КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Р.З. ДАУТОВ

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДА КОНЕЧНЫХ ЭЛЕМЕНТОВ В MATLAB

Учебное пособие

Казань — 2014

УДК 519.3

Р.З. Даутов. Программная реализация метода конечных элементов в MATLAB. 106 с.

Излагаются основные этапы построения и программной реализации схем метода конечных элементов приближенного решения краевых задач для одномерных и двумерных линейных уравнений в частных производных второго порядка.

Пособие рассчитано на студентов старших курсов и аспирантов, специализирующихся в области численных методов решения краевых 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 рекомендации кафедры вычислительной математики Казанского (Приволжского) федерального университета

ПРЕДИСЛОВИЕ

В пособии излагаются основные этапы построения и программной реализации схем метода конечных элементов (МКЭ) приближенного решения краевых задач для одномерных и двумерных линейных уравнений второго порядка. Книга может рассматриваться как дополнение к учебнику [2].

Вкратце содержание пособия таково. Первая глава посвящена основным краевым задачам для линейных обыкновенных дифференциальных уравнений второго порядка. Рассматриваются такие вопросы как: классические и обобщенные постановки краевых задач; определение p и hp-схем МКЭ; алгоритмы формирования системы алгебраических уравнений МКЭ (§ 1), а также вопросы реализации этих схем на ЭВМ (§ 2). Изложение материала достаточно подробное и не предполагает первоначального знакомства читателя с методом конечных элементов. При этом мы считаем, что читатель знаком с основами теории обыкновенных дифференциальных уравнений и численных методов.

Во второй главе формулируются краевые задачи для линейных эллиптических уравнений уравнений второго порядка в двумерных областях, а также строятся схемы МКЭ для этих задач на основе лагранжевых элементов. Далее рассматриваются вопросы автоматизации решения дискретных задач. Вопросы описания плоских областей и построения треугольных сеток для них рассматривается в третьей главе. Четвертая глава посвящена алгоритмам и программам формирования и решения систем алгебраических уравнений МКЭ для двумерных задач, построенных на основе линейных элементов.

Предполагается, что читатель имеет некоторые навыки программирования в системе MatLab. Фрагменты MatLab-программ, приведенные в пособии, являются исполняемыми и вставлены в текст непосредственно из редактора MatLab. Для этого был использован стилевой файл *mcode*. Он часто некорректно воспринимает кириллический текст, поэтому комментарии к программам написаны на английском языке.

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@kpfu.ru*.

Оглавление

Глава 1

ОДНОМЕРНЫЕ СХЕМЫ МКЭ

В данной главе рассматриваются основные вопросы численной реализации метода конечных элементов (МКЭ) на примере краевой задачи для линейного обыкновенного дифференциального уравнения (ОДУ) второго порядка.

Далее через R ообозначается множество действительных чисел. Знак «:=» означает, что левая часть выражения определяется его правой частью (равенство по определению).

§ 1. Метод конечных элементов для линейных ОДУ второго порядка

На протяжении главы рассматривается линейное уравнение

$$Lu := -\big(pu' + ru\big)' + su' + qu = f. \tag{1.1}$$

Здесь p, r, s, q — коэффициенты оператора L и f — правая часть уравнения есть заданные функции x, $u = u(x)$ — решение уравнения — подлежит определению, $u' := du/dx$. Предполагается, что уравнение задано на ограниченном интервале (a, b) числовой оси.

1.1. Формулировка краевой задачи

Для однозначного определения решения уравнения (1.1) требуется сформулировать два дополнительных условия.

Краевой (или граничной) называется задача определения $u(x)$ в случае, когда одно дополнительное условие задано в точке $x = a$, другое — в точке $x = b$.[1)]

Дополнительные условия могут быть весьма разнообразными. К основным относят условия, имеющие вид $\alpha_c u(c) + \beta_c u'(c) = \gamma_c$, где c

[1)] Если оба дополнительных условия заданы в точке $x = a$ (или $x = b$), то соответствующая задача называется задачей Коши. Свойства решений этих двух задач, а также приближенные методы их решения, существенно различаются.

есть граничная точка (либо a, либо b); α_c, β_c, γ_c — заданные числа, $|\alpha_c| + |\beta_c| \neq 0$. Если $\beta_c = 0$, то краевое условие имеет вид $u(c) = g_c$. Оно означает, что решение задачи в точке c известно и равно g_c. Такое условие принято называть условием Дирихле или условием первого рода. Если $\alpha_c = 0$, то краевое условие имеет вид $u'(c) = g_c$. Оно называется условием Неймана или условием второго рода. Если же и α_c и β_c отличны от нуля, то краевое условие может быть записано в виде $u'(c) + \delta_c u(c) = g_c$, или, в виде $\pm\big(pu' + ru\big)(c) + \sigma_c u(c) = g_c$ при подходящих σ_c и g_c. Оно называется условием третьего рода.

Уравнение (1.1), дополненное краевыми условиями

$$u(a) = u_a, \quad u(b) = u_b, \tag{1.2}$$

называется первой краевой задачей. Уравнение (1.1) в совокупности с условиями

$$-\big(pu' + ru\big)(a) + \sigma_a u(a) = g_a, \quad \big(pu' + ru\big)(b) + \sigma_b u(b) = g_b, \tag{1.3}$$

называется третьей краевой задачей. Знак минус в выражении $-\big(pu' + ru\big)$ в уравнении (1.1) и в первом краевом условии (1.3) при рассмотрении общих уравнений можно было бы опустить. Его ставят именно так, как это делаем мы, поскольку именно в таком виде уравнения возникают в приложениях чаще всего. Далее в интегральном тождестве этот знак заменится на противоположный и его не придется писать.

Если в одной граничной точке задано условие первого рода, а в другом — третьего рода, тогда краевая задача называется смешанной. Далее будем для определенности считать, что краевые условия имеют вид

$$u(a) = u_a, \quad \big(pu' + ru\big)(b) + \sigma_b u(b) = g_b, \tag{1.4}$$

указывая, по ходу изложения, изменения, необходимые при других способах постановки граничных условий.

Задачу (1.1), (1.4) *назовем, для краткости, задачей* $\mathcal{P}$.

Обозначим через $C[a,b]$ и $C^k[a,b]$ множество непрерывных и множество k раз непрерывно дифференцируемых на отрезке $[a,b]$ функций соответственно. Аналогично определим $C^k(a,b)$, $C^k(a,b]$.

Функция $u \in C^2(a,b) \cap C^1(a,b] \cap C[a,b]$, *удовлетворяющая в каждой точке* $x \in (a,b)$ *уравнению* (1.1), *а также краевым условиям* (1.4), *называется классическим решением краевой задачи* $\mathcal{P}$.

Далее будем предполагать, что

$$p, r \in C^1[a,b], \quad s, q, f \in C[a,b], \quad p(x) \geqslant \text{const} > 0 \ \ \forall\, x \in [a,b]. \tag{1.5}$$

Эти условия обеспечивают корректность данного выше определения. Из последнего условия в (1.5) следует, что (1.1) является дифференциальным уравнением второго порядка всюду на (a,b). Предполагая также, что *классическое решение* $u(x)$ *задачи* $\mathcal{P}$ *существует, единственно и* $u \in C^2[a,b]$, сосредоточимся на методе конечных элементов его приближенного нахождения. Для этого нам понадобится понятие об интегральном тождестве, соответствующем задаче $\mathcal{P}$.

УПРАЖНЕНИЯ.

1. Найдите точное решения уравнения

$$u'' + 2u' - 8u = 0, \quad x \in (0,1), \tag{1.6}$$

при дополнительных условиях:

a) $u(0) = 1$, $u'(0) = 0$;

b) $u(0) = 1$, $u(1) = 0$.

Постройте графики найденных решений. Вы должны обнаружить, что решение уравнения (1.6) при условиях а) (задачи Коши), даже качественно отличается от решения уравнения (1.6) при условиях б) (краевой задачи): в случае а) решение экспоненциально возрастает, в случае b) — экспоненциально убывает.

Указание. Поскольку уравнение однородное и имеет постоянные коэффициенты, то можно найти характеристическое уравнение и общее решение. Действительно, если искать решение (1.6) в виде $u(x) = e^{\lambda x}$, то для определения λ получим уравнение $\lambda^2 + 2\lambda - 8 = 0$, которое имеет два различных решения λ_1 и λ_2. Следовательно, общее решение уравнения есть $u(x) = c_1\, e^{\lambda_1 x} + c_2\, e^{\lambda_2 x}$. Остается найти c_1 и c_2 из дополнительных условий.

2. В отличие от задачи Коши даже простейшие краевые задачи могут не иметь решения. Изучите краевую задачу

$$u'' + u = 0, \quad x \in (0,\pi), \quad u(0) = 1, \ \ u(\pi) = \mu, \tag{1.7}$$

где μ — произвольно заданное число (параметр задачи). При каких значениях μ решение задачи существует? Единственно ли оно в этом случае?

Указание. Используйте общее решение уравнения $u(x) = c_1 \sin(\pi x) + c_2 \cos(\pi x)$.

3. Знаки в уравнениях и краевых условиях играют существенную роль. Задача

$$-u'' + u = 0, \quad x \in (0,\pi), \quad u(0) = 1, \ \ u(\pi) = \mu,$$

отличается от краевой задачи (1.7) только знаком при u'', но оно имеет решение и притом единственное при любых $\mu \in R$. Найдите его.

1.2. Интегральное тождество

Пространство функций $L_2(a,b)$ определяется как множество измеримых по Лебегу функций v, имеющих конечный интеграл

$$\int_a^b v^2(x)\,dx\,,$$

понимаемый в смысле Лебега. Пространство функций $H^1(a,b)$ определяется следующим образом:

$$H^1(a,b) := \{v \in L_2(a,b) :\ v' \in L_2(a,b)\}.$$

Известно, что функции из $H^1(a,b)$ являются абсолютно непрерывными; в частности, всякая непрерывная на $[a,b]$, кусочно непрерывно дифференцируемая функция принадлежит $H^1(a,b)$.[1)] Кроме того, для функций $u, v \in H^1(a,b)$ справедлива формула интегрирования по частям:

$$\int_a^b u'(x)v(x)\,dx = -\int_a^b u(x)v'(x)\,dx + (uv)\big|_{x=a}^b.$$

Определим также следующие подмножества $H^1(a,b)$:

$$V := \{v \in H^1(a,b) : v(a) = u_a\}, \quad V^0 := \{v \in H^1(a,b) : v(a) = 0\}.$$

Ясно, что, если $u \in C^2[a,b]$ есть классическое решение задачи $\mathcal{P}$, то $u \in V$, $p\,u' + ru \in H^1(a,b)$.

Получим интегральное тождество для задачи $\mathcal{P}$. Для этого умножим почленно уравнение (1.1) на функцию $v \in V^0$, проинтегрируем полученное равенство по интервалу (a,b) и используем формулу интегрирования по частям для преобразования интеграла, содержащего производную функции $pu' + ru$. В результате получим:

$$\int_a^b (pu'v' + ruv' + su'v + quv)\,dx - (pu' + ru)v\big|_{x=a}^b = \int_a^b fv\,dx.$$

Преобразуем здесь внеинтегральное слагаемое, учитывая граничные условия для u и v. В результате получим

$$\int_a^b ((pu'+ru)\,v'+su'v+quv)\,dx+\sigma_b u(b)v(b) = \int_a^b fv\,dx+g_b v(b). \quad (1.8)$$

[1)] Для определения схемы МКЭ далее под $H^1(a,b)$ достаточно понимать множество непрерывных, кусочно непрерывно дифференцируемых на $[a,b]$ функций, понимая под v' «кусочную» производную v, а под интегралом — интеграл Римана.

Соотношение (1.8) *при* $u \in V$ *и произвольном* $v \in V^0$ *называется интегральным тождеством, соответствующим задаче* $\mathcal{P}$.

Интегральное тождество в некотором смысле эквивалентно задаче $\mathcal{P}$. Действительно, если u — решение задачи $\mathcal{P}$, то u удовлетворяет тождеству (1.8). Обратно, пусть u удовлетворяет тождеству (1.8) и $u \in C^2[a,b]$. В этом случае $Lu - f \in C[a,b]$. Применяя обратно формулу интегрирования по частям в (1.8), получим, что

$$\int_a^b (Lu-f)v\,dx+\big((pu'+ru)(b)+\sigma_b u(b)-g_b\big)v(b)=0 \quad \forall\, v \in V^0. \tag{1.9}$$

В силу произвольности функции v отсюда нетрудно получить, что $Lu = f$ на интервале (a,b), а также второе краевое условие (1.4). Это и означает, что u есть решение задачи $\mathcal{P}$, поскольку из условия $u \in V$ следует, что $u(a) = u_a$.

Нетрудно видеть, что тождество (1.8) сохраняет смысл при гораздо более слабых предположениях относительно p, r, s, q, f, чем (1.5).

Функция $u \in V$ *называется обобщенным (или слабым) решением задачи* $\mathcal{P}$, *если она удовлетворяет тождеству* (1.8) *для любой функции* $v \in V^0$.

Введем следующие обозначения:

$$a(u,v) := \int_a^b \big((pu'+ru)v'+(su'+qu)v\big)\,dx, \quad \langle u,v\rangle := \sigma_b u(b)v(b),$$

$$(u,v) := \int_a^b uv\,dx, \quad \langle v\rangle := g_b v(b).$$

Тогда интегральное тождество кратко запишется в виде:

$$u \in V: \quad a(u,v)+\langle u,v\rangle = (f,v)+\langle v\rangle \quad \forall\, v \in V^0. \tag{1.10}$$

Нетрудно видеть, что если u_i, v_j — произвольные числа, а φ_i, ψ_j произвольные функции из $H^1(a,b)$, то

$$a\big(\sum_{i=1}^n u_i\varphi_i, \sum_{j=1}^m v_j\psi_j\big) = \sum_{i=1}^n\sum_{j=1}^m a(\varphi_i,\psi_j)\,u_i v_j,$$

$$\big(f, \sum_{j=1}^m v_j\psi_j\big) = \sum_{j=1}^m (f,\psi_j)\,v_j.$$

Аналогичными свойствами обладают скобки $\langle\cdot,\cdot\rangle$ и $\langle\cdot\rangle$.[1)]

Интегральное тождество служит основой построения многих методов приближенного решения задачи $\mathcal{P}$. Например, на нем основан метод Галеркина, частным случаем которого является метод конечных элементов. Кратко его можно определить формулой:

$$\textit{МКЭ} = \textit{интегральное тождество} + \textit{кусочно-полиномиальная интерполяция функций}$$

УПРАЖНЕНИЯ.

1. Докажите, что из тождества (1.9) действительно следует, что справедливо уравнение $Lu(x) = f(x)$ во всех точках $x \in (a,b)$, а также краевое условие в точке $x = b$.

