_\tau^{-1}(x - a_1))$. Поэтому

$$\begin{aligned} \varphi_1(x) &= 1 - \varphi_2(x) - \varphi_3(x), \\ \varphi_2(x) &= d(x_1 - (a_1)_1) - b(x_2 - (a_1)_2), \\ \varphi_3(x) &= -c(x_1 - (a_1)_1) + a(x_2 - (a_1)_2). \end{aligned}$$

Соответственно, градиенты этих функций имеют следующий вид:

$$\nabla\varphi_1 = \begin{pmatrix} c - d \\ b - a \end{pmatrix}, \quad \nabla\varphi_2 = \begin{pmatrix} d \\ -b \end{pmatrix}, \quad \nabla\varphi_3 = \begin{pmatrix} -c \\ a \end{pmatrix}.$$

Отметим, что $\varphi_i(x_\tau) = 1/3$, $i = 1, 2, 3$,[1)], а градиенты ($\hat{\nabla}$) функций $\hat{\varphi}_i$ связаны с $\nabla\varphi_i$ равенством

$$\nabla\varphi_i(x) = B_\tau^{-T} \hat{\nabla}\hat{\varphi}_i(\hat{x}), \quad \hat{x} = B_\tau^{-1}(x - a_1), \tag{3.5}$$

где $B_\tau^{-T} = (B_\tau^{-1})^T$. Для вычисления элементов локальной матрицы жесткости, таким образом, все данные подготовлены.

[1)] на базисном элементе это так, а точка $(1/3, 1/3)$ — центр тяжести элемента $\hat{\tau}$, при преобразовании координат переходит в x_τ — центр тяжести τ.

Заметим также, что для элементов τ любых типов

$$\sum_{i=1}^{m_\tau} \varphi_i(x) = 1, \quad \sum_{i=1}^{m_\tau} \nabla\varphi_i(x) = (0,0)^T, \quad x \in \tau, \tag{3.6}$$

где m_τ — число узлов интерполяции на элементе.

Отметим разницу между двумя способами вычисления интеграла, указанными выше. Поскольку базисные функции φ_{i_α} являются линейными, а $\nabla\varphi_{i_\alpha}$ — постоянными, то отличие состоит лишь в способе вычисления интеграла

$$m^\ell_{\alpha\beta} = \int\limits_{\tau_\ell} a\,\varphi_{i_\beta}\varphi_{i_\alpha}\,dx.$$

Матрица M^τ с этими элементами называется матрицей масс элемента τ. В случае использования квадратурной формулы все его элементы оказываются одинаковыми и равными $(J_\tau/18)\,a(x_\tau)$, а сама матрица является вырожденной (ранга 1). При "точном интегрировании"

$$m^\ell_{\alpha\beta} = a(x_\tau)\,J_\tau \int\limits_{\hat\tau} \hat\varphi_\alpha\hat\varphi_\beta\,d\hat x = a(x_\tau)\,J_\tau \begin{cases} 1/12, & \alpha = \beta, \\ 1/24, & \alpha \neq \beta, \end{cases}$$

а матрица M^τ является положительно определенной.

Аналогично вычисляются компоненты вектора сил элемента τ

$$F^\tau_\alpha = \int\limits_{\tau} f(x)\varphi_{i_\alpha}(x)\,dx.$$

Оба способа вычисления интеграла приводят к формуле

$$F^\tau_\alpha = (J_\tau\,/6)\,f(x_\tau), \quad \alpha = 1,2,3.$$

Для вычисления "граничной" матрицы масс и вектора сил (граничного ребра e с номерами вершин i_1 и i_2) с компонентами

$$h^e_{\alpha\beta} = \int\limits_{e} \sigma(x)\,\varphi_{i_\beta}\varphi_{i_\alpha}\,dx, \quad g^e_\alpha = \int\limits_{e} \mu(x)\,\varphi_{i_\alpha}(x)\,dx$$

применяются аналогичные приемы: либо считается, что $\sigma(x) = \sigma(x_e)$, $\mu(x) = \mu(x_e)$ и интегралы вычисляются точно, либо используется квадратурная формула центральных прямоугольников. Для g^e_α в

обоих случаях получается одна и та же формула $g^e_\alpha = |e|/2\,\mu(x_e)$, $\alpha = 1, 2$, а для $h^e_{\alpha\beta}$ — разные. В 1-ом случае

$$h^e_{\alpha\beta} = |e|\,\mu(x_e) \begin{cases} 1/3, & \alpha = \beta, \\ 1/6, & \alpha \neq \beta, \end{cases}$$

во втором случае $h^e_{\alpha\beta} = (|e|/4)\,\mu(x_e)$. Здесь x_e средняя точка грани (вычисляется как полусумма координат вершин), $|e|$ — длина e; мы также учли, что $\varphi_{i_\alpha}(x_e) = 1/2$.

2.2. Способы задания коэффициентов краевой задачи.

Матрицы жесткости элементов зависят то коэффициентов дифференциального уравнения (таких как c, b, a, f); прежде чем программировать сборку матрицы жесткости, необходимо решить в каком виде определять их в программе. Надо учесть, что в общем случае они заданы кусочно, своими выражениями в различных подобластях.[1)] Будем следовать решению, принятому в *pde toolbox* : функции могут быть одновременно (векторизованно) вычислены в точках (x_i, y_i), принадлежащих "подобластям" с метками sdl_i (x, y и sdl считаются векторами-строками).

Согласно этому принципу, будем считать, что функции могут быть заданы следующим образом:

1) константой;

2) текстовым выражением MatLab, записанным в терминах переменных-строк x, y, sdl для вычисления функции в точках (x, y) с метками подобластей sdl [2)];

3) последовательностью текстовых выражений MatLab (таких как выше), разделенных знаком " ! "; число текстовых выражений в последовательности должно быть равно числу подобластей в sdl;

4) именем MatLab-функции, определенным пользователем, с аргументами (x, y, sdl);

[1)] способ задания краевых условий и функций u_D, σ, μ мы обсудим далее

[2)] в *pde toolbox* предполагается, что функции зависят от x, y, sdl, u, ux, uy, t, что необходимо при решении нелинейных и нестационарных задач ($\nabla u = (ux, uy)$).

5) вектором строкой, представляющем значения функции в центрах тяжести элементов (только для P_1 элементов).

Поясним эти способы. Напомним, что область Ω, в которой решается задача, может состоять из ряда подобластей, в которых коэффициенты уравнения могут иметь свой вид; подобласти предполагаются пронумерованными и с этими номерами (sdl — метками подобластей) могут связывается свои функции. Так $sdl = t(4,:)$ для P_1 элементов, $sdl = t(7,:)$ — для P_2 элементов (t — матрица связности элементов) и каждый конечный элемент "знает" номер подобласти, которой он принадлежит.

Таким образом, например, функцию $c(x)$ можно задать следующими способами:

1) $c = 1$; (во всех подобластях c принимает значение равное 1);

2) во всех подобластях c принимает значение равное $x_1^2 + x_2$:

```
c='x.^2+y';     |     c=@(x,y,sdl) x.^2+y;
```

3) предполагается, что имеется 3 подобласти; в первой подобласти c постоянна и равна 1; во 2-ой — $c = x_1^2 + x_2$, в 3-ей — $c = \sin(x_1 + x_2)$:

```
c='1! x.^2+y! sin(x+y)';
```

Объемые строки удобно формировать следующим образом: создаем $c1 = '\ldots'$; $c2 = '\ldots'$; …, $cm = '\ldots'$ и определяем $c = [\,''''\,c1\,''!''\,c2\,''!''\,\ldots\,''!''\,cm\,''''\,]$.