Указание. Выбором $v \in C^1[a,b]$, $v \equiv 0$ вне $[x_0 - h, x_0 + h] \subset (a,b)$, предельным переходом при $h \to 0$ получите равенство $(Lu - f)(x_0) = 0$ в произвольной точке x_0.

2. Сформулируйте интегральное тождество и определите множества функций V и V^0 для уравнения (1.1) при следующих краевых условиях:

a) в точке $x = a$ задано краевое условие третьего рода, в точке $x = b$ — первое;

b) при $x = a$ и при $x = b$ задано краевое условие третьего рода (см. условия (1.3)).

3. Дайте определение обобщенного решения краевых задач предыдущего пункта. Проверьте, что гладкое обобщенное решение является решением краевой задачи.

1.3. Пространство лагранжевых сплайнов

Рассмотрим два способа приближения функций: посредством единого алгебраического полинома и посредством кусочно-полиномиальной функции. Оба способа основаны на идее интерполяции.

1. Интерполяционный полином. Пусть $u = u(x)$ есть непрерывная на отрезке $[a,b]$ функция, $h = b - a$. Пусть далее на $[a,b]$ задана сетка узлов

$$a = x_1 < x_2 < \cdots < x_{m+1} = b,$$

и $u_i = u(x_i)$, $i = 1, \ldots, m+1$. Известно, что существует единственный алгебраический полином u_h степени не выше m, удовлетворяющий следующим условиям, называемым условиями интерполяции:

$$u_h(x_i) = u_i, \quad i = 1, \ldots, m+1. \tag{1.11}$$

[1)] Это означает, что $a(\cdot,\cdot)$ и $\langle\cdot,\cdot\rangle$ определяют билинейные функционалы на $H^1(a,b) \times H^1(a,b)$, а $(f,\cdot)$ и $\langle\cdot\rangle$ — линейные функционалы на $H^1(a,b)$.

Полином $u_h(x)$ рассматривается как приближение функции $u(x)$ на отрезке $[a,b]$. Существуют различные формы записи этого полинома.

Далее мы будем использовать формулу Лагранжа

$$u_h(x) = \sum_{i=1}^{m+1} u_i\,\varphi_i(x), \quad \varphi_i(x) = \prod_{j=1,\,j\neq i}^{m+1} \frac{x - x_j}{x_i - x_j}\,. \tag{1.12}$$

Полиномы $\{\varphi_i(x)\}_{i=1}^{m+1}$ называются базисными функциями Лагранжа. Легко видеть, что они удовлетворяют двум условиям

$$a)\ \varphi_i \in \mathcal{P}_m\,, \quad b)\ \varphi_i(x_j) = \delta_{ij},\ \ j = 1,\ldots,m+1,$$

и однозначно определяются ими. Здесь $\mathcal{P}_m$ означает множество алгебраических полиномов степени не выше m; δ_{ij} есть символ Кронекера: $\delta_{ij} = 1$ при $i = j$, $\delta_{ij} = 0$ при $i \neq j$. Прилагательное «базисные» объясняется тем, что система $\{\varphi_i(x)\}_{i=1}^{m+1}$ образует базис пространства $\mathcal{P}_m$, поскольку любой полином $p \in \mathcal{P}_m$ можно однозначно представить в виде

$$p(x) = \sum_{i=1}^{m+1} p_i\,\varphi_i(x), \quad p_i = p\,(x_i).$$

Это следует из единственности интерполяционного полинома. При $p(x) \equiv 1$ отсюда следует, что

$$\varphi_1(x) + \varphi_2(x) + \ldots + \varphi_{m+1}(x) = 1, \quad x \in [a,b]. \tag{1.13}$$

Легко видеть, что для вычисления $u_h(x)$ при заданном x по формуле (1.12) требуется выполнить $O(m^2)$ арифметических операций. Поэтому при больших значениях m предпочтительными оказываются другие формы записи интерполяционного полинома.

Отметим барицентрическую формулу, которая может быть получена преобразованием формулы Лагранжа следующим образом. Пусть

$$w(x) = \prod_{j=1}^{m+1}(x - x_j), \quad \beta_i = \Big(\prod_{j=1,\,j\neq i}^{m+1}(x_i - x_j)\Big)^{-1}.$$

Тогда

$$\varphi_i(x) = w(x)\,\frac{\beta_i}{x - x_i}, \quad u_h(x) = w(x)\,\sum_{i=1}^{m+1}\frac{\beta_i}{x - x_i}\,u_i. \tag{1.14}$$

Согласно формуле (1.13) имеем

$$1 = \sum_{i=1}^{m+1} \varphi_i(x) = w(x) \sum_{i=1}^{m+1} \frac{\beta_i}{x - x_i}.$$

Отсюда находится другое выражение для $w(x)$. Его использование в (1.14) приводит к искомой барицентрической форме записи интерполяционного полинома

$$u_h(x) = \Big(\sum_{i=1}^{m+1} \frac{\beta_i}{x - x_i} \Big)^{-1} \sum_{i=1}^{m+1} \frac{\beta_i}{x - x_i}\, u_i, \tag{1.15}$$

а также базисных функций Лагранжа

$$\varphi_i(x) = \Big(\sum_{i=1}^{m+1} \frac{\beta_i}{x - x_i} \Big)^{-1} \frac{\beta_i}{x - x_i}\ . \tag{1.16}$$

Числа β_i называются весами или коэффициентами барицентрической формулы. Легко видеть, что вычисление $u_h(x)$ при заданном x по формуле (1.15) требует $O(m)$ арифметических операций, если веса β_i предварительно вычислены.

Важно отметить, что *представления* (1.15), (1.16) *остаются справедливыми, если все веса* β_i *умножить на некоторую постоянную.* Это часто позволяет упростить формулы для них (например, когда x_i являются корнями или экстремумами классических ортогональных полиномов, сдвинутых на интервал $[a, b]$). Известно, что:

1) если узлы x_i распределены равномерно, т.е $x_{i+1} - x_i = h/m$, $i = 1, \ldots, m$, то

$$\beta_i = (-1)^i \binom{m}{i-1}, \quad \binom{m}{k} := \frac{m!}{(m-k)!\,k!}\ ; \tag{1.17}$$

2) если узлы x_i совпадают с экстремумами полинома Чебышева первого рода $T_m(x)$ на $[a, b]$, то

$$\beta_i = (-1)^i\, \gamma_i, \quad \gamma_1 = \gamma_{m+1} = 0.5, \ \ \gamma_i = 1,\ i = 2, \ldots, m. \tag{1.18}$$

3) если узлы x_i совпадают с экстремумами полинома Лежандра $L_m(x)$ на $[a, b]$ (узлами квадратуры Лобатто), то

$$\beta_i = (-1)^i\, \sqrt{c_i}, \quad i = 1, \ldots, m+1, \tag{1.19}$$

где c_i — веса квадратуры Лобатто с $m+1$ узлом (см. далее).

2. Кусочно-полиномиальная интерполяция. Определим на отрезке $[a,b]$ сетку узлов

$$a = a_1 < a_2 < \cdots < a_n = b.$$

Узлы сетки могут быть распределены как равномерно (расстояние между соседними узлами одинаково), так и неравномерно. Положим $h_i := a_{i+1} - a_i$, $h := \max\{h_i,\ i = 1, \ldots, n-1\}$.

На каждом отрезке $[a_i, a_{i+1}]$ выберем $m+1$ различных точек $a_i^{(j)}$,

$$a_i = a_i^{(1)} < a_i^{(2)} < \ldots < a_i^{(m+1)} = a_{i+1}.$$

Пусть теперь $u \in C[a,b]$ есть заданная функция. Определим ее приближение $u_h \in C[a,b]$ так, что: а) $u_h \in \mathcal{P}_m$ на каждом $[a_i, a_{i+1}]$; b) выполнены условия интерполяции

$$u_h(a_i^{(j)}) = u(a_i^{(j)}), \quad j = 1, \ldots, m+1, \quad i = 1, \ldots, n-1. \tag{1.20}$$

Ясно, что при $n = 2$ приходим к интерполяционному полиному степени m, рассмотренному в предыдущем пункте. При $n > 2$ функция u_h является интерполяционным полиномом степени m на каждом элементе $[a_i, a_{i+1}]$, а в целом на отрезке $[a,b]$ — лишь непрерывной функцией. Функцию u_h будем называть лагранжевым сплайном степени m, а множество всех таких функций обозначим через $S_h^m(a,b)$ и будем называть пространством лагранжевых сплайнов степени m.

1.4. Построение схемы МКЭ

Пусть как и выше на отрезке $[a,b]$ выбрана сетка узлов

$$a = a_1 < a_2 < \cdots < a_n = b.$$

Ячейки сетки $e_i := [a_i,\ a_{i+1}]$, $i = 1, \ldots, n-1$, будем называть конечными элементами. С этим разбиением свяжем пространство $S_h^m(a,b)$ непрерывных лагранжевых сплайнов степени m, $m \geqslant 1$,

$$S_h^m(a,b) := \{v \in C[a,b] : v \in \mathcal{P}_m \ \text{ на каждом } \ e_i,\ i = 1, \ldots, n-1\}.$$

Оно рассматривается как аппроксимация пространства $H^1(a,b)$. Отметим, что $S_h^m(a,b) \subset H^1(a,b)$, поскольку, если функция $v \in S_h^m(a,b)$,

то она непрерывна и кусочно непрерывно дифференцируема, т. е. $v \in H^1(a,b)$. Введем также аппроксимации множеств V и V^0 :

$$V_h := \{v \in S_h^m(a,b) : v(a) = u_a\}, \quad V_h^0 := \{v \in S_h^m(a,b) : v(a) = 0\}.$$

Функция $u_h \in V_h$, удовлетворяющая тождеству

$$\mathcal{P}_h : \qquad a(u_h, v) + \langle u_h, v\rangle = (f, v) + \langle v\rangle \quad \forall\, v \in V_h^0,$$

называется приближенным решением задачи $\mathcal{P}$ по методу конечных элементов. Сама задача $\mathcal{P}_h$ называется схемой МКЭ для задачи $\mathcal{P}$.

Как видим, схема МКЭ определяется весьма просто: она получается из (1.10) заменой V и V^0 их аппроксимациями. Именно такой способ определения приближенного решения принят в методе Галеркина. МКЭ от других вариантов методов Галеркина отличает то, что приближения в нем определяются на основе каких-либо сплайнов (лагранжевых, эрмитовых и т.д.). Схемы МКЭ на основе лагранжевых сплайнов наиболее просты и именно они чаще всего используются для уравнений второго порядка.

1.5. Система алгебраических уравнений МКЭ

Задачу $\mathcal{P}_h$ можно свести к системе линейных алгебраических уравнений, если выбрать базис в пространстве $S_h^m(a,b)$. Базис Лагранжа, принятый в МКЭ, определяется следующим образом.

На каждом элементе e_ℓ выберем $m+1$ различных точек (узлов) $a_\ell^{(j)}$ по некоторому правилу, одинаковому для всех элементов, так что

$$a_\ell = a_\ell^{(1)} < a_\ell^{(2)} < \ldots < a_\ell^{(m+1)} = a_{\ell+1}.$$

В МКЭ принято узлы на элементе определять так, что $a_\ell^{(j)} = a_\ell + h_\ell t_j$, $j = 1, \ldots, m+1$, для всех ℓ, где t_j — различные узлы на фиксированном отрезке $\Delta := [0,1]$. Такой способ определения узлов существенно сокращает вычисления.

Таким образом, на всем отрезке $[a,b]$ получим $N = (n-1)m + 1$ узлов, которые последовательно пронумеруем слева-направо, т. е. введем глобальную нумерацию узлов. Полученную сетку узлов обозначим через ω_h,

$$\omega_h = \{a = x_1 < x_2 < \cdots < x_N = b\},$$

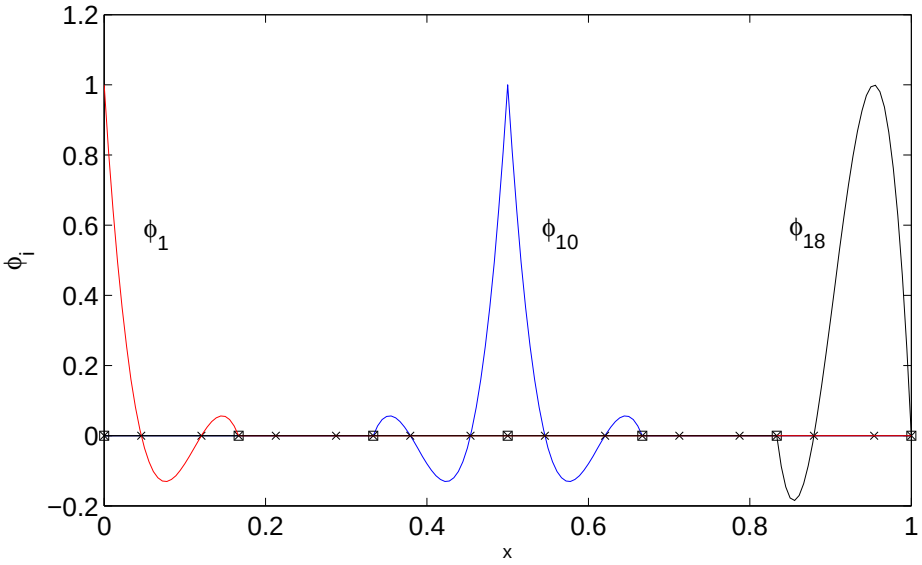

Рис. 1. Графики трех базисных функций φ_i с номерами $i = 1, 10, 18$ (на рис. вместо буквы φ использована ϕ) при $m = 3$, $n = 7$ (элементов — 6, узлов на элементе — 4). Крестиками обозначены точки сетки x_i, квадратиками — граничные точки элементов a_i. Каждая из функций равна нулю вне соответствующих элементов.

причем $x_k = a_i^{(j)}$ при $k = m(i-1) + j$.

Каждому узлу x_i поставим в соответствие функцию $\varphi_i \in S_h^m(a,b)$, называемую базисной, так, что $\varphi_i(x_j) = \delta_{ij}$, $j = 1, \ldots, N$. Базис Лагранжа обладает следующим важным свойством: если x_i не принадлежит некоторому конечному элементу, то φ_i тождественно равна нулю на этом элементе. Область, на которой φ_i отлична от нуля, есть либо один, либо два элемента (см. рис. 1).

Значение функции $v_h \in S_h^m(a,b)$ в узле $x_i \in \omega_h$ договоримся обозначать через v_i, а вектор $v = (v_i)_{i=1}^N$ называть вектором узловых параметров v_h. Нетрудно проверяется, что $v_h \in S_h^m(a,b)$ однозначно представляется в виде суммы

$$v_h(x) = \sum_{i=1}^{N} v_i \varphi_i(x), \quad x \in [a,b]. \tag{1.21}$$

Соответственно, произвольные функции $u_h \in V_h$ и $v_h \in V_h^0$ имеют

следующие однозначные разложения на всем $[a, b]$:

$$u_h(x) = u_a\varphi_1(x) + \sum_{j=2}^{N} u_j\,\varphi_j(x), \quad v_h(x) = \sum_{i=2}^{N} v_i\,\varphi_i(x). \tag{1.22}$$

Определим матрицу A и вектор Φ с компонентами

$$a_{ij} = a(\varphi_j, \varphi_i) + \langle\varphi_j, \varphi_i\rangle, \quad \phi_i = (f, \varphi_i) + \langle\varphi_i\rangle. \tag{1.23}$$

Подставляя в $\mathcal{P}_h$ вместо u_h его разложение (1.22), а вместо v_h — поочередно φ_i, $i = 2, 3, \ldots, N$, придем к искомой системе линейных алгебраических уравнений относительно u_j [2)]

$$\sum_{j=2}^{N} a_{ij}u_j = F_i, \quad i = 2, \ldots, N, \quad F_i = \phi_i - u_a\,a_{i1}\,. \tag{1.24}$$

Решая систему (1.24), найдем u_j, $j = 2, \ldots, N$. Тогда решение задачи $\mathcal{P}_h$ определится по первой формуле в (1.22).

Запишем систему (1.24) в матричном виде. Образуем вектор столбец неизвестных u_0, матрицу K_0 и вектор F_0 по правилу:

$$u_0 := \{u_i\}_{k=2}^{N}, \quad K_0 := \{a_{ij}\}_{i,j=2}^{N}, \quad F_0 := \{F_i\}_{i=2}^{N}.$$

Тогда система (1.24), очевидно, запишется в виде

$$K_0u_0 = F_0. \tag{1.25}$$

Формулы (1.23) дают некоторый способ вычисления элементов K_0 и F_0, но ими непосредственно не пользуются в практических вычислениях, т. к. известен более удобный и более экономичный метод, который мы и рассмотрим. *По традиции K_0 называют глобальной матрицей жесткости, F_0 — глобальным вектором сил, а систему уравнений* (1.25) — *системой МКЭ.*

[2)] Из этой системы следует тождество $\mathcal{P}_h$. Чтобы убедиться в этом достаточно умножить i-тое равенство в (1.24) на v_i, просуммировать по всем $i = 2, \ldots, N$, и учесть (1.22), (1.23).

УПРАЖНЕНИЯ.

Обратите внимание, что матрица K_0 и вектор F_0 определяются как уравнением, так и краевыми условиями. Можно сказать, что матричное уравнение (1.25) (сеточное уравнение) является аппроксимацией краевой задачи.

1. Дайте определение K_0 и F_0 в случае, когда решается первая краевая задача для уравнения (1.1).

2. Дайте определение K_0 и F_0 в случае краевой задачи с условием третьего рода в точке $x = a$ и условием Дирихле в точке $x = b$.

3. Дайте определение K_0 и F_0 в случае, когда решается третья краевая задача для уравнения (1.1).

Указание. Получите аналоги формул (1.22), (1.23) в каждом случае.

5. Докажите справедливость разложения (1.21).

1.6. Алгоритм формирования системы МКЭ

Система алгебраических уравнений (1.25) главным образом определяется матрицей A и вектором Φ. Поэтому сначала рассмотрим принятый в МКЭ способ их вычисления. Из (1.23) следует, что матрица A есть сумма двух квадратных матриц размерности $N \times N$: [1)]

$$A = K + H, \quad K := \big\{a(\varphi_j, \varphi_i)\big\}_{i,j=1}^{N}, \quad H := \big\{\langle \varphi_j, \varphi_i \rangle\big\}_{i,j=1}^{N}.$$

Аналогично,

$$\Phi = F + G, \quad F := \big\{(f, \varphi_i)\big\}_{i=1}^{N}, \quad G := \big\{\langle \varphi_i \rangle\big\}_{i=1}^{N}.$$

1. Вычисление K и F. По определению матрицы матрицы K имеем

$$K\, u \cdot v = a(u_h, v_h),$$

где $u, v \in R^N$ — векторы узловых параметров произвольно фиксированных функций $u_h, v_h \in S_h^m(a, b)$. Здесь и далее '$\cdot$' означает стандартное скалярное произведение векторов: $u \cdot v = \sum_{i=1}^{N} u_i\, v_i$. Положим

$$a_\ell(u, v) := \int_{a_\ell}^{a_{\ell+1}} \big((pu' + ru)v' + (su' + qu)v\big)\, dx,$$

$$(f, v)_\ell := \int_{a_\ell}^{a_{\ell+1}} f\, v\, dx.$$

[1)] отметим, что H — симметричная матрица; K является симметричной, если функция r тождественно равна функции s, т. к. в этом случае $a(u, v) = a(v, u)$, т. е. $a(\varphi_j, \varphi_i) = a(\varphi_i, \varphi_j)$.

Поскольку $[a,b] = \bigcup_{\ell=1}^{n-1}[a_\ell, a_{\ell+1}]$, то

$$K\,u\cdot v = \sum_{\ell=1}^{n-1} a_\ell(u_h, v_h). \tag{1.26}$$

Рассмотрим представление u_h и v_h на элементе e_ℓ с узлами интерполяции $x_{n_\ell+\beta}$, $n_\ell = m(l-1)$, $\beta = 1,\ldots,m+1$. Учтем, что $\varphi_i(x) = 0$, если x не принадлежит элементу, содержащему x_i. Поэтому на e_ℓ

$$u_h(x) = \sum_{i=1}^{N} u_i\varphi_i(x) = \sum_{\beta=1}^{m+1} u_{n_\ell+\beta}\varphi_{n_\ell+\beta}(x). \tag{1.27}$$

Аналогичное представление на e_ℓ имеет функция $v_h \in S_h^m(a,b)$:

$$v_h(x) = \sum_{\alpha=1}^{m+1} v_{n_\ell+\alpha}\varphi_{n_\ell+\alpha}(x).$$

Подставляя эти разложения в (1.26), получим

$$K\,u\cdot v = \sum_{\ell=1}^{n-1}\sum_{\alpha,\beta=1}^{m+1} k_{\alpha\beta}^{(\ell)}\,u_{n_\ell+\beta}\,v_{n_\ell+\alpha}. \tag{1.28}$$

Образовавшаяся здесь матрица $K^{(\ell)} = \{k_{\alpha\beta}^{(\ell)}\}_{\alpha,\beta=1}^{m+1}$, где

$$k_{\alpha\beta}^{(\ell)} = a_\ell(\varphi_{n_\ell+\beta}, \varphi_{n_\ell+\alpha}),$$

по традиции называется матрицей жесткости элемента e_ℓ (локальной матрицей жесткости). Преобразуем равенство (2.13). Введем в рассмотрение матрицу $\tilde{K}^{(\ell)}$ размера $N\times N$, состоящую из нулей, за исключением $(m+1)^2$ элементов, которые определим так, что

$$\tilde{K}^{(\ell)}_{n_\ell+\alpha,n_\ell+\beta} := k_{\alpha\beta}^{(\ell)}, \quad \alpha,\beta = 1,\ldots,m+1.$$

Тогда формула (2.13) преобразуется следующим образом:

$$K\,u\cdot v = \sum_{\ell=1}^{n-1}\sum_{\alpha,\beta=1}^{m+1} \tilde{K}^{(\ell)}_{n_\ell+\alpha,n_\ell+\beta}\,u_{n_\ell+\beta}\,v_{n_\ell+\alpha} = \sum_{\ell=1}^{n-1}\sum_{i,j=1}^{N} \tilde{K}_{ij}^{(\ell)}\,u_j\,v_i =$$

$$= \sum_{\ell=1}^{n-1}\tilde{K}^{(\ell)}u\cdot v = \Big(\sum_{\ell=1}^{n-1}\tilde{K}^{(\ell)}\Big)u\cdot v,$$

откуда, в силу произвольности векторов u и v, вытекает, что

$$K = \sum_{\ell=1}^{n-1} \tilde{K}^{(\ell)}.$$

Следовательно, глобальную матрицу жесткости можно получить суммированием по всем конечным элементам расширенных локальных матриц жесткости. Аналогичные соображения приводят к алгоритму вычисления F. Поскольку нули не имеет смысла суммировать, приходим к следующим алгоритмам.

Алгоритм сборки матрицы жесткости

- Положить $K = 0$ (K — матрица размера $N \times N$);
- Для каждого $\ell = 1, 2, \ldots, n-1$:
  - вычислить $K^{(\ell)}$ (размера $(m+1) \times (m+1)$);
  - для $\alpha, \beta = 1, \ldots, m+1$ суммировать:
    $$K_{n_\ell+\alpha, n_\ell+\beta} = K_{n_\ell+\alpha, n_\ell+\beta} + k^{(\ell)}_{\alpha\beta}.$$

Алгоритм сборки вектора сил

- Положить $F = 0$ (F — вектор столбец длины N);
- Для каждого $\ell = 1, 2, \ldots, n-1$:
  - вычислить $F^{(\ell)}$ (вектор длины $m+1$);
  - для $\alpha = 1, \ldots, m+1$ суммировать: $\quad F_{n_\ell+\alpha} = F_{n_\ell+\alpha} + F^{(\ell)}_\alpha$.

Здесь $F^{(\ell)}$ — вектор сил элемента с компонентами $F^{(\ell)}_\alpha = (f, \varphi_{n_\ell+\alpha})_\ell$.

2. Вычисление H и G. Матрица H и вектор G учитывают вклад краевых условий условий третьего рода в систему МКЭ. Имеем

$$H = \left\{ \langle \varphi_j, \varphi_i \rangle = \sigma_b \, \varphi_j(x_N) \, \varphi_i(x_N) \right\}_{i,j=1}^{N},$$
$$G = \left\{ \langle \varphi_i \rangle = g_b \, \varphi_i(x_N) \right\}_{i=1}^{N},$$

так как $b = x_N$. Поскольку $\varphi_i(x_j) = \delta_{ij}$, то ясно, что все элементы H и G равны нулю, кроме $H_{N,N} = \sigma_b$ и $G_N = g_b$.

На основе вышеизложенного, приходим к следующему алгоритму.

Алгоритм формирования системы МКЭ

- Вычисляем K и F, используя соответствующие алгоритмы сборки (вклады элементов в систему МКЭ).
- Учитываем краевые условия:
  - третьего рода: полагаем $K_{N,N} = K_{N,N} + \sigma_b,\ \ F_N = F_N + g_b$.
  - первого рода:

  a) полагаем $F = F - u_a\, K_1$, где K_1 — первый столбец K;

  b) вычеркивая первый столбец и первую строку K и первый элемент F, получаем соответственно матрицу K_0 и вектор F_0.

Видно, что основу этого алгоритма составляют алгоритмы сборки K и F, которые, в свою очередь, определяются алгоритмами вычисления локальной матрицы жесткости и сил элемента.

УПРАЖНЕНИЯ.

Обратите внимание, что построение системы МКЭ $K_0 u_0 = F_0$ (т. е. вычисление K_0 и F_0) складывается из двух шагов:

a) подсчитываются вклады от элементов (вычисляется матрица K и вектор F). Этот шаг не зависит от краевых условий, т. е. K, F определяются лишь уравнением;

b) K и F изменяются так, чтобы учесть краевые условия и получаются K_0 и F_0. Обращает на себя внимание также то, что условия третьего рода учитываются проще, чем условия первого рода. В связи с этим сформулируйте *алгоритм формирования системы МКЭ* в следующих случаях:

1. решается первая краевая задача.
2. решается третья краевая задача.
3. решается смешанная краевая задача и при $x = b$ задано условие первого рода.
4. Пусть задана разреженная (sparse) матрица K размера $N \times N$. Напишите MatLab код преобразования K в матрицу K_0 вычеркиванием: a) первого столбца и первой строки; a) строки и столбца с номером N; c) строк и столбцов с номерами 1 и N.

1.7. Алгоритм вычисления матрицы жесткости и вектора сил конечного элемента

Пусть ℓ номер конечного элемента, для которого необходимо вычислить матрицу $K^{(\ell)}$ и вектор $F^{(\ell)}$ с компонентами

$$k^{(\ell)}_{\alpha\beta} = a_\ell(\varphi_{n_\ell+\beta}, \varphi_{n_\ell+\alpha}), \quad F^{(\ell)}_\alpha = (f, \varphi_{n_\ell+\alpha})_\ell\,,$$

где $\alpha, \beta = 1, \ldots, m+1$. Эти числа представляют собой интегралы по интервалу $(a_\ell, a_{\ell+1})$, подынтегральные выражения которых зависят от коэффициентов ОДУ, функции f и, что существенно, от сужений базисных функций $\varphi_{n_\ell+\alpha}$ на элемент e_ℓ, $n_\ell = m(\ell-1)$. Базисные функции однозначно определяются следующими условиями:

1) $\varphi_{n_\ell+\alpha} \in \mathcal{P}_m$; 2) $\varphi_{n_\ell+\alpha}(x_{n_\ell+\beta}) = \delta_{\alpha\beta}$.

и в форме Лагранжа имеют вид:

$$\varphi_{n_\ell+\alpha}(x) = \prod_{j=1,\, j\neq\alpha}^{m+1} \frac{x - x_{n_\ell+j}}{x_{n_\ell+\alpha} - x_{n_\ell+j}}\,, \quad x \in [a_\ell, a_{\ell+1}]. \tag{1.29}$$

Используя их можно вычислить $k_{\alpha\beta}^{(\ell)}$ и $F_\alpha^{(\ell)}$. Например, имеем

$$F_\alpha^{(\ell)} = \int_{a_\ell}^{a_{\ell+1}} f(x)\, \varphi_{n_\ell+\alpha}(x)\, dx. \tag{1.30}$$

Ясно, что такие интегралы могут быть вычислены точно только для функций f простого вида. Это касается и чисел $k_{\alpha\beta}^{(\ell)}$. В силу этого для вычисления интегралов используются квадратурные формулы. Рассмотрим как они применяются.