4) предыдущая функция c может быть задана также следующей m-функцией:

```
function f=c3(x,y,sdl)
f=zeros(size(x));
% 1-st subdomain
I=find(sdl==1); f(I)=1;
% 2-nd subdomain
I=find(sdl==2); f(I)=x(I).^2+y(I);
% 3-d subdomain
I=find(sdl==3); f(I)=sin(x(I)+y(I));
```

Мы реализуем эти возможности при помощи функции calc:

```
function y=calc(x,y,sdl,f)
if isnumeric(f)
  y=f;
elseif pdeisfunc(f),
  y=feval(f,x,y,sdl);
elseif ischar(f)
  y=pdetexpd(x,y,sdl,[],[],[],[],f);
end
```

Здесь $pdeisfunc(f)$ и *pdetexpd* — функции *pde toolbox*; $pdeisfunc(f)$ возвращает 1, если f имя m-файла, *mex*- или *dll*-файла в текущем директории пользователя или в путях поиска функций в Matlab, а также, если f встроенная функция MatLab или указатель на функцию. Функция *pdetexpd* реализует вычисление строк, примеры которых приведены выше. Отметим, что если входной параметр f есть число или числовой вектор, то функция *calc* просто возвращает его.

2.3. Вклад элементов в систему МКЭ.

В разделе 2.1, стр. 45, мы получили все расчетные формулы для сборки матриц и векторов для P_1 элементов. Сведем в одну функцию вычисление глобальной матрицы жесткости K и вектора сил F, учитывающих вклад элементов. Начнем с невекторизованной версии функции.

```
function [K,F] = assemban(p,t,c,a,b1,b2,f)
% ASSEMBAN: Assembles area integral contributions in a PDE problem.
%           Nonvectorized version.
%           Stiffness matrix K associates wih the oparator
%               u --> -div(c grad u) + b.grad u+ au
%
%  The following call is also allowed:
%  K = ASSEMBAN(p,t,c,a,b1,b2)
%
np=size(p,2); nt=size(t,2);
Kt= zeros(nt,3,3);  % all local stiffness matrix
F = zeros(np,1);    % global force vector
for it=1:nt
  % mesh point indices
  i1=t(1,it); i2=t(2,it); i3=t(3,it);
  sdl=t(4,it); % subdomain labels

  % barycenter of the triangle
  x = (p(1,i1)+p(1,i2)+p(1,i3))/3;
  y = (p(2,i1)+p(2,i2)+p(2,i3))/3;

  % gradient of the triangle base functions, multiplied on J
```

```
  g1x=p(2,i2)-p(2,i3); g1y=p(1,i3)-p(1,i2);
  g2x=p(2,i3)-p(2,i1); g2y=p(1,i1)-p(1,i3);
  g3x=p(2,i1)-p(2,i2); g3y=p(1,i2)-p(1,i1);

  J=abs(g3y.*g2x-g3x.*g2y);    % J=2*area

  % evaluate c,b,a on triangles barycenter
  cx   = calc(x,y,sdl,c);
  ax   = calc(x,y,sdl,a);
  b1x  = calc(x,y,sdl,b1);
  b2x  = calc(x,y,sdl,b2);

  % diagonal and offdiagonal elements of mass matrix
  ao = (ax/24).*J;    ad  = 4*ao;  % 'exact' integration
  % ao = (ax/18).*J; ad  = 3*ao;  % quadrature rule

  % b contributions
  b1x=b1x/6; b2x=b2x/6;
  bg1=b1x.*g1x+b2x.*g1y;
  bg2=b1x.*g2x+b2x.*g2y;
  bg3=b1x.*g3x+b2x.*g3y;

  % coefficients of the stiffness matrix
  cx  = (0.5*cx)./J;
  k12= cx.*(g1x.*g2x+g1y.*g2y)+ao;
  k23= cx.*(g2x.*g3x+g2y.*g3y)+ao;
  k31= cx.*(g3x.*g1x+g3y.*g1y)+ao;

  Kt(it,:,:)=[ad-k31-k12+bg1   k12+bg2          k31+bg3
              k12+bg1          ad-k12-k23+bg2   k23+bg3
              k31+bg1          k23+bg2          ad-k23-k31+bg3];
  if nargout==2
    fx  = calc(x,y,sdl,f);
    I=[i1;i2;i3];
    F(I) = F(I)+  fx*J/6;
  end
end

K=sparse(np,np); % global stiffness matrix
for i=1:3, for j=1:3
    K=K+sparse(t(i,:),t(j,:),Kt(:,i,j),np,np);
end, end

if nargout==1, F=[]; end
```

Простой анализ этой функции показывает, что для векторизации достаточно убрать цикл по *it*, векторизовать команды помеченные знаком процента с цифрами 1, 2, а также рассылать элементы сразу, как только их вычислили. В результате получим следующую векторизованную версию функции.

```
function [K,F] = assemba(p,t,c,a,b1,b2,f)
% ASSEMBA: Assembles area integral contributions in a PDE problem.
%          Stiffness matrix K associates with the operator
%               u --> -div(c grad u) + b.grad u+ au
%
%  The following call is also allowed:
%  K = ASSEMBA(p,t,c,a,b1,b2)
%
np = size(p,2);
i1 = t(1,:); i2 = t(2,:); i3 = t(3,:);  % mesh point indices
sdl= t(4,:); % sub domain labels

% barycenter of the triangles
x = (p(1,i1)+p(1,i2)+p(1,i3))/3;
y = (p(2,i1)+p(2,i2)+p(2,i3))/3;

% gradient of the triangle base functions, multiplied on J
g1x=p(2,i2)-p(2,i3); g1y=p(1,i3)-p(1,i2); g2x=p(2,i3)-p(2,i1);
g2y=p(1,i1)-p(1,i3); g3x=p(2,i1)-p(2,i2); g3y=p(1,i2)-p(1,i1);

J=abs(g3y.*g2x-g3x.*g2y);    % J=2*area

% evaluate c,b,a on triangles barycenter
cx   = calc(x,y,sdl,c);
ax   = calc(x,y,sdl,a);
b1x  = calc(x,y,sdl,b1);
b2x  = calc(x,y,sdl,b2);

% diagonal and off diagonal elements of mass matrix
ao = (ax/24).*J;    ad  = 4*ao;  % 'exact' integration
% ao = (ax/18).*J; ad  = 3*ao;  % quadrature rule

% coefficients of the stiffness matrix
cx = (0.5*cx)./J;
k12= cx.*(g1x.*g2x+g1y.*g2y)+ao;
k23= cx.*(g2x.*g3x+g2y.*g3y)+ao;
k31= cx.*(g3x.*g1x+g3y.*g1y)+ao;

if all(b1x==0) && all(b2x==0) % symmetric problem
  K =   sparse(i1,i2,k12,np,np);
  K = K+sparse(i2,i3,k23,np,np);
  K = K+sparse(i3,i1,k31,np,np);

  K = K + K.';

  K = K+sparse(i1,i1,ad-k31-k12,np,np);
  K = K+sparse(i2,i2,ad-k12-k23,np,np);
  K = K+sparse(i3,i3,ad-k23-k31,np,np);
else
  % b contributions
  b1x=b1x/6; b2x=b2x/6;
  bg1=b1x.*g1x+b2x.*g1y;
  bg2=b1x.*g2x+b2x.*g2y;
  bg3=b1x.*g3x+b2x.*g3y;
```

```
  K =    sparse(i1,i2,k12+bg2,np,np);
  K = K+sparse(i2,i3,k23+bg3,np,np);
  K = K+sparse(i3,i1,k31+bg1,np,np);

  K = K+sparse(i2,i1,k12+bg1,np,np);
  K = K+sparse(i3,i2,k23+bg2,np,np);
  K = K+sparse(i1,i3,k31+bg3,np,np);

  K = K+sparse(i1,i1,ad-k31-k12+bg1,np,np);
  K = K+sparse(i2,i2,ad-k12-k23+bg2,np,np);
  K = K+sparse(i3,i3,ad-k23-k31+bg3,np,np);
end

if nargout==2
  fx = calc(x,y,sdl,f);
  fx = (fx/6).*J;
  F =     sparse(i1,1,fx,np,1);
  F = F + sparse(i2,1,fx,np,1);
  F = F + sparse(i3,1,fx,np,1);
else
  F=[];
end
```