Сделаем в интеграле (1.30) замену переменных, переходя от элемента $e_\ell := [a_\ell, a_{\ell+1}]$ к «элементу» с фиксированными размерами $\Delta := [0,1]$ (называемому базисным элементом) согласно линейной замене переменных

$$x = a_\ell + h_\ell\, t, \quad t \in [0,1], \quad a_\ell = x_{n_\ell+1}.$$

При таком переходе независимо от номера ℓ узлы $x_{n_\ell+j}$ отобразятся в некоторые узлы t_j, $0 = t_1 < t_2 < \ldots < t_{m+1} = 1$,. Поэтому

$$x - x_{n_\ell+j} = h_\ell\,(t - t_j), \quad x_{n_\ell+\alpha} - x_{n_\ell+j} = h_\ell\,(t_\alpha - t_j).$$

Следовательно, при замене переменных базисная функция $\varphi_{n_\ell+\alpha}(x)$ перейдет в $\hat{\varphi}_\alpha(t)$, так что

$$\varphi_{n_\ell+\alpha}(x) = \hat{\varphi}_\alpha((x - a_\ell)/h_\ell), \quad \hat{\varphi}_\alpha(t) = \prod_{j=1\, j\neq\alpha}^{m+1} \frac{t - t_j}{t_\alpha - t_j}\,. \tag{1.31}$$

Используя обозначение $\hat{f}(t) := f(a_\ell + h_\ell t)$, после замены переменных в (1.30) получим

$$F_\alpha^{(\ell)} = h_\ell \int_0^1 \hat{f}(t)\,\hat{\varphi}_\alpha(t)\,dt.$$

Для приближенного вычисления интеграла используем какую-либо квадратурную формулу (квадратуру)

$$\int_0^1 \phi(t)\,dt \simeq \sum_{k=1}^{M} c_k\,\phi(d_k),$$

где c_k, d_k — коэффициенты (веса) и узлы квадратуры соответственно. Ее использование приводит к приближенной формуле

$$F_\alpha^{(\ell)} \simeq \sum_{k=1}^{M} h_\ell\,c_k\,\hat{f}(d_k)\,\hat{\varphi}_\alpha(d_k), \quad \alpha = 1,\ldots,m+1. \tag{1.32}$$

Аналогично вычисляются $k_{\alpha\beta}^{(\ell)}$ с использованием замены переменных и квадратурной формулы. Поскольку при замене переменных справедливы равенства $u(x) = \hat{u}(t)$, $u'(x) = \hat{u}'(t)/h_\ell$, $dx = h_\ell\,dt$, то

$$a_\ell(u,v) = \int_{a_\ell}^{a_{\ell+1}} (pu'v' + ruv' + su'v + quv)(x)\,dx = \\ = \int_0^1 (h_\ell^{-1}\hat{p}\,\hat{u}'\,\hat{v}' + \hat{r}\,\hat{u}\,\hat{v}' + \hat{s}\,\hat{u}'\,\hat{v} + h_\ell\,\hat{q}\,\hat{u}\,\hat{v})(t)\,dt.$$

Поэтому

$$k_{\alpha\beta}^{(\ell)} \simeq \sum_{k=1}^{M} c_k\,(h_\ell^{-1}\hat{p}\,\hat{\varphi}'_\beta\,\hat{\varphi}'_\alpha + \hat{r}\,\hat{\varphi}_\beta\,\hat{\varphi}'_\alpha + \hat{s}\,\hat{\varphi}'_\beta\,\hat{\varphi}_\alpha + h_\ell\,\hat{q}\,\hat{\varphi}_\beta\,\hat{\varphi}_\alpha)(d_k). \tag{1.33}$$

Формулы (1.31), (1.32) и (1.33) определяют искомые формулы приближенного вычисления элементов матрицы $K^{(\ell)}$ и вектора $F^{(\ell)}$.

Использование квадратурных формул в алгоритмах сборки вместо исходной системы МКЭ $K_0 u_0 = F_0$ приводит к другой, близкой ей системе $\widetilde{K}_0\widetilde{u}_0 = \widetilde{F}_0$. Функцию $\widetilde{u}_h \in V^h$, определяемую по $\widetilde{u}_0$, называют приближенным решением исходной задачи по методу МКЭ с численным интегрированием, чтобы отличать ее от u_h. Интуитивно ясно, что отличие $\widetilde{u}_h$ от u_h тем меньше, чем точнее квадратурная

формула. В теории МКЭ доказывается, что это именно так и далее мы отметим условия на квадратуру, которые должны быть соблюдены. Далее под схемой МКЭ мы будем понимать именно схему с численным интегрированием.

Запишем формулы (1.32) и (1.33) в матричном виде. Для этого введем матрицы I и D размера $M \times (m+1)$ с компонентами

$$I_{k,\alpha} = \hat{\varphi}_\alpha(d_k), \quad D_{k,\alpha} = \hat{\varphi}'_\alpha(d_k). \tag{1.34}$$

Если определить вектор $f = h_\ell(c_1 \hat{f}(d_1), \ldots, c_M \hat{f}(d_M))^T$, то из (1.32) следует, что $F^{(\ell)} = I^T f$, где I^T есть матрица транспонированная к I. Аналогично можно записать $K^{(\ell)}$. Например, ее составляющая с элементами

$$E_{\alpha\beta} = \sum_{k=1}^{M} c_k \hat{r}(d_k)\, \hat{\varphi}_\beta(d_k)\, \hat{\varphi}'_\alpha(d_k) = \sum_{k=1}^{M} D^T_{\alpha,k}\, c_k \hat{r}(d_k)\, I_{k,\beta},$$
$$D^T_{\alpha,k} = D_{k,\alpha} = \hat{\varphi}'_\alpha(d_k),$$

записывается в виде $E = D^T R I$, где $R = \operatorname{diag}(c_1 \hat{r}(d_1), \ldots, c_M \hat{r}(d_M))$ есть диагональная матрица.

Матрицы I и D будем называть матрицами интерполирования и дифференцирования соответственно. Поясним эти названия. Приближенное решение на произвольном элементе e_ℓ представляется формулой (см. (1.27), (1.31))

$$u_h(x) = \sum_{\alpha=1}^{m+1} u_{n_\ell+\alpha} \varphi_{n_\ell+\alpha}(x) = \sum_{\alpha=1}^{m+1} \hat{\varphi}_\alpha\Big(\frac{x - a_\ell}{h_\ell}\Big)\, u_{n_\ell+\alpha}. \tag{1.35}$$

Пусть $\chi = a_\ell + h_\ell\, d$ — соответствующее d множество точек на e_ℓ (например, множество узлов квадратуры на e_ℓ), $u^{(\ell)} = \{u_{n_\ell+\alpha}\}_{\alpha=1}^{m+1}$ есть вектор столбец узловых параметров u_h на элементе e_ℓ. Тогда

$$u_h(\chi_k) = \sum_{\alpha=1}^{m+1} \hat{\varphi}_\alpha(d_k)\, u_{n_\ell+\alpha} = \sum_{\alpha=1}^{m+1} I_{k,\alpha}\, u_{n_\ell+\alpha} = (I\, u^{(\ell)})_k.$$

Следовательно, $u_h(\chi) = I u^{(\ell)}$. Таким образом, чтобы вычислить интерполяционный полином u_h в точках χ, достаточно умножить матрицу I на вектор $u^{(\ell)}$. Рассуждая аналогично получаем равенство

$u'_h(\chi) = h_\ell^{-1}\, D\, u^{(\ell)}$. Полученные формулы и объясняют названия этих матриц. Далее мы встретимся с матрицами I и D, определенными согласно (1.34), по произвольному множеству d. Они будут полезны, например, при построении графиков функций u_h и u'_h.

Суммируя сказанное, приходим к следующему алгоритму.

Алгоритм вычисления $K^{(\ell)}$ и $F^{(\ell)}$

- Исходные данные:
  - $a = (a_i)_{i=1}^{n}$ — вектор, определяющий разбиение $[a, b]$ на элементы;
  - $t = (t_i)_{i=1}^{m+1}$ — вектор узлов на базисном элементе $\Delta = [0, 1]$;
  - c, d — векторы длины M — веса и узлы квадратуры на Δ;
  - I, D — матрицы $M \times (m+1)$; $I_{k,\alpha} = \hat\varphi_\alpha(d_k)$, $\quad D_{k,\alpha} = \hat\varphi'_\alpha(d_k)$;
  - функции p, r, s, q, f переменной $x \in [a, b]$, определяющие ОДУ.
- Для заданного $\ell = 1, \ldots, n-1$:
  - вычисляем $x = a_\ell + h_\ell\, d$ — вектор узлов квадратуры на элементе e_ℓ (длины M), $h_\ell = a_{\ell+1} - a_\ell$;
  - вычисляем диагональные матрицы

    $P = h_\ell^{-1}\ \mathrm{diag}\,(c_1\, p(x_1), \ldots, c_M\, p(x_M))$,

    $R = \mathrm{diag}\,(c_1\, r(x_1), \ldots, c_M\, r(x_M))$,

    $S = \mathrm{diag}\,(c_1\, s(x_1), \ldots, c_M\, s(x_M))$,

    $Q = h_\ell\ \mathrm{diag}\,(c_1\, q(x_1), \ldots, c_M\, q(x_M))$,

    и вектор столбец $f = h_\ell\,(c_1\, f(x_1), \ldots, c_M\, f(x_M))^T$.
  - вычисляем: $F^{(\ell)} = I^T f$,

    $K^{(\ell)} = \big(D^T P + I^T S\big)\, D + \big(D^T R + I^T\, Q\big)\, I$.

УПРАЖНЕНИЯ.

1. Проверьте, что базисная функция (1.29) действительно удовлетворяет условиям 1) и 2), сформулированным чуть выше (1.29).

2. Проверьте правильность записи $K^{(\ell)}$ в матричном виде в приведенном выше алгоритме. Правильно ли расставлены знаки транспонирования матриц?

3. Какими способами можно определить в MatLab диагональную матрицу, если известны значения его диагональных элементов в виде a) вектора-строки; b) вектора-столбца?

§ 2. Программирование МКЭ решения основных краевых задач для линейных ОДУ второго порядка

Составим программу в среде программирования MatLab, которая позволила бы решить любую из краевых задач, указанных в первом параграфе, используя описанный выше метод конечных элементов. Напишем также функции, позволяющие, в частности: вычислить решение схемы МКЭ и его первую производную в произвольных точках $[a, b]$ и построить их графики, а также измерить погрешность найденного решения в нормах пространств $C(\omega_h)$, $C([a, b])$, $L_2(a, b)$ и $H_0^1(a, b)$ в случае, когда точное решение задачи известно.

Для этого необходимо, прежде всего, определить исходные данные программы, которые можно разделить на a) данные о решаемой задаче; b) данные метода конечных элементов.

2.1. Данные о решаемой задаче

Конкретная краевая задача определяется:

1) числами a, b, задающими область определения решения;
2) функциями p, r, s, q, f, определяющими ОДУ;
3) типом и данными краевых условий в точках a и b.

Если с пунктом 1) все понятно, то касательно пунктов 2) и 3) необходимо решить, в каком виде возможно (и удобно для пользователя) задать эти данные.

2) Как это принято в MatLab, будем считать, что функции всегда определяются векторизованно (т. е. при $x = [x_1, \ldots, x_n]$ командой

$y = f(x)$ вычисляется вектор $y = [f(x_1), \ldots, f(x_n)]$). Будем считать также, что функции могут быть заданы одним из следующих двух способов (на примере функции $r(x) = x\sin(x)$):

a) в виде неименованной (строчной) функции:
$r = @(x)\ x.*\sin(x)$. В этом случае r является указателем на функцию.

b) в виде m-функции:

```
function y=r(x)
y=x.*sin(x);
```

Способ a) удобен для простых функций, способ b) — для сложных.

3) Задание краевых условий. Краевые условия в каждой граничной точке могут быть двух типов: первого или третьего рода. Условие первого рода определяется одним числом u_a или u_b; условие третьего рода определяется двумя числами σ_a, g_a или σ_b, g_b. Ясно, что эту информацию можно упаковать в 2×3-матрицу bc. Будем считать, что ее первая строка $bc(1,:)$ определяет краевое условие в точке a, а вторая строка $bc(2,:)$ определяет краевое условие в точке b, и

$$\text{либо} \quad bc(1,:) = [1,\ 0,\ u_a], \qquad \text{либо} \quad bc(1,:) = [3,\ \sigma_a,\ g_a].$$

Аналогично, либо $bc(2,:) = [1,\ 0,\ u_b]$, либо $bc(2,:) = [3,\ \sigma_b,\ g_b]$.

2.2. Данные МКЭ

Прежде всего, конкретную схему МКЭ определяют три числа: m (степень полинома на элементе), n ($n-1$ — число конечных элементов), и M (число узлов квадратурной формулы на базисном элементе).

С этими числами напрямую связаны четыре массива (вектора) a, t, c, d. Эти данные влияют как на свойства матрицы системы МКЭ, так и на точность приближенного решения u_h.

Приведем соображения, которые полезно иметь в виду при задании этих параметров МКЭ. Напомним, что они используются в алгоритме вычисления локальных матриц жесткости и сил (см. с. 25).

(D_1) Числа n и m однозначно определяют структуру заполнения глобальной матрицы жесткости K ненулевыми элементами (как говорят, ее портрет). Если n намного больше m, то K содержит «очень

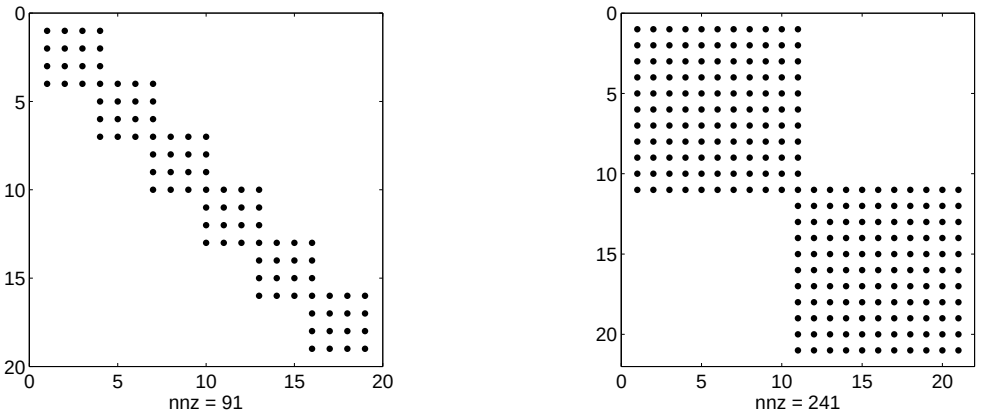

Рис. 2. Примеры портретов глобальной матрицы жесткости K. Левый рис. соответствует значениям $m = 3$, $n = 7$, правый — $m = 10$, $n = 3$. Звездочками указано положение ненулевых элементов K, nnz означает число ненулевых элементов K.

много» нулевых элементов, т. е. является разреженной. Напротив, при малых n и больших значениях m число нулевых элементов «невелико» и матрицу, можно считать заполненной. На рис. 2 указаны портреты K в этих двух случаях.