Сравним эти две версии программы на примере. Рассмотрим область, состоящую из 3-х подобластей, изображенную на рис. 1 (см. §2, с. 19). Его геометрия определяется матрицей gtm. Замерим время выполнения на 3-х сетках, выполняя функцию

```
function mainTestAssemba
tassemba=[]; tassemban=[];  % assembling time
nt=[]; np=[];

g=gtm;  % geometry matrix

% PDE coefficients
c='1! x.^2+y! sin(x+y)'; % string
a=@(x,y,sdl) y.^2;       % anonymous function
b1='x+y!x-y!x.*y';       % string
b2=1;                    % double
f=@c3;                   % handle of the m.file
for hmax=[0.1 0.05 0.02]
  [p,~,t]=initmesh(g,'hmax',hmax);
  np=[np size(p,2)];  nt=[nt size(t,2)];
  tic; [K,F] = assemba(p,t,c,a,b1,b2,f);  tassemba=[tassemba toc];
  tic; [K,F] = assemban(p,t,c,a,b1,b2,f); tassemban=[tassemban toc];
end disp('          np          nt       tassemba     tassemban')
disp([np' nt' tassemba' tassemban' ])
```

В результате получим следующую таблицу

```
   np      nt    tassemba    tassemban
  697    1298    0.015625     3.2813
 2652    5112    0.078125     12.969
16459   32438      0.5625     82.063
```

Несмотря на не высокую точность замера времени выполнения по CPU, можно уверенно говорить о пользе векторизации при программировании в MatLab.

2.4. Учет краевых условий. Формирование системы МКЭ.

Способ задания краевых условий. Способ задания краевых условий, принятый в *pde toolbox* , тесно связан с графическим приложением *pdetool*, в котором можно определить как геометрию области, так и краевые условия. Это приложение позволяет решать не только скалярные уравнения, но и системы уравнений, что и определяет сложность задания краевых условий. Для нашей модельной задачи мы примем решение, которое мотивируется следующими соображениями: части границы Γ_0 и Γ_1 являются объединениями граничных сегментов (целиком), номера которых которых прописаны в файле или матрице, определяющей геометрию; с каждым таким сегментом связывается либо функция u_D (определяющее краевое условие Дирихле), либо пара функций (σ, μ) (определяющее краевое условие 3-го рода). В связи с этим будем определять условия на Γ_0 и Γ_1 раздельно.

Пусть Γ_0 не пусто и является объединением d граничных сегментов с номерами $(i_1, i_2, \ldots, i_d)$. На всех сегментах функция u_D может определяться одной формулой или d формулами (возможно разными для разных сегментов). Будем считать, что u_D задается так, как это описано в пунктах 1-4 секции 2.2, с. 49, для коэффициентов уравнения. Таким образом, информация об условиях Дирихле задается строкой

```
bsD=[i_1,i_2,\ldots,i_d],
```

а также строкой, постоянной или указателем на m-функцию:

```
uD='f1!f2!...! fd'; | uD='f'; |  uD=c; | uD=@f; | uD=@(x,y,sdl)...;
```

Здесь формула $f1$ соответствует сегменту i_1, $f2$ соответствует i_2 и т.д., $c = \text{const}$. Способ вполне ясный и удобный, но нужно помнить, что теперь *sdl* имеет “локальные” номера $1, 2, \ldots, d$.

Рассмотрим пример. Пусть имеется три граничных сегмента, с номерами $5, 2, 7$ на которых задано условие Дирихле. Тогда $bsD =$ [5 2 7]. Если на всех сегментах $u_D = x_1 \sin(x_1 + x_2)$ ($u_D = 2$), то $uD =' x. * \sin(x + y)'$ ($uD = 2$). Если на 5 и 7 граничных сегментах $u_D = 1$, а на 2-ом — $u_D = x + y$, то $uD =' 1\,!\,x + y\,!\,1'$. Эту функцию можно задать также следующим m-файлом:

```
function f=uD(x,y,sdl) f=zeros(size(x));
% 1-st segment (in local numeration)
I=find(sdl==1); f(I)=1;
% 2-nd segment
I=find(sdl==2); f(I)=x(I)+y(I);
% 3-d segment
I=find(sdl==3); f(I)=1;
```

Если условия Дирихле не заданы, то данные bsD и uD договоримся определять пустыми: $bsD = []$, $uD = []$; если же $\Gamma_0 = \Gamma$, то полагаем $bsD = inf$.

Совершенно аналогично задаются краевые условия 3-го рода; в этом случае, если на сегменте $\sigma = \mu = 0$, т.е. задано однородное условие Неймана, то такой сегмент можно не указывать в списке; если $\Gamma_1 = \Gamma$, то полагаем $bsN = inf$. Например,

```
bsN=[1     4    2    8];
sg=  'x ! y ! 1 ! 2';
mu=0;
```

Наконец, если $\Gamma_0 = \emptyset$, а на всей Γ_1 задано однородное условие Неймана, то полагаем $bsN = inf$, $sg = []$, $mu = []$.

Для удобства ссылок, эти данные (bsD, uD, bsN, sg, mu) нам будет удобно упаковать далее в структуру bc, с соответствующими полями.

Учет краевых условий. Подготовим данные, которые нам позволили бы легко сформировать систему алгебраических уравнений МКЭ $K_0 u_0 = F_0$, которая, напомним, имеет следующий вид (см. (1.8), с. 8):

$$\sum_{j \in i_n} a_{ij} u_j = \phi_i - \sum_{j \in i_d} a_{ij} u_{D\,j}, \quad i \in i_n,$$

где матрица $A = \{a_{ij}\}_{i,j=1}^{n_p} = K + H$, $\Phi = \{\phi_i\}_{i=1}^{n_p} = F + G$, i_d — множество номеров точек сетки, в которых задано условие Дирихле, $i_n = \{i_1, i_2, \ldots, i_n\}$ — множество номеров остальных точек. Выше

мы рассмотрели функцию *assemba* вычисления K и F. Необходима аналогичная функция вычисления H и G; в этой же функции естественно вычислить вектор столбец u_D размера n_p такой, что все его элементы равны нулю, кроме элементов с номерами $j \in i_d$, равными $u_{D\,j}$.

Матрица K_0 получается из матрицы A вычеркиванием его строк и столбцов с номерами из множества i_d. Чтобы реализовать эту операцию, удобно ввести матрицу N размера $n_p \times m$, $m = numel(i_n)$, которую удобнее определить через его транспонированную N^T равенством

$$N^T(u_1, u_2, \ldots, u_{n_p})^T = (u_{i_1}, u_{i_2}, \ldots, u_{i_n})^T.$$

Таким образом, матрица N^T реализует требуемую операцию вычеркивания элементов u с номерами из i_d. Легко проверить, что

```
N=sparse([i_1,i_2,\ldots,i_n],1:m,1,np,m);
```

Следовательно, если вычислить H, G, N и u_D, то K_0 и F_0 легко получить по формулам

$$K_0 = N^T(K + H)N, \quad F_0 = N^T((F + G) - (K + H)u_D).$$

Отметим, что после решения системы $K_0 u_0 = F_0$ формула $u = Nu_0 + u_D$ позволяет получить вектор узловых значений решения u_h схемы МКЭ (дополняя вектор u_0 граничными значениями u_D). Имея в виду расчетные формулы для H и G, полученные ранее, приходим к следующей функции.

```
function [N,H,uD,G]=assembe(bc,p,e)
% ASSEMBE: Assembles boundary conditions contributions in a PDE problem.
%
%  The following call is also allowed:
%  N=ASSEMBE(bc,p,e)
%   [N,H]=ASSEMBE(bc,p,e)
%   [N,H,uD]=ASSEMBE(bc,p,e)
np=size(p,2);

H=sparse(np,np);  G=sparse(np,1); N=speye(np,np);
uD=sparse(np,1);

if all(bc.bsN==inf) && (numel(bc.sg)==0) && (numel(bc.mu)==0)
  return   % on all boundary homogeneous Neumann b.c.
end
```

```
e=e(:,e(6,:)==0|e(7,:)==0);  % all boundary edges
ie=e(5,:);
[bsg,~]=find(sparse(ie,ie,1,np,np));  % indices of boundary segments
nbsg=numel(bsg);

% set local numeration of boundary segments
loc=zeros(1,nbsg);

% set mixed boundary conditions
if nargout>=2
  bsN=bc.bsN;
  if numel(bsN)>0
    if bsN==inf, bsN=bsg; end % all b.c. are mixed
    loc(bsN)= 1:numel(bsN);
    % find boudary edges with mixed b.c.
    eN=e([1 2 5],ismember(ie, bsN));

    i1=eN(1,:); i2=eN(2,:); % indices of start and end points
    x=0.5*(p(1,i1)+p(1,i2)); y=0.5*(p(2,i1)+p(2,i2));
    h=sqrt((p(1,i2)-p(1,i1)).^2+(p(2,i2)-p(2,i1)).^2); % edges length

    sdl=loc(eN(3,:));
    % evaluate sigma, mu on edges barycenter
    sx    = calc(x,y,sdl,bc.sg);
    mx    = calc(x,y,sdl,bc.mu);

    % diagonal and off diagonal elements of edge mass matrix
    so = (sx/6).*h;    sd  = 2*so;  % 'exact' integration
    %so = (sx/4).*h; sd  = so;      % quadrature rule

    H =      sparse(i1,i2,so,np,np);
    H = H + sparse(i2,i1,so,np,np);
    H = H + sparse(i1,i1,sd,np,np);
    H = H + sparse(i2,i2,sd,np,np);

    if nargout==4
      mx = calc(x,y,sdl,bc.mu);
      mx = (mx/2).*h;
      G =      sparse(i1,1,mx,np,1);
      G = G + sparse(i2,1,mx,np,1);
    end
  end
end

% set Dirichlet boundary conditions
bsD=bc.bsD;
if numel(bsD)>0
  if bsD==inf, bsD=bsg; end % all b.c. are Dirichlet
  loc(bsD)= 1:numel(bsD);
  if all(loc==0),
    disp('error. bsD+bsN~=number of boundary segments')
  end
  eD=e([1 2 5],ismember(ie, bsD));  % boudary edges with Dirichlet b.c.
```

```
  sdl=loc(eD(3,:));

  i1=eD(1,:); i2=eD(2,:); % indices of start and end points
  iD=[i1 i2];  % Dirichlet points indices (with copy)
  [id,~]=find(sparse(iD,iD,1,np,np));  % id=Dirichlet point indices

  iN=ones(1,np);   iN(id)=zeros(1,numel(id));
  iN=find(iN);     % indices of non-Dirichlet points in the domain
  niN=numel(iN);
  N=sparse(iN,1:niN,1,np,niN);

  if nargout>=3 % evaluate uD in Dirichlet points
    uD(i1)    = calc(p(1,i1),p(2,i1),sdl,bc.uD);
    uD(i2)    = calc(p(1,i2),p(2,i2),sdl,bc.uD);
  end
end
```

Здесь мы воспользовались MatLab функцией *ismember*. Она имеет несколько вариантов вызова. Команда $t = ismember(a, b)$ возвращает логический вектор такой же длины как и a, и такой, что $t_k = 1$, если $a_k \in b$ (является элементом b) и $t_k = 0$ — иначе.

Формирование и решение системы МКЭ. На основании приведенных выше функций можем легко написать различные полезные функции для формирования и решения системы алгебраических уравнений МКЭ.

Следующая функция является аналогом функции *assempde* в *pde toolbox*. Она позволяет как находить решение исходной задачи, так и формировать компоненты системы уравнений МКЭ. Функция легка для понимания и не требует дополнительных пояснений.

```
function [K,F,N,uD,H,G] = assembpde(bc,p,e,t,c,a,b1,b2,f)
% ASSEMBPDE: Assembles a PDE problem.
%
%  u=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,f)
%  [K0,F0,N,uD]=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,f)
%  [K,F,N,uD,H,G]=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,f)
%  [K0,M0,N]=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,d)
%
%  u=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,f) assembles and solves
%  the PDE problem Lu=-div(c*grad(u))+b.grad(u)+a*u=f
%  on a mesh described by p,e,t, with boundary conditions
%  given by bc. It eliminates the Dirichlet boundary conditions
%  from the system of linear equations when solving for u.
%  The solution u is represented as a column vector of solution
%  values at the corresponding node points from p.
%
%  [K0,F0,N,uD]=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,f) assembles the
%  PDE problem by eliminating the Dirichlet boundary conditions
```

```
%  from the FEM system. uN=K0\F0 returns the solution
%  on the non-Dirichlet points. The solution to the full PDE
%  problem can be obtained by the MATLAB command u=N*uN+uD.
%
%  [K,F,N,uD,H,G]=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,f) gives a split
%  representation of the PDE problem.
%
%  [K0,M0,N]=ASSEMBPDE(bc,p,e,t,c,a,b1,b2,d) assembles the
%  PDE eigenvalue problem Lu=lam du by eliminating the Dirichlet
%  boundary conditions. Eigenvalues lam and eigenvectors u=N*uN
%  can be obtained by solving eigenvalue problem K0 uN=lam M0 uN.
%
%  The geometry of the PDE problem is given by the mesh data p,e,t,
%  bc describes the boundary conditions of the PDE problem.
%
if nargout==6
  [K,F] = assemba(p,t,c,a,b1,b2,f);
  [N,H,uD,G]=assembe(bc,p,e);
elseif (nargout==4)||(nargout==1)
  [K,F] = assemba(p,t,c,a,b1,b2,f);
  [N,H,uD,G]=assembe(bc,p,e);
  if size(N,2)==size(p,2) % no Dirichlet boundary conditions
    K=K+H;                % K=K0
    F=F+G;                % F=F0
  else
    Nt=N.';
    K=K+H;
    F=Nt*((F+G)-K*uD);    % F=F0
    K=Nt*K*N;             % K=K0
  end
  if nargout==1
    uN=K\F;
    K=N*uN+uD;            % K=u
  end
elseif nargout==3
  K = assemba(p,t,c,a,b1,b2);
  F = assembam(p,t,f);    % F=M
  [N,H]=assembe(bc,p,e);
  if size(N,2)==size(p,2) % no Dirichlet boundary conditions
    K=K+H;                % K=K0
  else
    Nt=N.';
    K=Nt*(K+H)*N;         % K=K0
    F=Nt*F*N;             % F=M0
  end
else
  error('assembpde:nargout', 'Wrong number of output parameters.');
end
```