Рассмотрим, как устроен портрет матрицы K. С каждой точкой сетки $x_i \in \omega_h$ связана базисная функция $\varphi_i(x)$. Носителем функции f называется множество точек области ее определения, в которых функция отлична от нуля и обозначается $\mathrm{supp}(f)$. Точка x_i принадлежит либо одному элементу, либо — двум. Следовательно, $\mathrm{supp}\,(\varphi_i)$ состоит либо из одного, либо из двух элементов, т. к. $\varphi_i(x)$ отлична от нуля только на элементах, содержащих x_i. Поэтому $\mathrm{supp}\,(\varphi_i\varphi_j)$ (как и $\mathrm{supp}\,(\varphi_i'\varphi_j)$, $\mathrm{supp}\,(\varphi_i\varphi_j')$, $\mathrm{supp}\,(\varphi_i'\varphi_j')$) есть пустое множество, если x_i и x_j не принадлежат одному элементу, т. е. их номера таковы, что $|i-j| > m$. Отсюда и определения элемента K_{ij} следует, что $K_{ij} = 0$, если $|i-j| > m$. Далее, если x_i — внутренняя точка некоторого элемента e_ℓ, то $K_{ij} \neq 0$ только для тех j, для которых $x_j \in e_\ell$; т. е. на i-той строке K ненулевыми будут только $m+1$ подряд идущих элементов, включая диагональный. Наконец, если x_i есть общая точка двух элементов, то ненулевыми на i-той строке будут все элементы K_{ij} с номерами $|i-j| \leqslant m$, т. е. $2m+1$ элементов (см. рис. 2). Таким образом, число ненулевых элементов матрицы K есть

$$nnz(K) = (n-1)(m+1)^2 - (n-2).$$

Это число определяет затраты памяти и времени ЭВМ, необходимые

для вычисления и хранения K. Отметим, что в разреженном формате хранения матрицы память отводится только под ее ненулевые элементы и их индексы. Компьютерное время решения системы уравнений с матрицей типа K прямыми методами пропорционально величине порядка $O(n\,m^2 + m^3)$.

Отсюда следует, что числа n и m по разному влияют на затраты, необходимые для получения приближенного решения: затраты по n растут линейно, по m — квадратично (кубически, при малом n).

(D_2) Числа n и m оказывают существенное влияние и на точность приближенного решения u_h. Поясним сказанное, считая, что численное интегрирование не используется. Пусть u — решение задачи, коэффициенты оператора L таковы, что найдутся положительные числа α и β такие, что

$$\alpha\|v\|^2_{H^1} \leqslant \int_a^b (p\,|v'|^2 + r\,v\,v' + s\,v'\,v + q\,|v|^2)\,dx \leqslant \beta\|v\|^2_{H^1}$$

для любого $v \in V^0$,[1)] где

$$\|v\|^2_{H^k} = \int_a^b \left(v^2 + \left|d^k v/dx^k\right|^2\right) dx.$$

Пусть также $\|u\|_{H^s} < \infty$ для некоторого целого $s \geqslant 2$. В теории МКЭ установлено, что справедливы следующие оценки точности

$$\|u - u_h\|_{H^1} \leqslant C_1\,\frac{h^{k-1}}{m^{k-1}}\,\|u\|_{H^k}, \quad \max_{x\in[a,b]} |u(x) - u_h(x)| \leqslant C_2\,\frac{h^k}{m^k}\,\|u\|_{H^k},$$

для $2 \leqslant k \leqslant \min\{m+1, s\}$. Здесь C_1, C_2 некоторые постоянные, не зависящие от h и m. Обратим внимание на два случая: (a) s «мало» ($s = 2, \ldots, 5$, $s < m+1$); (b) s «велико» ($s \geqslant m+1$). Случай a) соответствует не очень гладкому решению задачи, в случае b) решение предполагается достаточно гладким.[2)] Выбирая максимальное k в оценках, получим, что

$$(a) \quad \|u - u_h\|_{H^1} = O\left(\left(\frac{h}{m}\right)^{s-1}\right), \quad (b) \quad \|u - u_h\|_{H^1} = O\left(\frac{h^m}{m^m}\right).$$

[1)] V^0 — пространство, фигурирующее в интегральном тождестве; $V^0 = H^1(a,b)$ для третьей краевой задачи. Это условие выполняется для широкого набора функций p, r, s, q, встречающихся в приложениях.

[2)] случай (b) имеет место, например, для аналитических функций (т. е. функций, допускающих разложение в ряд Тейлора, равномерно сходящийся в окрестности каждой точки $[a,b]$).

Видно, что m оказывает разное влияние на точность решения в этих двух случаях. Проясним эти оценки.

Пусть $h_{\min}$ есть минимальный размер используемых конечных элементов. Предположим, что при увеличении числа элементов n выполняется оценка $h_{\min}/h \geqslant c_0$ с постоянной c_0, не зависящей от n (квази-равномерные разбиения $[a,b]$; например, если при каждом n строится разбиение $[a,b]$ на равные элементы, то $h=(b-a)/n$). В этом случае $h=O(n^{-1})$ и оценкам можно придать следующий вид:

$$(a)\quad \|u-u_h\|_{H^1}=O\left(\frac{1}{(nm)^{s-1}}\right),\quad (b)\quad \|u-u_h\|_{H^1}=O\left(\frac{1}{(nm)^{m}}\right).$$

Отсюда следует, что точность приближенного решения степенным и одинаковым образом увеличивается с ростом n и m в случае (a); в случае (b), напротив, точность решения с увеличением n растет степенным образом, тогда как по m — экспоненциально.

Учитывая соображения, приведенные в п.п. (D_1) и (D_2), можно руководствоваться двумя стратегиями выбора параметров m и n при решении конкретной задачи:

1) выбрав небольшое значение m (на практике $m=1,\ldots,5$), и увеличивая n (уменьшая h — максимальный размер элементов), добиваться приемлемого решения задачи (значения $n\simeq 10^3$ не считаются большими); эта *стратегия называется h-версией МКЭ.*

2) выбрав небольшое значение n (допускается $n=2$, что соответствует использованию одного конечного элемента) и увеличивая m (на практике $m=5,\ldots,50\ldots$), добиваться приемлемого решения задачи; эта *стратегия называется p-версией МКЭ.*[1)]

Конечно, можно руководствоваться и третьей стратегией, которая называется *hp-версией МКЭ.* В этом случае степень полиномов на разных элементах может быть различной. Эта стратегия оказывается полезной в тех ситуациях, когда справедливы оценки (b), но в некоторых подобластях $[a,b]$ k-производная решения быстро растет с ростом k. В таких подобластях на элементах выбирают небольшие значения m и дробят элементы (уменьшают h), а в оставшейся области фиксируют элементы и увеличивают степени полиномов на них.

[1)] в теории МКЭ степень полиномов часто обозначается через p (по первой букве слова polynome).

(D_3) вектор $a = (a_i)_{i=1}^n$ определяет разбиение $[a, b]$ на элементы, причем необходимо, чтобы $a_1 = a$, $a_n = b$. Часто разбиение выбирается равномерным, так что $a_i = a + (i-1)h$, $h = (b-a)/(n-1)$. Если коэффициенты или решение задачи имеют особенность или большие производные в окрестности некоторой точки, то полезно около таких точек выбирать элементы меньших размеров, чем в остальной части области. Необходимо иметь в виду, что число обусловленности матрицы K_0 при наличии сгущения сетки имеет порядок $O(h_{\min}^{-2})$. Поэтому строить разбиения с слишком малым $h_{\min}$ не рекомендуется.[2)]

Также важно иметь ввиду следующее обстоятельство. Если коэффициенты или правая часть уравнения или их производные имеют особенность в окрестности некоторой точки (например, терпят разрыв первого рода в некоторых точках[1)]) то необходимо, чтобы эти точки совпадали с одними из a_i. В этом случае эти данные будут гладкими внутри конечных элементов и этого достаточно для обеспечения нужной точности МКЭ. В противном случае точность метода катастрофически снизится. В практике вычислений с такой ситуацией приходится часто сталкиваться.

(D_4) вектор узлов $t = (t_i)_{i=1}^{m+1}$ на базисном элементе $\Delta = [0, 1]$ определяет распределение узлов сетки на каждом элементе, $t_1 = 0$, $t_{m+1} = 1$. *Теоретически (в точной арифметике) решение u_h не зависит от того, как будут выбраны эти узлы.* Практически (при вычислениях на ЭВМ) их распределение имеет важное значение и, в частности, влияет на обусловленность матрицы K_0. При малых значениях m ($m < 7$) узлы t_i можно выбрать с равным шагом. При бóльших значениях m рекомендуется выбирать их так, чтобы они являлись корнями какого-либо ортогонально полинома (степени $m+1$). Например, удачным является их выбор в виде корней полинома $(1-t^2)U'_{m-1}(t)$, или $(1-t^2)L'_{m-1}(t)$, сдвинутых на $[0, 1]$, где U_n и L_n — полиномы Чебышева второго рода и полиномы Лежандра соответственно. В первом случае корни известны точно и $t_j = -\cos(\pi(j-1)/m)$, $1 \leqslant j \leqslant m+1$. Во втором случае они могут быть вычислены приближенно с машинной точностью и, что важно, они образуют узлы квадратурной формулы

[2)] Если не быть очень точным, то можно считать, что при решении системы уравнений $K_0 u = F_0$ с матрицей K_0, имеющей обусловленность порядка 10^p, $p > 0$, при вычислениях с d значащими цифрами в решении будут верными только $d - p$ значащих цифр.

[1)] Напомним, что для таких данных слабое решение корректно определено.

Лобатто (на отрезке $[-1,1]$), которая является точной на полиномах $2m-1$ степени (такая квадратура является подходящей при вычислении локальной матрицы жесткости).

(D_5) c, d — векторы длины M — веса и узлы квадратуры на Δ. Теория МКЭ рекомендует (а практика вычислений подтверждает) выбор кадратуры, удовлетворяющей трем требованиям:

1) коэффициенты кадратуры должны быть положительны;

2) $M \geqslant m$;

3) квадратура должна быть точной на полиномах степени $2m-1$ или выше.

Фактически, имеются лишь два типа квадратур, подходящих под эти требования при произвольном m. Это квадратура Гаусса (с числом узлов равным m или больше) и квадратура Лобатто (с числом узлов равным $m+1$ или больше).

2.3. Ортогональные полиномы и квадратуры типа Гаусса

Система полиномов $\{p_n(t)\}_{n=0}^{\infty}$ называется ортогональной на интервале (a,b) с весом ω (или просто ортогональной), если

$$(a)\ \ p_n \in \mathcal{P}_n; \qquad (b)\ \ \int_a^b \omega(t)\, p_n(t) p_m(t)\, dt = 0 \ \text{ при } \ n \neq m. \quad (1.36)$$

Здесь ω есть интегрируемая на (a,b) неотрицательная функция, не равная нулю на множестве положительной меры, называемая весовой функцией (или просто весом). Допускается случай $b-a=\infty$.

Нетрудно видеть, что ортогональные полиномы определяются неоднозначно (с точностью до постоянных множителей): наряду с системой $\{\tilde{p}_n\}$ ортогональной является также система $\{p_n = c_n \tilde{p}_n\}$, где c_n произвольные ненулевые числа. Способ выбора нормирующих множителей c_n называется стандартизацией. Например, c_n можно выбрать так, чтобы старший коэффициент p_n был равен единице или $p_n(b) = \delta_n$ при заданных δ_n.

Отметим наиболее важные с вычислительной точки зрения свойства ортогональных полиномов.

1) p_n имеет n различных корней на (a,b). Нули (а также экстремумы) p_n и p_{n+1} перемежаются.

2) Справедливы трехчленные рекуррентные соотношения

$$p_{k+1}(t) = (a_k t - b_k)p_k(t) - c_k p_{k-1}(t), \quad k \geqslant 0, \quad p_{-1}(t) = 0. \tag{1.37}$$

3) Корни $\{t_k\}_{k=1}^n$ полинома p_n являются собственными числами симметричной трехдиагональной матрицы

$$A_n = \begin{pmatrix} \alpha_0 & \beta_1^{1/2} & & & \\ \beta_1^{1/2} & \alpha_1 & \beta_2^{1/2} & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2}^{1/2} & \alpha_{n-2} & \beta_{n-1}^{1/2} \\ & & & \beta_{n-1}^{1/2} & \alpha_{n-1} \end{pmatrix},$$

где

$$\alpha_k = \frac{b_k}{a_k}, \quad k \geqslant 0, \qquad \beta_k = \frac{c_k}{a_{k-1}a_k}, \quad k \geqslant 1,$$

числа $\{a_k, b_k, c_k\}$ те же, что и в формуле (1.37).

4) Корни $\{t_k\}_{k=1}^n$ полинома p_n являются узлами квадратурной формулы типа Гаусса

$$\int_a^b \omega(t)\, f(t)\, dt \simeq \sum_{k=1}^n w_k f(t_k), \tag{1.38}$$

точной на полиномах $2n-1$ степени, веса которой положительны. Будем считать, что узлы $t = [t_1, t_2, \ldots, t_n]$ упорядочены по возрастанию и пусть $A_n U = U\Lambda$, $\Lambda = \mathrm{diag}(t)$ (т. е. столбцы матрицы U есть нормированные собственные векторы A_n, соответствующие t). Тогда вектор $w = [w_1, w_2, \ldots, w_n]$ весов квадратуры (1.38) совпадает с вектором $m_0 v_1$, где v_1 есть первая строка U, m_0 есть момент ω нулевого порядка, т. е.

$$m_0 = \int_a^b \omega(t)\, dt. \tag{1.39}$$

Важный класс ортогональных полиномов образуют полиномы Якоби, порождаемые весом

$$\omega(t) = (1-t)^\alpha (1+t)^\beta, \quad (a, b) = (-1, 1), \tag{1.40}$$

где параметры $\alpha, \beta > -1$. Они обозначаются через $P_n^{(\alpha,\beta)}(t)$ и стандартизируются равенством

$$P_n^{(\alpha,\beta)}(1) = \frac{\Gamma(n+\alpha+1)}{\Gamma(n+1)\Gamma(\alpha+1)},$$

где $\Gamma(x)$ есть гамма-функция. Квадратурная формула (1.38) в этом случае называется квадратурой Гаусса – Якоби, а при $\alpha = \beta = 0$ — квадратурой Гаусса (в этом случае вес $\omega \equiv 1$).

Отметим следующие частные случаи семейства полиномов Якоби. При $\alpha = \beta = 0$ эти полиномы известны как полиномы Лежандра и обозначаются через L_n; при $\alpha = \beta = -1/2$ — как полиномы Чебышева первого рода (T_n); при $\alpha = \beta = 1/2$ — как полиномы Чебышева второго рода рода (U_n). Отметим, что в двух последних случаях известны точные значения узлов и весов соответствующей квадратуры Гаусса (1.38).

Первые два полинома Якоби имеют следующий вид

$$P_0^{(\alpha,\beta)}(t) = 1, \quad P_1^{(\alpha,\beta)}(t) = 0.5\big((\alpha+\beta+2)t + \alpha - \beta\big),$$

а коэффициенты реккурентной формулы (1.37) равны

$$a_k = \frac{(2k+\alpha+\beta+1)(2k+\alpha+\beta+2)}{2(k+1)(k+\alpha+\beta+1)},$$
$$b_k = \frac{(\beta^2-\alpha^2)(2k+\alpha+\beta+1)}{2(k+1)(k+\alpha+\beta+1)(2k+\alpha+\beta)},$$
$$c_k = \frac{(k+\alpha)(k+\beta)(2k+\alpha+\beta+2)}{(k+1)(k+\alpha+\beta+1)(2k+\alpha+\beta)}.$$

Интеграл (1.39) выражается формулой

$$m_0 = 2^{\alpha+\beta+1}\frac{\Gamma(\alpha+1)\Gamma(\beta+1)}{\Gamma(\alpha+\beta+2)}.$$

Отметим следующую полезную формулу

$$\frac{d}{dt}\,P_n^{(\alpha,\beta)}(t) = 0.5(n+\alpha+\beta+1)P_{n-1}^{(\alpha+1,\beta+1)}(t), \tag{1.41}$$

которая позволяет, вычислять производные полиномов Якоби, используя рекуррентные формулы.

Квадратура Лобатто с n узлами определяется формулой

$$\int_{-1}^{1} f(t)\,dt \simeq w_1 f(-1) + \sum_{k=2}^{n-1} w_k f(t_k) + w_n f(1), \quad n \geqslant 2. \tag{1.42}$$

Она точна на полиномах $2n-3$ степени и имеет положительные веса. Ее внутренние узлы $\{t_k\}_{k=2}^{n-1}$ являются корнями полинома L'_{n-1} (или

согласно (1.41) корнями $P_{n-2}^{(1,1)}$). Коэффициенты квадратуры можно определить по следующей формуле:

$$\omega_k = \frac{2}{n(n-1)L_{n-1}^2(t_k)}\,, \quad k = 1, \ldots, n.$$

Отметим, что $L_n(-1) = (-1)^n$, $L_n(1) = 1$.

Численное интегрирование функций по произвольному конечному отрезку $[a, b]$ с использованием квадратур Гаусса и Лобатто осуществляется следующим образом. Используется линейная замена переменных в интеграле по формуле $x = (a + b + (b - a)t)/2$ и для полученного интеграла по отрезку $[-1, 1]$ используется соответствующая квадратура. Например в случае квадратуры Гаусса получим:

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f((a+b+(b-a)t)/2)\,dt \simeq$$
$$\simeq \frac{b-a}{2}\sum_{k=1}^{n} w_k f((a+b+(b-a)t_k)/2) = \sum_{k=1}^{n} c_k f(d_k),$$

где $c_k = (b-a)w_k/2$, $d_k = (a+b+(b-a)t_k)/2$ есть веса и узлы квадратуры Гаусса на отрезке $[a, b]$. Аналогично определяется квадратура Лобатто на $[a, b]$.

Узлы и веса квадратурных формул Гаусса и Лобатто на произвольном отрезке $[a, b]$ можно вычислить, используя следующие две функции, согласно приведенным выше формулам. Отметим, что узлы квадратуры Лобатто могут быть также выбраны в качестве узлов интерполяции на конечном элементе.

```
function [x,w] = qgauss(n,a,b)
% QGAUSS: find nodes and weights of n-point
% Gauss quadrature rule on [a,b].
% x = nodes=zeroes of Legendre polynomial L_n(x) on [a,b]
% w = weights.
% x,w  are column vectors

if nargin==1, a=-1; b=1; end
if n==1, x=(a+b)/2; w=b-a; return; end

j=1:n-1;
beta=j./sqrt(4*j.^2-1);

A=diag(beta,-1)+diag(beta,1);
```

```
% Find nodes x and weights w for [-1,1].
[V,D]=eig(A);
[x,k]=sort(diag(D));
w=2*V(1,k).^2;
w=w(:);

% nodes and weights on [a,b]
x=a+0.5*(b-a)*(x+1);
w=0.5*(b-a)*w;


function [x,w] = qlob(n,a,b)
% QLOB: find nodes and weights of
% n-point Lobatto quadrature rule on [a,b].
% x = nodes =zeros of (1-t^2)L'_{n-1}(t) shifted on [a,b],
% w = weights.
% x,w  are column vectors

if n<2,
  disp('QLOB: number of Lobatto nodes less then 2'); return;
end
if nargin==1, a=-1; b=1; end
if n==2, x=[a;b]; w=[(b-a)/2;(b-a)/2]; return; end

% Find nodes x and weights w of (n-2) point
% Gauss-Jacoby quadrature with weight 1-x^2 on [-1,1]

k=1:n-3;
beta=sqrt(k.*(k+2)./(2*k+1)./(2*k+3));

A=diag(beta,-1)+diag(beta,1);

[V,x]=eig(A);
[x,k]=sort(diag(x));
w=4/3*(V(1,k).^2)';

% Lobatto nodes and weights on [-1,1]
w=[2/(n^2-n); w./(1-x.^2); 2/(n^2-n)];
x=[-1; x; 1];

% Lobatto nodes and weights on [a,b]
x=a+0.5*(b-a)*(x+1);
w=0.5*(b-a)*w;
```

2.4. Функции, связанные с базисным элементом

Отнесем к базисному элементу Δ следующие данные: числа m, M и векторы c, d, t. Как отмечвлось равнее, имеется не так уж много вариантов задания этих векторов. Следующая функция определяет их в зависимости от типа узлов t ($typet$) и типа квадратуры ($typeq$).

```
function [t,d,c]=baselm(m,M,typet,typeq)
%% BASELM: set basic element data
% m     = polynomial degree
% M     = number of quadrature nodes
% typet = type of distribution of mesh points
% typeq = type of quadrature rule

switch lower(typet)
   case {'lin', 'linear'}
      t=linspace(0,1,m+1);
   case {'c', 'ch', 'chb', 'cheb'}
      t=sort((cos(pi*(0:m)/m)+1)/2);
   case {'l', 'lo', 'lob', 'lobat', 'lobatto'}
      t=qlob(m+1,0,1);
   otherwise
      t=qlob(m+1,0,1);
end
switch lower(typeq)
   case {'g', 'ga', 'gau', 'gaus', 'gauss'}
      [d,c]=qgauss(M,0,1);
   case {'l', 'lo', 'lob', 'lobat', 'lobatto'}
      [d,c]=qlob(M,0,1);
   otherwise
      [d,c]=qlob(M,0,1);
end
```

Опишем вычисление матриц I и D. Эти матрицы с необходимостью возникают при решении следующих задач: a) при вычислении локальных матриц жесткости и вектора сил; b) при построения графиков решения u_h и его производной u_h'; c) при вычисления погрешности решения. В каждом из этих случаев вектор d, фигурирующий в определении этих матриц, может иметь разную длину. Следующая функция вычисляет эти матрицы. Согласно определению базисных функций Лагранжа в ней реализованы следующие формулы:

$$b_\alpha = 1/\prod_{j=1\, j\neq\alpha}^{m+1}(t_\alpha - t_j), \quad I_{k,\alpha} = \hat{\varphi}_\alpha(d_k) = b_\alpha \prod_{j=1,\, j\neq\alpha}^{m+1}(d_k - t_j),$$

$$D_{k,\alpha} = \hat{\varphi}'_\alpha(d_k) = b_\alpha \sum_{s=1,\, s\neq\alpha}^{m+1} \prod_{j=1,\, j\neq\alpha,\, j\neq s}^{m+1}(d_k - t_j).$$

```
function [I,D]=basfnc(t,d)
%% BASFNC find interpolation (I) and differentiation matrix (D).
% t = interpolation points
% d = arbitrary nodes
% I ={\phi_j (d_i), j=1,..,n, i=1,...,N},
% D ={\phi'_j(d_i), j=1,..,n, i=1,...,N},
```

```
% \phi_j=Lagrange basis functions: \phi_j(t_i)=\delta_ij.

n=numel(t); N=numel(d);
I =zeros(N,n); D =zeros(N,n);
b=zeros(1,n);
for i=1:n
  j=[1:(i-1),(i+1):n];
  b(i)=1/prod(t(i)-t(j));
end

for i=1:n
  j=[1:(i-1),(i+1):n];
  for k=1:N
    I(k,i)=b(i)*prod(d(k)-t(j));      % \phi_i(d_k)
    ds=0;
    for s=j
      i1=min(i,s); i2=max(i,s);
      jj=[1:(i1-1),(i1+1):(i2-1), (i2+1):n];
      ds=ds + prod(d(k)-t(jj));
    end
    D(k,i)=b(i)*ds;                   % \phi'_i(d_k)
  end
end
```

УПРАЖНЕНИЯ.

Разработка быстрых и устойчивых к ошибкам округления алгоритмов вычисления матриц I и D имеет важное значение и является актуальной задачей. В связи с этим

1. Проверьте, что число арифметических операций, затрачиваемых на вычисление I и D в приведенной выше функции $basfnc$, имеет порядок $O(n^2\,N)$ и $O(n^3\,N)$ соответственно. Отсюда можно заключить, что уже при n и N порядка сотни функция будет работать медленно (особенно, если учесть, что MatLab является интерпретатором и глубоко вложенные циклы в MatLab неэффективны).

2. Экспериментально определите зависимость времени работы функции $basfnc$ от n в случае $t = d$. Согласуются ли полученные числа с оценками в упражнении 1?

3. В случае $t = d$ матрица I является единичной. Покажите, что внедиагональные элементы D можно вычислить по формуле

$$D_{ij} = \frac{\beta_j\,/\beta_i}{t_i - t_j}, \quad i \neq j\,,$$

а диагональные элементы по формуле $D_{ii} = -\sum_{j=1,\,j\neq i}^{n} D_{ij}$. Трудоемкость этого метода вычисления D есть величина $O(n^2)$, что существенно лучше, чем $O(n^4)$. Напишите соответствующую этим формулам функцию MatLab — аналог функции $basfnc$.

Указание. Воспользуйтесь барицентрической формулой записи базисной функции.

4. Напишите аналог функции $basfnc$ на основе барицентрической формы записи базисных функций в случае, когда ни один узел t не совпадает ни с одним узлом d. Оцените трудоемкость полученной функции.

5. Предложите свой метод вычисления матриц I и D.

2.5. Функции сборки и решения системы МКЭ

Формирование системы МКЭ, как было описано ранее, состоит из двух шагов. На первом осуществляется сборка K и F; на втором — учитываются краевые условия и получаются K_0 и F_0. Следующая функция реализует первый шаг; обозначения переменных полностью соответствует данному выше описанию алгоритмов.

```
function [K,F] = assemKF(p,r,s,q,f,a,t,d,c)
%% ASSEMKF: Assembles element contributions to FEM system.
% Stiffness matrix K and force vector F corresponds to the equation
%       -(pu'+ru)'+su'+qu=f,  a1<x<b1,
% and homogenous Neumann bondary conditions  pu'+ru=0.
% input data are commented in LINBVP

[v,dv]=basfnc(t,d); u=v'; du=dv'; % matrices I, D, I^T, D^T

m=numel(t)-1;  n=numel(a);  N=(n-1)*m+1;

K=sparse(N,N);     % stiffness matrix
F=zeros(N,1);      % rhs
for l=1:n-1
  h =a(l+1)-a(l); % size of element l
  nl=(l-1)*m;     % nl+j = global index of point j on element l
  ne=nl+(1:m+1);  % global point indexes on element l
  xd =a(l)+h*d;   % quadrature nodes on element l

  P=diag((c/h).*p(xd));
  R=diag(   c.*r(xd));
  S=diag(   c.*s(xd));
  Q=diag((h*c).*q(xd));

  Kl=(du*P+u*S)*dv + (du*R+u*Q)*v;
  K(ne,ne)=K(ne,ne)+Kl;

  Fl=(h*c).*f(xd);
  F(ne)=F(ne) + u*Fl(:);
end
```

Обратим внимание на команду $K(ne, ne) = K(ne, ne) + Kl$. Здесь K матрица $N \times N$, а Kl — $(m+1) \times (m+1)$, и мы используем векторную адресацию на элементы K. Поскольку ne содержит глобальные номера узлов на элементе K_l, то тем самым элемент $Kl_{\alpha\beta}$ добавляется в $K_{ne(\alpha),ne(\beta)}$ при всех $\alpha, \beta = 1, \ldots, m+1$, что и требуется. Такой простой способ рассылки элементов эффективен для не слишком больших значений n (см. далее гл. 2).

Следующая функция вычисляет матрицу и правую часть системы МКЭ. Она модифицирует матрицу жесткости и вектор сил по-

средством учета краевых условий.

```
function [K0,F0] = assemBC(bc,K,F)
%% ASSEMBC: Assembles boundary conditions to
% stiffness matrix (K) and rhs (F)
% bc = boundary conditions matrix is commented if LINBVP

N=numel(F);  % number of all mesh points

% set boundary conditions of 3-d type
if bc(1,1)==3
   K(1,1)=K(1,1)+ bc(1,2);
   F(1)  =F(1)  + bc(1,3);
end
if bc(2,1)==3
   K(N,N)=K(N,N)+ bc(2,2);
   F(N)  =F(N)  + bc(2,3);
end

% set boundary conditions of 1 type
ib=1; ie=N;
if bc(1,1)==1, ib=2;    ua=bc(1,3); F=F-ua*K(:,1); end
if bc(2,1)==1, ie=N-1; ub=bc(2,3); F=F-ub*K(:,N); end

I=(ib:ie)';
K0=K(I,I);
F0=F(I);
```

После того как результирующая система МКЭ сформирована предыдущей функцией, остается решить эту систему и получить вектор узловых параметров приближенного решения. Это обеспечивается следующей функцией.

```
function y = solveLbvp(bc,K0,F0)
%% SOLVELBVP: Solve by FEM a linear BVP for 2-nd order ODE
%  bc    = defined in LINBVP
%  K0, F0    = resulting stiffness matrix and rhs of fem system
%  y = fem solution (column vector)

% get Dirichlet data
ua=[]; ub=[];
if bc(1,1)==1, ua=bc(1,3); end
if bc(2,1)==1, ub=bc(2,3); end

u0=K0\F0;
y=[ua; u0; ub];
```