2.5. Решение модельной задачи.

Применим функцию *assembpde* для решения модельной задачи. Рассмотрим краевую задачу

$$-\operatorname{div}(c(x)\,\nabla u) + b(x)\cdot\nabla u + a(x)u = f(x), \quad x = (x_1, x_2) \in \Omega,$$
$$u(x) = u_D(x), \quad x \in \Gamma_0, \quad c(x)\,\nabla u\cdot\nu(x) + \sigma(x)u = \mu(x), \quad x \in \Gamma_1,$$

в единичном круге Ω с центром в начале координат. Геометрия Ω задается функцией *circleg*, которая находится в *pde toolbox*. На рис. 3 (слева) указано разбиение границы этой области на 4 сегмента и нумерация этих сегментов; будем считать, что сегменты 1 и 3 образуют Γ_0, а сегменты 2 и 4 — Γ_1.

Пусть коэффициенты уравнения имеют следующий вид:

$$c = 2 + x_1 + x_2, \quad a = x_1 + x_2, \quad b_1 = x_1, \quad b_2 = x_2,$$
$$f = -8 - 6(x_1 + x_2) + (2 + x_1 + x_2)(x_1^2 + x_2^2).$$

Функции определяющие краевые заданы следующим образом:

$$u_D = x_1^2 + x_2^2, \quad \sigma = \begin{cases} 0, & \text{на 2 сег.,} \\ 2, & \text{на 4 сег.,} \end{cases} \quad \mu = \begin{cases} 2(2 + x_1 + x_2)u_D, & \text{на 2 сег.,} \\ 2((2 + x_1 + x_2) + 1)u_D, & \text{на 4 сег..} \end{cases}$$

Непосредственной подстановкой в уравнения нетрудно убедиться, что функция $u = x_1^2 + x_2^2$ является решением задачи.

Решим эту задачу используя функцию *assembpde* на последовательности из 4-х сеток с $h = [0.5\ \ 0.1\ \ 0.05\ \ 0.02]$. Поскольку точное решение известно, вычислим погрешность решения

$$err = \max_{x\in\omega_h} |u(x) - u_h(x)|$$

и выведем относительную погрешность $errh2 = err/h^2$; Также построим график погрешности решения при $h = 0.1$ и замерим времена выполнения функций *assemba*, *assembe* и *assembpde*. Эти задачи решает функция

```
function maintestPDE
clear all; close all; clc

g='circleg'; np=[]; nt=[]; errh2=[]; err=[];
tassemba=[]; tbc=[]; tsol=[];
```

```
for h=[0.5 0.1 0.05 0.02]
  [p,e,t]=initmesh(g,'hmax',h);

  exu=@(x,y,sdl) x.^2+y.^2;   % exact solution

  x=p(1,:); y=p(2,:);
  u=exu(x,y,1)';

  c='2+x+y';   a='x+y';   f='-8-6*(x+y)+(2+x+y).*(x.^2+y.^2)';
  b1='x'; b2='y';

  bc.bsD=[1 3];   bc.uD='x.^2+y.^2!x.^2+y.^2';
  bc.bsN=[4 2];   bc.sg='2!0';
  bc.mu='2*(2+x+y).*sqrt(x.^2+y.^2)+2*(x.^2+y.^2)!2*(2+x+y).*sqrt(x.^2+y.^2)';

  tic
  [A,F] = assemba(p,t,c,a,b1,b2,f);
  tassemba=[tassemba toc];

  tic, [N,Me,ud,Ge]=assembe(bc,p,e);   tbc=[tbc toc];

  tic
  y= assembpde(bc,p,e,t,c,a,b1,b2,f);
  tsol=[tsol toc];

  if h==0.1
    figure
    pdesurf(p,t,u-y);
    xlabel('x_1')
    ylabel('x_2')
    zlabel('u-u_h')
    colormap('hsv')
  end

  np=[np size(p,2)];   % number of mesh points
  nt=[nt size(t,2)];   % number of finite elements
  h2=h^2;

  err  =[err norm(u-y,inf)];
  errh2=[errh2 norm(u-y,inf)/h2];
end
disp('np    tassemba      tassembe      tassembpde')
disp([np' tassemba' tbc'  tsol'])
err
errh2
```

В результате выполнения этой функции был получен левый рис. 3 и следующая таблица:

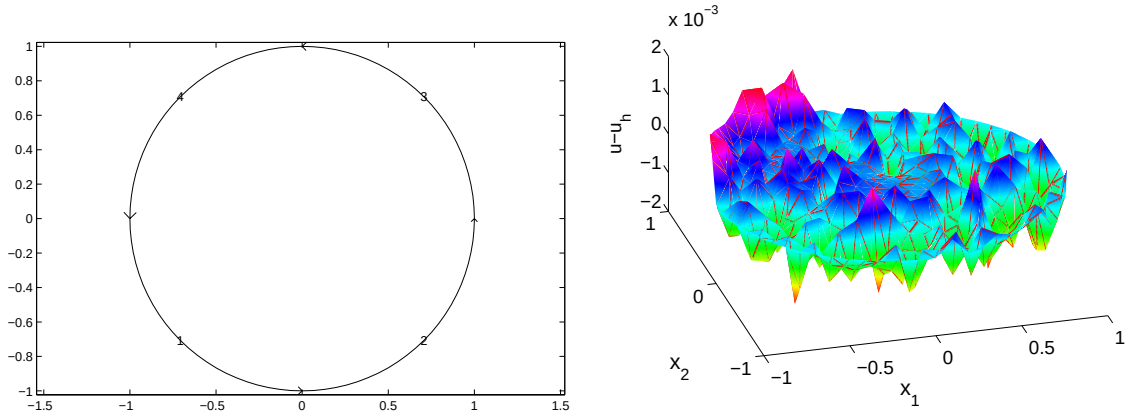

Рис. 3. Геометрия Ω (слева) и график погрешности решения (справа) на сетке с максимальным размером элементов $h = 0.1$ ($np = 544$).

np	tassemba	tassembe	tassembpde	
33	0.082	0.1439	0.0608	
544	0.022	0.0077	0.0462	
2173	0.065	0.0088	0.1455	
13693	0.436	0.0158	0.9947	
err =	0.065	0.0016	0.00064	8.9e−005
errh2 =	0.26	0.16	0.26	0.22

Полученные результаты свидетельствуют об успешной работе функции *assembpde*.

§ 3. Формирование системы МКЭ для P_2 элементов

Получим прежде всего расчетные формулы для базисных функций как для прямолинейных P_2 элементов, так и для криволинейных (изопараметрических).

3.1. Базисные функции прямолинейных P_2 элементов.

Расчетные формулы для P_2 элементов с прямолинейными сторонами получаются по той же схеме, как и для P_1 элементов: разница чисто количественная (больше узлов интерполяции, несколько сложнее вид базисных функций, больше узлов квадратурной формулы).

Пусть $(\hat{\tau}, \hat{\omega}, \hat{P}_2)$ — базисный конечный элемент, где $\hat{\tau}$ — единичный треугольник в декартовых координатах $\hat{x}$; $\hat{\omega}$ — множество узлов интерполяции $\hat{a}_i$, $i = 1, 2, \ldots, 6$ ($\hat{a}_4$, $\hat{a}_5$, $\hat{a}_6$ — середины сторон); $\hat{P}_2$ — множество полиномов суммарной 2 степени на $\hat{\tau}$ (см. левый рис. 4).

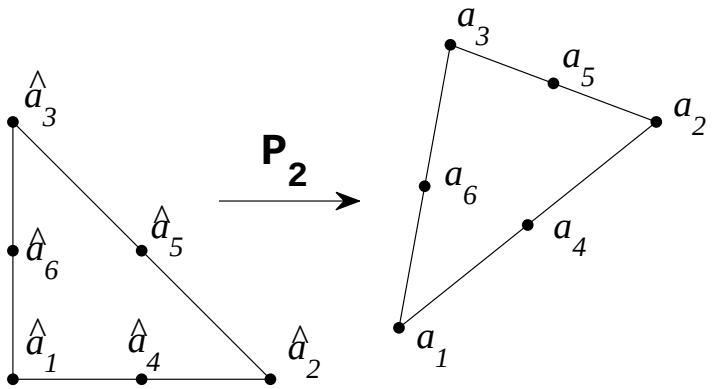

Рис. 4. Базисный P_2 элемент (слева) и P_2 элемент с прямолинейными сторонами.

Если $\hat{x}_3 = 1 - \hat{x}_1 - \hat{x}_2$, то легко проверить, что базис Лагранжа в $\hat{P}_2$ определяют функции

$$\hat{\varphi}_1 = \hat{x}_3(2\hat{x}_3 - 1), \quad \hat{\varphi}_2 = \hat{x}_1(2\hat{x}_1 - 1), \quad \hat{\varphi}_3 = \hat{x}_2(2\hat{x}_2 - 1/2) \qquad (3.7)$$
$$\hat{\varphi}_4 = 4\hat{x}_1\hat{x}_3, \quad \hat{\varphi}_5 = 4\hat{x}_1\hat{x}_2, \quad \hat{\varphi}_6 = 4\hat{x}_2\hat{x}_3,$$

а произвольная функция $\hat{p} \in \hat{P}_2$ допускает разложение

$$\hat{p}(\hat{x}) = \sum_{i=1}^{6} \hat{p}(\hat{a}_i)\hat{\varphi}_i(\hat{x}), \quad \hat{x} \in \hat{\tau}.$$

На рис. 4 справа изображен соответствующий этому базисному элементу произвольный элемент $(\tau, \omega_\tau, P_\tau)$. Точки a_i, $i = 1, \ldots, 6$, образуют множество ω_τ. Множество P_τ определяется следующим образом. Отображение (см. формулы (3.1)-(3.4))

$$x = B_\tau \hat{x} + a_1 \ : \ \hat{\tau} \to \tau,$$

задает преобразование $\hat{\tau}$ на τ, при этом точки $\hat{a}_i$ преобразуются в a_i, $i = 1, 2, \ldots, 6$. Обозначим обратное преобразование через $\hat{x} = \hat{x}_\tau(x) = B_\tau^{-1}(x - a_1)$. Тогда по определению

$$P_\tau = \{p : p(x) = \hat{p}(\hat{x}_\tau(x)),\ \hat{p} \in \hat{P}\} = \{p : p(x) = \sum_{i=1}^{6} c_i\hat{\varphi}_i(\hat{x}_\tau(x)),\ c_i \in R\},$$

причем $c_i = p(a_i)$. Легко видеть, что $P_\tau = P_2$. Таким образом, функции $\{\varphi_\alpha(x) = \hat{\varphi}_\alpha(\hat{x}_\tau(x))\}_{\alpha=1}^{6}$ образуют базис Лагранжа в P_τ.

3.2. Базисные функции изопараметрических P_2 элементов.

Криволинейные элементы, к которым относятся изопараметрические элементы, используются обычно в качестве приграничных элементов, с целью более точной аппроксимации как области Ω, так и ее границы Γ.

Схема определения изопараметрического P_2 элемента представлена на рис. 5, где слева указан базисный P_2 элемент $(\hat{\tau}, \hat{\omega}, \hat{P}_2)$, введенный выше (его базисные функции определяются формулами (3.7)). На рис. 5 справа изображен соответствующий этому базисному элементу изопараметрический элемент $(\tau, \omega_\tau, P_\tau)$. Точки a_i, $i = 1, \ldots, 6$, образуют множество ω_τ и, фактически, порождают как сам "элемент" τ, так и пространство функций P_τ. Осуществляется это следующим образом. Через узлы a_2, a_3, a_5 (они могут быть расположены на границе области Ω, например) можно провести единственную параболу; дуга этой параболы (между a_2 и a_3) и определяет часть границы τ (одну из 3-х его ребер). Важно отметить, что если эти узлы лежат на одной прямой, то парабола вырождается в прямую и ребро становится прямолинейным. Аналогично определяются остальные два ребра τ. Таким образом мы получаем "криволинейный" треугольник τ.[1)] Множество P_τ определяется следующим образом. Отображение

$$x = x_\tau(\hat{x}) = \sum_{i=1}^{6} a_i \hat{\varphi}_i(\hat{x}), \quad \hat{x} \in \hat{\tau}, \tag{3.8}$$

задает преобразование $\hat{\tau}$ на τ, при этом точки $\hat{a}_i$ преобразуются в a_i, $i = 1, 2, \ldots, 6$. Доказывается, что если диаметр τ достаточно мал, а точки a_4, a_5 и a_6 не "сильно" отклоняются от средних точек отрезков a_1a_2, a_2a_3 и a_3a_1 соответственно, то это преобразование взаимно однозначное и как само преобразование, так и обратное к нему — дифференцируемо. Обозначим обратное преобразование через $\hat{x} = \hat{x}_\tau(x)$.[2)] Тогда по определению

$$P_\tau = \{p : p(x) = \hat{p}(\hat{x}_\tau(x),\ \hat{p} \in \hat{P}\} = \{p : p(x) = \sum_{i=1}^{m_\tau} c_i \hat{\varphi}_i(\hat{x}_\tau(x)),\ c_i \in R\},$$

[1)] с одним, двумя или тремя прямолинейными ребрами (это зависит от расположения a_4, a_5, a_6).

[2)] его явное выражение нам далее не понадобится.