Предыдущие три функции удобно объединить в одну функцию, которая по исходным данным позволяет найти МКЭ-решение.

```
function [y,x] = linbvp(bc,p,r,s,q,f,a,t,d,c)
% LINBVP: Solve by FEM a BVP for 2-nd order ODE
%       -(pu'+ru)'+su'+qu=f,  a<x<b.
% with boundary conditions:
%     u(a)=ua (type=1)  OR   -(pu'+ru)(a)+siga u(a)=ga (type=3)
%     u(b)=ub (type=1)  OR    (pu'+ru)(b)+sigb u(b)=gb (type=3).
%
%  bc   = boundary conditions matrix of size 2x3:
%         bc(1,:)=[1, 0, ua], OR  bc(1,:)=[3, siga, ga]
%         bc(2,:)=[1, 0, ub], OR  bc(2,:)=[3, sigb, gb]
%
%  p,r,s,q,f = pointers of vectorized functions of one variable x
%  a(1:n)    = f.e. boundary points
%  t(1:m+1)  = mesh points on basic element [0,1],
%              m=degree polynomial on each finite element
%  (d,c)(1:M)= quadrature nodes and weights on basic element [0,1]
%
%  y = fem solution on mesh x
%  y,x are column vectors

m=numel(t)-1; n=numel(a);
N=(n-1)*m+1;              % number of mesh points

x=zeros(N,1);             % all mesh points
for l=1:n-1
  nl=(l-1)*m;             % nl+j = global index of point j on element l
  ne=nl+(1:m+1);          % global point indexes on element l
  x(ne)=a(l)+(a(l+1)-a(l))*t;  % mesh points on element l
end

[K,F]   = assemKF(p,r,s,q,f,a,t,d,c);
[K0,F0] = assemBC(bc,K,F);
y =solveLbvp(bc,K0,F0);
```

2.6. Построение графиков. Погрешность решения

Следующая функция finesol вычисляет решение u_h и его производную u_h' на мелкой сетке узлов X, которая получается делением каждого элемента на k частей (например, $k = 3m$). Эта функция полезна при построении графиков u_h и u_h'.

```
function [X,Y,dY]=finesol(y,a,t,k)
%% FINESOL: find fem solution and its derivative on fine mesh
%  y(1:N)    = fem solution
%  a(1:n)    = f.e. boundary points
%  t(1:m+1)  = grid points on basic element [0,1]
%  k+1       = number of fine mesh points on each f.e.
%  X         = fine mesh
%  Y,dY      = fem solution and its derivative on fine mesh X
%  X,Y,dY    are column vectors
```

```
tf=linspace(0,1,k+1); % fine mesh on basic element
[I,D]=basfnc(t,tf);

n=numel(a);  m=numel(t)-1;
Nf=(n-1)*k+1;          % number of all fine mesh points

X=zeros(Nf,1);
Y=zeros(size(X));
dY=zeros(size(X));
for l=1:n-1
  h=a(l+1)-a(l);  % size of element l
  nc=(l-1)*m;     % nc+j = global index of (coarse) point j on element l
  nf=(l-1)*k;     % nf+j = index of fine point j on element l

  c=nc+(1:m+1);   % global coarse point indexes on element l
  f=nf+(1:k+1);   % global fine points indexes on element l

  X(f)=linspace(a(l),a(l+1),k+1)';
  Y(f)=I*y(c);
  dY(f)=D*y(c)/h;
end
```

При анализе точности рассматриваемого метода на тестовых задачах с известным решением u полезна следующая функция для вычисления погрешности $e = u - u_h$ в нормах пространств L_2 и H_0^1 :

$$|e|_0 = \Big(\int_a^b e^2(x)\,dx \Big)^{1/2}, \quad |e|_1 = \Big(\int_a^b \big(e'(x)\big)^2\,dx \Big)^{1/2}.$$

В femerr для вычисления интегралов на каждом элементе используется квадратура Гаусса с $(m+2)$ узлами.

```
function [e0,e1]=femerr(u,du,y,a,t)
%% FEMERR: find fem solution error in different norms
%  u, du     = pointers to functions of exact solution u and u'
%  y(1:N)    = fem solution on mesh x
%  a(1:n)    = f.e. boundary points
%  t(1:m+1)  = grid points on basic element [0,1]
%  e0=\|u-u_h\|_L_2,    e1=\|u-u_h\|_H^1_0,

m=numel(t)-1;

[d,c] = qgauss(m+2,0,1); % new quadrature on [0,1]
[I,D] = basfnc(t,d);

e0=0; e1=0;
for l=1:numel(a)-1
  h =a(l+1)-a(l); % size of element l
  dl=a(l)+h*d;    % quadrature nodes on e_l
  nc=(l-1)*m;     % nc+j = global index of  point j on element l
  ic =nc+(1:m+1); % global point indexes on element l
```

```
  uh =I*y(ic);      duh=D*y(ic)/h;
  e  = u(dl)-uh;    de = du(dl)-duh;

  e0=e0 + h/2*sum(c.*e.^2);
  e1=e1 + h/2*sum(c.*de.^2);
end
e0=e0^(1/2); e1=e1^(1/2);
```

2.7. Подготовка данных. Решение тестовой задачи

Используя функцию linbvp можно решить конкретную краевую задачу, если определить bc, p, r, s, q, f, а также задать разбиение области на элементы и определить базисный элемент. Для тестирования метода также желательно иметь точное решение задачи для определения погрешности найденного решения.

Тестовую задачу можно построить самостоятельно следующим образом. 1) Зададим отрезок $[a, b]$, а также функции p, r, s, q — коэффициенты оператора L краевой задачи. 2) фиксируем тип краевых условий; например, первого рода в точке $x = a$ и третьего рода в точке $x = b$. 3) зададим значение σ_b в краевом условии третьего рода. 4) выбирем точное решение u задачи и найдем его производную u'. 5) определим правую часть уравнения f из равенства $f = Lu$. 6) определим правые части краевых условий. В рассматриваемом случае положим $u_a = u(a)$, $g_b = (pu' + ru)(a) + \sigma_b u(b)$. В результате получим смешанную краевую задачу с известным точным решением.

Если в Вашей инсталляции MatLab есть symbolic toolbox, то вычисление производных и данных задачи можно провести в символьном виде, например, так, как это сделано ниже. Данные о краевой задаче удобно задавать в отдельном файле (bvpdata), оформленном в виде script. В этом файле x есть символьная переменная, и, как следствие, символьными являются также все функции, которые определяются в терминах x. Функция f вычисляется с использованием функции символьного дифференцирования. Значения u_a, u_b и g_a, g_b в краевых условиях первого и третьего рода подсчитываются согласно краевым условиям (возможно избыточно). Конкретный набор краевых условий определяется заданием двух чисел *tbca* и *tbcb*. Таким образом, мы можем легко определить краевую задачу с известным

точным решением при любом сочетании краевых условий. Команда p=matlabFunction(p,'vars',x) преобразует символьную функцию p в неименованную функцию с указателем p.

```
% BVPDATA: set linear bvp data's
% with exact solution
% WARNING: SYMBOLIC TOOLBOX is needed

syms x

% BVP numeric data's
a=0; b=1;          % integration domain
tbca=1; tbcb=3;  % type of bc condition
siga=1; sigb=2;  % if you need.

% BVP functional data
w=12;              % solution parameter
u = sin(w*x);      % exact solution and other in symbolic form
p = 2+x.*cos(x);
r = 1;
s = 2*x;
q = sin(x);

%find the rest of data's
du=diff(u);                    % du=u'
f=-diff(p*du+r*u)+s*du+q*u;  % rhs

p=matlabFunction(p,'vars',x);
r=matlabFunction(r,'vars',x);
s=matlabFunction(s,'vars',x);
q=matlabFunction(q,'vars',x);
f=matlabFunction(f,'vars',x);
u=matlabFunction(u,'vars',x);
du=matlabFunction(du,'vars',x);

ua=u(a);   ub=u(b);
if tbca==1
  sa=0; ga=ua;
else
  sa=sigb;
  ga=-(p(a)*du(a)+r(a)*u(a))+siga*ua;
end

if tbcb==1
  sb=0; gb=ub;
else
  sb=sigb; gb=(p(b)*du(b)+r(b)*u(b))+sigb*ub;
end
bc=[ tbca sa ga
     tbcb sb gb ];
```

Так просто определить краевую задачу можно лишь в том случае, если имеется доступ к symbolic toolbox. Иначе вычисление правой ча-

сти уравнения (f) и производной решения (du) необходимо провести вручную. Можно использовать следующую функцию, вставляя вместо многоточий нужные выражения.

```
% BVPDATA: set linear bvp data's
% with exact solution

syms x

% BVP numeric data's
a=0; b=1;          % integration domain
tbca=1; tbcb=3;  % type of bc condition
siga=1; sigb=2;  % if you need.

% BVP functional data
w=12;                    % solution parameter
u = @(x) sin(w*x);      % exact solution
p = @(x) 2+x.*cos(x);
r = @(x)  1;
s = @(x) 2*x;
q = @(x) sin(x);

%Next, find the rest of the data's
du=.....;  % du=u'
f =.....;  % rhs

ua=u(a);  ub=u(b);
if tbca==1
  sa=0; ga=ua;
else
  sa=sigb;
  ga=-(p(a)*du(a)+r(a)*u(a))+siga*ua;
end

if tbcb==1
  sb=0; gb=ub;
else
  sb=sigb; gb=(p(b)*du(b)+r(b)*u(b))+sigb*ub;
end
bc=[ tbca sa ga
     tbcb sb gb ];
```

Теперь остается написать основную функцию (script), которая позволит проводить различные численные эксперименты при решении краевой задачи, определенной в bvpdata. Простой вариант такой функции приводится ниже. Меняя коэффициенты уравнения, тип краевых условий, параметры МКЭ, можно наблюдать за точностью метода. Интерес представляет зависимость погрешности и времени решения задачи от величин h, m и от общего числа узлов сетки N.

```
% main file
clc; clear all; close all

bvpdata;  % set ODE and bc data's

% set fem data's
m =3;                    % polynomial degree
ne=6;                    % number of f.e., n=ne+1
M =m+1;                  % number of quadrature  nodes
xa=linspace(a,b,ne+1); % [a,b]=sum of e_i=[xa_i,xa_{i+1}]
[t,d,c]=baselm(m,M,'lob','lob'); % basic element data's

% solve bvp
tic
[y,x] = linbvp(bc,p,r,s,q,f,xa,t,d,c); % y=fem solution on mesh x
timeFemSol=toc

% find error in L_2 and H^1 norms
[e0,e1]=femerr(u,du,y,xa,t)

[X,Y,dY]=finesol(y,xa,t,10*m);  % Y=solution on the fine grid X

% find error in different mesh-max norm
Z=u(X)-Y;              % u-uh   on grid X
z=u(x)-y;              % u-uh   on grid x
dZ=du(X)-dY;           % u'-uh' on grid X

% error in max-norm
ie=1:m:numel(x);  % indexes of elements boundary nodes
einfh=norm(Z,inf)
einfdh=norm(dZ,inf)
einfbh=norm(u(xa(:))-y(ie),inf)

figure;  % plot solution
plot(X,Y,'-b',x,y,'rx',x(ie),y(ie),'ro')
xlabel('x');  ylabel('u_h')
title('Values at f.e. boundary -- o, at mesh points -- x')

figure;% plot solution error
plot(X,Z,'-b',x,z,'rx',x(ie),z(ie),'rs')
xlabel('x'); ylabel('u-u_h')

figure;% plot derivatives error
plot(X,dZ,'-b',x,zeros(size(x)),'rx',x(ie),zeros(size(x(ie))),'rs')
xlabel('x'); ylabel('u''-u''_h')
```

Часть результатов этой программы представлена на рис. 3.

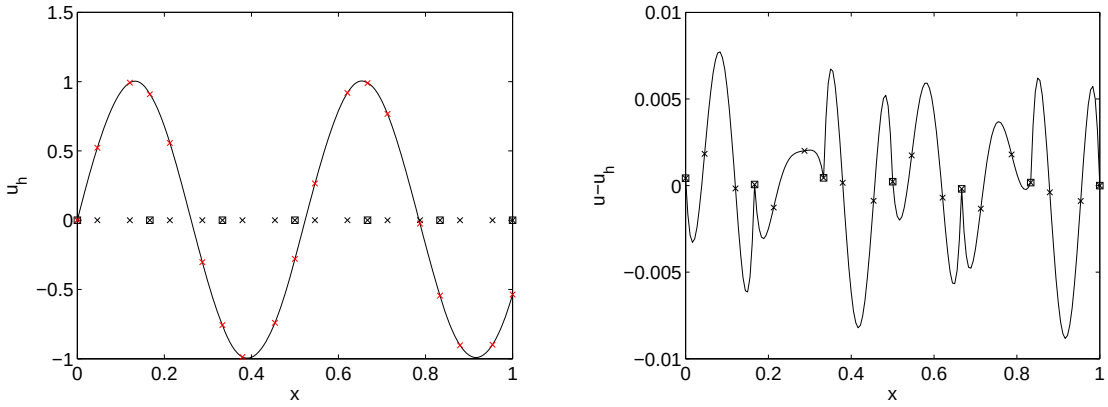

Рис. 3. Решение МКЭ (слева) и распределение его погрешности (справа). Конечных элементов — 6 ($n = 7$), на каждом элементе u_h есть полином степени $m = 3$.

УПРАЖНЕНИЯ.

1. Напишите функцию, которая при фиксированном m определяет зависимость погрешности решения схемы МКЭ в нормах C, L_2 и H_0^1 от шага сетки h и от N.

Указание. Решите тестовую задачу на последовательности сеток с шагами $h = [h_1, h_2, \ldots, h_k]$ (т. е. с числом элементов $n = [n_1, n_2, \ldots, n_k]$) и вычислите погрешности $e = [e_1, e_2, \ldots, e_k]$ (например в норме C). Если зависимость погрешности от h степенная, т. е. $e_i = c\, h_i^\alpha$ при некотором α, то построив график функции $e = e(h)$ в двойных логарифмических координатах командой loglog(h,e,'-b.') вы должны увидеть фактически график линейной функции, наклон которой и есть α (почему?). Найдите формулу для вычисления α. Аналогично рассмотрите зависимость $e = e(N)$. Положите $n_i = n_0 2^i$, $n_0 = 2, ..., 5$.

2. Напишите функцию, которая при фиксированном h определяет зависимость погрешности решения схемы МКЭ в нормах C, L_2 и H_0^1 от m и от N.

2.8. Примеры тестовых задач

Далее приводятся примеры тестовых задач, решение u которых обладает той или иной особенностью. В этих задачах параметр $\varepsilon > 0$ и особенность решения проявляется при малых ε (кроме некоторых задач). Прежде чем использовать эти задачи в качестве тестовых, необходимо убедиться в их правильности! Во всех задачах можно выбрать краевые условия Дирихле согласно точному решению.

1. $-\varepsilon\, u'' + u = 0$, $x \in (0, 1)$. Точное решение имеет пограничный слой ширины $O(\varepsilon^{1/2})$ при $x = 0$:

$$u = \big(\exp(-x/\varepsilon^{1/2}) - \exp((x-2)/\varepsilon^{1/2})\big)/\big(1 - \exp(-2/\varepsilon^{1/2})\big).$$

2. $-\varepsilon\, u'' + u' = 0$, $x \in (0,1)$. Точное решение имеет пограничный слой ширины $O(\varepsilon)$ при $x = 1$:

$$u = \big(1 - \exp((x-1)/\varepsilon)\big)/\big(1 - \exp(-1/\varepsilon)\big).$$

3. $-\varepsilon\, u'' - pu' + u = (1 + \varepsilon\,\pi^2)\cos(\pi x) + p\,\pi\,\sin(\pi x)$, $x \in (-1,1)$. Здесь $p = 2 + \cos(\pi x)$. Точное решение является гладким и не зависит от ε : $u = \cos(\pi x)$.