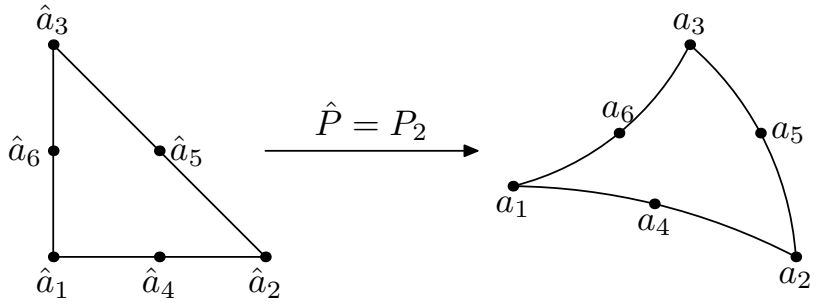

Рис. 5. Базисный P_2 элемент (слева) и произвольный изопараметрический P_2 элемент (справа).

причем $c_i = p(a_i)$. Таким образом, функции

$$\varphi_i(x) = \hat{\varphi}_i(\hat{x}_\tau(x)), \quad i = 1, 2, \ldots, 6,$$

образуют базис Лагранжа в P_τ.

Важно отметить, что если τ является треугольником с прямолинейными сторонами, то преобразование (3.8) совпадает с линейным преобразованием $x = B_\tau \hat{x} + a_1$ (это просто доказать) и, в этом случае, изопараметрический элемент совпадает с рассмотренным ранее P_2 элементом; в противном случае базисные функции и элементы P_τ не будут полиномами, но полиномами (2-ой степени) будут их сужения на каждую грань (относительно дуговой координаты грани).

3.3. Расчетные формулы для P_2 элементов.

Вклад элементов. Рассмотрим задачу вычисления матрицы жесткости K^τ для P_2 элемента τ, имеющую компоненты

$$k^\tau_{\alpha\beta} = \int_\tau \left(c\,\nabla\varphi_\beta \cdot \nabla\varphi_\alpha + b \cdot \nabla\varphi_\beta\varphi_\alpha + a\,\varphi_\beta\varphi_\alpha\right) dx. \tag{3.9}$$

Как мы могли видеть выше, элементы с прямолинейными сторонами и изопараметрические элементы различаются только видом преобразования $x = x(\hat{x}) : \hat{\tau} \to \tau$; поэтому неудивительно, что метод вычисления матрицы K^τ одинаков для обоих типов элементов.

Пусть $x = x_\tau(\hat{x}) = (x^\tau_1(\hat{x}), x^\tau_2(\hat{x}))^T$, определим матрицу $J_\tau(\hat{x})$,

транспонированную к матрице Якоби этого преобразования:

$$J_\tau(\hat{x}) = \begin{pmatrix} \dfrac{\partial x_1^\tau}{\partial \hat{x}_1} & \dfrac{\partial x_2^\tau}{\partial \hat{x}_1} \\ \dfrac{\partial x_1^\tau}{\partial \hat{x}_2} & \dfrac{\partial x_2^\tau}{\partial \hat{x}_2} \end{pmatrix} (\hat{x}),$$

а модуль его определителя (модуль якобиана преобразования $x = x(\hat{x})$) обозначим $|J_\tau(\hat{x})|$. Перейдем в интеграле (3.9) от элемента τ к элементу $\hat{\tau}$, учитывая, что

$$c(x) \to \hat{c}(\hat{x}) = c(x_\tau(\hat{x})), \quad \varphi_\alpha(x) \to \hat{\varphi}_\alpha(\hat{x}),$$

$$\nabla \to J_\tau^{-1}(\hat{x})\hat{\nabla}, \quad \hat{\nabla} = (\partial/\partial\hat{x}_1, \partial/\partial\hat{x}_2)^T, \quad dx = |J_\tau(\hat{x})| d\,\hat{x},$$

базисные функции $\hat{\varphi}_\alpha$, $\alpha = 1, \ldots, 6$, определяются формулами (3.7). Будем иметь

$$k_{\alpha\beta}^\tau = \int\limits_{\hat{\tau}} \big(\hat{c}(\hat{x}) J_\tau^{-1}(\hat{x})\hat{\nabla}\hat{\varphi}_\beta \cdot J_\tau^{-1}(\hat{x})\hat{\nabla}\hat{\varphi}_\alpha + \hat{b}(\hat{x}) \cdot J_\tau^{-1}(\hat{x})\hat{\nabla}\hat{\varphi}_\beta\hat{\varphi}_\alpha + \\ + \hat{a}(\hat{x})\hat{\varphi}_\beta\hat{\varphi}_\alpha\big)\, |J_\tau(\hat{x})|\, d\hat{x}.$$

Конечно, только в частных случаях этот интеграл может быть вычислен точно; поэтому используются квадратурные формулы для его приближенного вычисления. Если через $\hat{f}(\hat{x})$ обозначим подынтегральную функцию, то

$$k_{\alpha\beta}^\tau = \int\limits_{\hat{\tau}} \hat{f}(\hat{x})\, d\hat{x} \approx \sum_{i=1}^{q_\tau} \hat{c}_i \hat{f}(\hat{d}_i),$$

здесь коэффициенты $\hat{c}_i$ и узлы $\hat{d}_i$, $i = 1, \ldots, q_\tau$ определяют выбранную квадратурную формулу. Согласно теории МКЭ, достаточно использовать квадратуры, точные на полиномах из P_2 и имеющие положительные коэффициенты.

Различие в трудоемкости вычислений между элементами с прямолинейными и криволинейными сторонами заключается в том, что в 1-ом случае матрица $J_\tau^{-1} = B_\tau^{-T}$ не зависит от $\hat{x}$ и может быть вычислена лишь раз, тогда как во 2-м случае необходимо q_τ раз вычислять $J_\tau^{-1}(\hat{d}_i)$. Заметим также, что если ввести при $i = 1, \ldots, q_\tau$, следующие матрицы размера 2×6

$$\hat{D}_i = \{\hat{\nabla}\hat{\varphi}_\beta((\hat{d}_i))\}_{\beta=1}^6, \quad D_i = J_\tau^{-1}(\hat{b}_i)\hat{D}_i,$$

и векторы столбцы $g_i = \{\hat{\varphi}_\beta(\hat{d}_i)\}_{\beta=1}^6$, а также числа $C_i = \hat{c}_i\,|J_\tau(\hat{d}_i)|$, то $k^\tau_{\alpha\beta}$ можно записать в виде

$$k^\tau_{\alpha\beta} = \sum_{i=1}^{q_\tau} \hat{c}_i \hat{f}(\hat{d}_i) = \sum_{i=1}^{q_\tau} C_i \Big(\hat{c}(\hat{d}_i) D_i(\beta) \cdot D_i(\alpha) + \\ + \big(\hat{b}(\hat{d}_i) \cdot D_i(\beta) + \hat{a}(\hat{d}_i)\hat{g}_{i\beta}\big)\,\hat{g}_{i\alpha}\Big),$$

где $D_i(\alpha)$ — столбец D_i с номером α. Отсюда следует, что

$$K^\tau = \sum_{i=1}^{q_\tau} C_i \Big(\hat{c}(\hat{d}_i) D_i^T D_i + g_i \left(\hat{b}(\hat{d}_i) D_i + \hat{a}(\hat{d}_i)\hat{g}_i^T\right)\Big), \tag{3.10}$$

поскольку $x \cdot y = x^T y$, $xy^T = \{x_i y_j\}_{i,j=1}^n$, $x, y \in R^n$. Здесь $\hat{b} = (\hat{b}_1, \hat{b}_2)$ — вектор строка.

Аналогично вычисляется вектор сил конечного элемента. Имеем:

$$F^\tau_\alpha = \int\limits_\tau f(x)\varphi_\alpha(x)\,dx \approx \sum_{i=1}^{q_\tau} \hat{c}_i \big(\hat{f}(\hat{x})\hat{\varphi}_\alpha(\hat{x})\,|J_\tau(\hat{x})|\big)(\hat{d}_i),$$

$$F^\tau = \sum_{i=1}^{q_\tau} C_i\, \hat{f}(\hat{d}_i)\hat{g}_i. \tag{3.11}$$

Заметим, что для изопараметрического элемента (см. (3.8))

$$\hat{f}(\hat{d}_i) = f(X\hat{g}_i), \quad J_\tau(\hat{d}_i) = \hat{D}_i\, X^T, \quad X_{2\times 6} = (a_1, a_2, \ldots, a_6), \tag{3.12}$$

для любой функции f. Здесь столбцами X являются координаты узлов элемента τ.

Укажем две подходящие квадратурные формулы на $\hat{\tau}$. Первая из них имеет три узла ($q_\tau = 3$), является точной на полиномах суммарной второй степени ($\hat{P}_2$) и имеет следующие узлы (середины сторон $\hat{\tau}$) и коэффициенты

```
d=[0.5  0.5  0          c=[1  1  1]/6.
   0    0.5  0.5],
```

Другая квадратура точна на полиномах суммарной 4-ой степени ($\hat{P}_4$) с $q_\tau = 6$. Ее узлы и коэффициенты имеют следующий вид:

```
a1=0.09157621350977073;    a2=0.09157621350977073;
```

```
b1=0.4459484909159649;      b2=0.4459484909159649;
a3=1-a1-a2; b3=1-b1-b2;
d=[a1  a1  a3  b1  b1  b3
   a2  a3  a2  b2  b3  b2];

c1=0.1099517436553218/2; c2=0.2233815896780115/2;
c=[c1  c1  c1  c2  c2  c2].
```

Вклад ребер. Пусть, теперь, e — граничное ребро с вершинами a_{i_1}, a_{i_2}, и "средним" узлом a_{i_3}. Рассмотрим вычисление граничной матрицы масс $H^e = \{h^e_{\alpha\beta}\}$ и вектора сил $G^e = \{g^e_\alpha\}$ с компонентами

$$h^e_{\alpha\beta} = \int\limits_e \sigma(x)\,\varphi_{i_\beta}\varphi_{i_\alpha}\,dx, \quad g^e_\alpha = \int\limits_e \mu(x)\,\varphi_{i_\alpha}(x)\,dx, \quad \alpha,\beta = 1,2,3.$$

Пусть $\hat{e} = \{\hat{s} \in [0,1]\}$ есть базисное ребро с вершинами $\hat{a}_1 = 0$, $\hat{a}_2 = 1$, средней точкой $\hat{a}_3 = 0.5$, и базисными функциями

$$\hat{\varphi}_1(\hat{s}) = (1-2\hat{s})\,(1-\hat{s}), \quad \hat{\varphi}_2(\hat{s}) = \hat{s}\,(2\hat{s}-1), \quad \hat{\varphi}_3(\hat{s}) = 4\hat{s}\,(1-\hat{s}).$$

Отображение

$$x = x_e(\hat{s}) = a_{i_1}\hat{\varphi}_1(\hat{s}) + a_{i_2}\hat{\varphi}_2(\hat{s}) + a_{i_3}\hat{\varphi}_3(\hat{s})$$

задает взаимно-однозначное отображение $\hat{e} \to e$ ($\hat{a}_k \to a_{i_k}$, $k = 1,2,3$) и одновременно определяет параметризацию ребра e. Нетрудно проверить, что если $a_{i_3} = (a_{i_1} + a_{i_2})/2$, т.е. ребро является прямолинейным, то $x_e(\hat{s}) = a_{i_1}(1-\hat{s}) + a_{i_2}\hat{s}$. Пусть $x_e(\hat{s}) = (x_1(\hat{s}), x_2(\hat{s}))$. Перейдем в интегралах от e к $\hat{e}$. Получим:

$$h^e_{\alpha\beta} = \int\limits_0^1 \ell(\hat{s})\sigma(x_e(\hat{s}))\,\hat{\varphi}_\beta(\hat{s})\,\hat{\varphi}_\alpha(\hat{s})\,d\hat{s}, \quad g^e_\alpha = \int\limits_0^1 \ell(\hat{s})\mu(x_e(\hat{s}))\,\hat{\varphi}_\alpha(\hat{s})\,d\hat{s},$$

где $\ell(\hat{s}) = \Big(\big(x_1'(\hat{s})\big)^2 + \big(x_2'(\hat{s})\big)^2\Big)^{1/2}$. Для прямолинейного ребра, очевидно, $\ell(\hat{s}) = |a_{i_2} - a_{i_1}|$ — длина ребра. Для вычисления одномерных интегралов используем квадратурную формулу

$$\int\limits_0^1 \hat{f}(\hat{s})\,d\hat{s} \approx \sum_{i=1}^{q_e} \hat{c}_i \hat{f}(\hat{d}_i).$$

Как и выше введем векторы столбцы $\hat{g}_i = \{\hat{\varphi}_\beta(\hat{d}_i)\}_{\beta=1}^3$, $d\hat{g}_i = \{\hat{\varphi}'_\beta(\hat{d}_i)\}_{\beta=1}^3$, матрицу координат узлов $X_{2\times 3} = (a_{i_1}, a_{i_2}, a_{i_3})$. Тогда $x'_e(\hat{d}_i) = Xd\hat{g}_i$ и $\ell(\hat{d}_i) = |Xd\hat{g}_i|$ — длина вектора $Xd\hat{g}_i$, $C_i = \hat{c}_i|Xd\hat{g}_i|$,

$$H^e = \sum_{i=1}^{q_e} \big(C_i\,\sigma(X\hat{g}_i)\big)\ \hat{g}_i\hat{g}_i^T, \quad G^e = \sum_{i=1}^{q_e} \big(C_i\,\mu(X\hat{g}_i)\big)\ \hat{g}_i\ . \tag{3.13}$$

Отметим две подходящие квадратурные формулы на $[0,1]$, точные на полиномах третьей степени. Первая из них — квадратура Гаусса:

$$\hat{c}_1 = \hat{c}_2 = 0.5, \quad \hat{d}_1 = (1 - \sqrt{1/3})/2,\ \hat{d}_2 = (1 + \sqrt{1/3})/2, \quad q_e = 2,$$

а вторая — квадратура Симпсона (в этом случае H^e становится диагональной):

$$\hat{c}_1 = \hat{c}_3 = 1/6,\ c_2 = 4/6, \quad \hat{d}_1 = 0,\ \hat{d}_2 = 0.5,\ \hat{d}_3 = 1, \quad q_e = 3.$$

Литература

1. Сьярле Ф. Метод конечных элементов для эллиптических задач — М.: Мир, 1980. — 512 с.

2. Даутов Р.З., Карчевский М.М. Введение в теорию метода конечных элементов — Казань: Изд-во КГУ, 2004. — 239 с.