4. $-\varepsilon\, u'' - u' + (1+\varepsilon)u = 0$, $x \in (-1,1)$. Решение имеет пограничный слой ширины $O(\varepsilon)$ при $x = -1$:

$$u = \exp(x-1) + \exp(-(1+\varepsilon)(1+x)/\varepsilon).$$

5. $-\varepsilon\, u'' + xu' + u = (1 + \varepsilon\,\pi^2)\cos(\pi x) - \pi x \sin(\pi x)$, $x \in (-1,1)$. Точное решение является гладким и не зависит от ε : $u = \cos(\pi x)$.

6. $-\varepsilon\, u'' - xu' = \varepsilon\,\pi^2 \cos(\pi x) + \pi x \sin(\pi x)$, $x \in (-1,1)$. Точное решение имеет внутренний слой при $x = 0$:

$$u = \cos(\pi x) + \mathrm{erf}(x/(2\varepsilon)^{1/2})/\mathrm{erf}(1/(2\varepsilon)^{1/2}).$$

7. $-(\varepsilon + x^2)\, u'' - 4xu' - 2u = 0$, $x \in (-1,1)$. Точное решение имеет всплеск высоты $1/\varepsilon$ при $x = 0$: $\quad u = 1/(\varepsilon + x^2)$.

8. $-\varepsilon\, u'' - xu' = 0$, $x \in (-1,1)$. Точное решение резко меняется в слое шириной $O(\varepsilon^{1/2})$ с центром в точке перегиба $x = 0$:

$$u = 1 + \mathrm{erf}(x/(2\varepsilon)^{1/2})/\mathrm{erf}(1/(2\varepsilon)^{1/2}).$$

9. $-\varepsilon\, u'' + u = (1 + \varepsilon\,\pi^2)\cos(\pi x)$, $x \in (-1,1)$. Точное решение имеет пограничный слой шириной $O(\varepsilon^{1/2})$ при $x = \pm 1$:

$$u = \cos(\pi x) + \exp((x-1)/\varepsilon^{1/2}) + \exp(-(x+1)/\varepsilon^{1/2}).$$

Глава 2

МКЭ ДЛЯ ДВУМЕРНЫХ ЭЛЛИПТИЧЕСКИХ ЗАДАЧ

В данной главе рассматриваются основные вопросы численной реализации МКЭ на примере краевой задачи для линейного дифференциального уравнения второго порядка эллиптического типа в двумерной области. Предполагается, что читатель знаком с основами теории дифференциальных уравнений в частных производных, с основами теории МКЭ и программирования в MatLab.

Как и ранее, знак «:=» означает, что левая часть выражения определяется его правой частью (равенство по определению).

§ 1. Исходная краевая задача

Далее рассматривается линейное дифференциальное уравнение в частных производных второго порядка в плоской ограниченной области Ω следующего вида

$$-\nabla \cdot (\, c\, \nabla u) + b \cdot \nabla u + a\, u = f, \quad x \in \Omega. \tag{2.1}$$

Здесь скалярные функции c, a, f, а также вектор-функция $b = (b_1, b_2)$, предполагаются заданными и достаточно гладкими функциями переменной $x = (x_1, x_2) \in \overline{\Omega}$; $c(x) > 0$ на $\overline{\Omega}$; функция $u = u(x)$ — подлежит определению. Для записи уравнения мы воспользовались дифференциальными операторами градиента (∇) и дивергенции ($\nabla\cdot$). Для скалярных функций v и векторных $q = (q_1, q_2)$ по определению

$$\nabla v = \Big(\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2} \Big), \quad \nabla \cdot q = \frac{\partial q_1}{\partial x_1} + \frac{\partial q_2}{\partial x_2}.$$

Использование этих операторов позволяет сокращать запись различных дифференциальных выражений. Так, уравнение

$$-\sum_{i=1}^{2} \frac{\partial}{\partial x_i}\Big(c\, \frac{\partial u}{\partial x_i}\Big) + \sum_{i=1}^{2} b_i\, \frac{\partial u}{\partial x_i} + a\, u = f$$

есть эквивалентная запись уравнения (2.1).

Далее предполагается, что Ω имеет кусочно-гладкую (липшицеву) границу Γ, состоящую из двух необязательно связных частей Γ_0 и Γ_1. Подмножество Γ_0 границы считается замкнутым множеством, так что $\Gamma = \Gamma_0 \cup \Gamma_1$, $\Gamma_0 \cap \Gamma_1 = \emptyset$.

Дополнительные условия вида

$$u = u_D, \quad x \in \Gamma_0, \tag{2.2}$$

$$c\,\nabla u \cdot \nu + \sigma\, u = \mu, \quad x \in \Gamma_1, \tag{2.3}$$

называются смешанными краевыми условиями для уравнения (2.1), а задача определения функции u, удовлетворяющей соотношениям (2.1)–(2.3) — смешанной краевой задачей. В этих условиях σ, μ, u_D есть заданные функции (на соответствующих участках границы); $\nu = \nu(x)$ — единичный вектор внешней нормали в точке $x \in \Gamma_1$; точка означает скалярное произведение векторов.

Условие (2.2) принято называть первым краевым условием или условием Дирихле, а условие (2.3) — третьим краевым условием. Отметим, что условие Дирихле может быть задано на всей границе Γ ($\Gamma_1 = \emptyset$), как и условие третьего рода ($\Gamma_0 = \emptyset$).

Задачу (2.1)–(2.3) *назовем, для краткости, задачей* $\mathcal{P}$.

Обозначим через $C^k(\Omega)$ множество непрерывных и k раз непрерывно дифференцируемых на Ω функций ($C(\Omega) = C^0(\Omega)$). Аналогично определяются $C^k(\overline{\Omega})$.

Функция $u \in C^2(\Omega) \cap C(\overline{\Omega})$ *такая, что ее частные производные первого порядка имеют непрерывное продолжение на* Γ_1, *и удовлетворяющая в каждой точке* $x \in \Omega$ *уравнению* (2.1), *а также краевым условиям* (2.2), (2.3), *называется классическим решением краевой задачи* $\mathcal{P}$.

Чтобы это определение имело смысл, достаточно предполагать, что $c \in C^1(\Omega) \cap C(\overline{\Omega})$, $b, a, f \in C(\Omega)$, а функции σ, μ, u_D — непрерывными на соответствующих участках границы. Далее будем считать, что определенное таким образом решение существует, единственно и сосредоточимся на методе МКЭ его приближенного нахождения. Метод конечных элементов основан на интегральном тождестве, соответствующем краевой задаче. Получим это тождество.

Напомним, что $L_2(\Omega)$ определяется как линейное пространство измеримых по Лебегу функций v, имеющих конечный интеграл

$$\int_\Omega v^2(x)\,dx\,,$$

понимаемый в смысле Лебега. Множество функций $H^1(\Omega)$ — пространство Соболева, определяется следующим образом:

$$H^1(\Omega) := \{v \in L_2(\Omega) :\ \partial v/\partial x_i \in L_2(\Omega),\ i = 1, 2\}.$$

Здесь частные производные понимаются по Соболеву. Известно, что всякая непрерывная на $\overline{\Omega}$ функция, имеющая кусочно непрерывные частные производные, принадлежит $H^1(\Omega)$.[1)] Кроме того, если q такая непрерывная на $C(\overline{\Omega})$ вектор-функция, что $\nabla \cdot q \in C(\Omega)$, то для любого $v \in H^1(\Omega)$ справедлива формула Остраградского – Гаусса

$$\int_\Omega \nabla \cdot q\ v\,dx = -\int_\Omega q \cdot \nabla v\,dx + \int_\Gamma q \cdot \nu\, v\,dx.$$

Определим следующие подмножества $H^1(\Omega)$:

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

Умножим обе части уравнения (2.1) на функцию $v \in V^0$ и проинтегрируем полученное равенство по области Ω. Воспользуемся формулой Остраградского – Гаусса при $q = -c\,\nabla u$. Придем к равенству

$$\int_\Omega (c\,\nabla u \cdot \nabla v + b \cdot \nabla u v + a u v)\ dx - \int_{\Gamma_1} (c\,\nabla u \cdot \nu) v\,dx = \int_\Omega f v\,dx\,, \quad (2.4)$$

поскольку $v = 0$ на Γ_0. После учета краевого условия (2.3) во внеынтегральном слагаемом получим равенство

$$\int_\Omega (c\nabla u \cdot \nabla v + b \cdot \nabla u v + a u v)\ dx + \int_{\Gamma_1} \sigma u v\,dx = \int_\Omega f v\,dx + \int_{\Gamma_1} \mu v\,dx\,, \quad (2.5)$$

[1)] Для определения схемы МКЭ далее под $H^1(\Omega)$ достаточно понимать множество непрерывных, кусочно непрерывно дифференцируемых на $\overline{\Omega}$ функций, понимая под $\partial v/dx_i$ «кусочную» производную v, а под интегралом — интеграл Римана.

справедливое для любого $v \in V^0$.

Соотношение (2.5) при $u \in V$ принято называть *интегральным тождеством, соответствующим краевой задаче* $\mathcal{P}$.

Заметим, что оно в некотором смысле эквивалентно задаче $\mathcal{P}$. Действительно, если u — решение задачи $\mathcal{P}$, то u удовлетворяет тождеству (2.5). Обратно, пусть u удовлетворяет тождеству (2.5) и обладает необходимой для существования классического решения гладкостью. Применяя обратно формулу интегрирования по частям в (2.5), в силу произвольности функции v, нетрудно получить уравнение (2.1), а также краевое условие (2.3). Это и означает, что u есть решение задачи $\mathcal{P}$, поскольку из условия $u \in V$ следует краевое условие (2.2).

Нетрудно видеть, что тождество (2.5) сохраняет смысл при гораздо более слабых предположениях относительно входящих в него функций чем те, которые были сформулированы ранее. Это позволяет дать другое определение решения задачи $\mathcal{P}$.

Функция $u \in V$ называется обобщенным (или слабым) решением задачи $\mathcal{P}$, если она удовлетворяет тождеству (2.5) *для любой функции $v \in V^0$.*

Интегральное тождество часто используют при построении методов приближенного решения задачи $\mathcal{P}$. Например, на нем основан метод Галеркина, частным случаем которого является МКЭ.

УПРАЖНЕНИЯ.

1. Докажите, что из тождества (2.5) следует, что справедливо уравнение (2.1) во всех точках $x \in \Omega$, а также краевое условие (2.3), при условии, что u есть достаточно гладкая функция.

Указание. Используйте формулу интегрирования по частям в (2.5). Выбором $v \in C^\infty(\Omega)$, $v \equiv 0$ вне $S_r(x_0)$, предельным переходом при $r \to 0$ получите уравнение ($S_r(x_0) \subset \Omega$ есть круг радиуса r с центром в точке x_0). Аналогично получите краевое условие.

2. Сформулируйте интегральное тождество и определите множества функций V и V^0 для уравнения (2.1) при следующих краевых условиях:

a) на всей границе Γ задано краевое условие третьего рода;

b) на всей границе Γ задано краевое условие Дирихле.

c) на всей границе Γ задано однородное краевое условие Неймана $c\,\nabla u \cdot \nu = 0$.

3. Дайте определение обобщенного решения краевых задач предыдущего пункта. Проверьте, что гладкое обобщенное решение является решением краевой задачи.

§ 2. МКЭ на основе лагранжевых элементов

Пусть $\mathcal{T}_h$ — разбиение (триангуляция) области Ω на лагаранжевы конечные элементы τ одного типа (аффинно-эквивалентные, изопараметрические или криволинейные элементы) [2]; предполагается, что все граничные точки Γ_0 являются вершинами каких-либо элементов. 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 $\mathcal{T}_h$.

Введем обозначения: $\Omega_h = \bigcup_{\tau\in\mathcal{T}_h} \tau$; Γ_0^h, Γ_1^h — части границы области Ω_h, соответствующие Γ_0 и Γ_1 (Γ_0^h замкнуто); $\omega_h = \bigcup_{\tau\in\mathcal{T}_h} \omega_\tau$ — множество всех узлов интерполяции (сетка узлов в $\overline{\Omega}$), γ_h множество узлов интерполяции, принадлежащих Γ_0^h. Определим пространство конечных элементов (аппроксимацию $H^1(\Omega)$):

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

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 (2.5) 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

$$\int_{\Omega_h} (c\nabla u_h \cdot \nabla v_h + b \cdot \nabla u_h v_h + a u_h v_h)\, dx + \int_{\Gamma_1^h} \sigma u_h v_h\, dx = \\ = \int_{\Omega_h} f v_h\, dx + \int_{\Gamma_1^h} \mu v_h\, dx. \quad (2.6)$$

2.1. Примеры триангуляций области

На левом рис. 1 приведен пример разбиения области на трехузловые конечные элементы. В этом случае элемент τ является одним

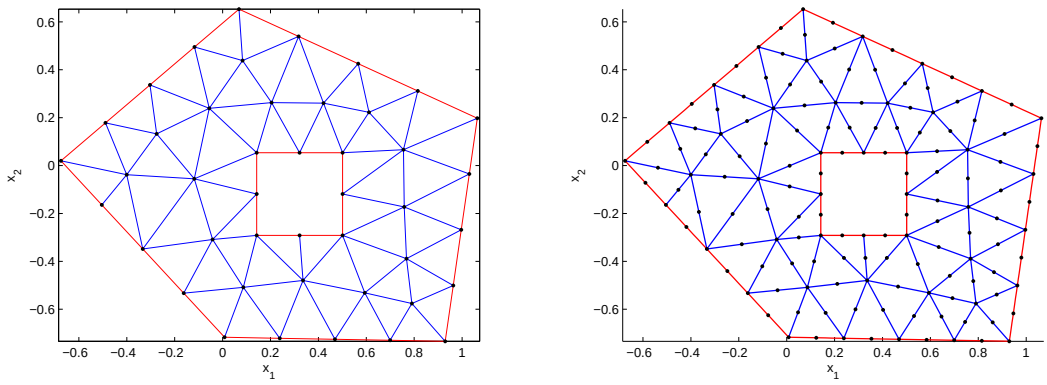

Рис. 1. Примеры разбиения двусвязной многоугольной области на конечные элементы треугольной формы. Узлы интерполяции выделены точками. Сетку узлов на левом рисунке назовем P_1 сеткой, на правом — P_2 сеткой.

из треугольников, ω_τ образуется его вершинами, $P_\tau = P_1 = \{p : p = c_1 + c_2x_1 + c_3x_2\}$ — множество полиномов степени не выше первого. Отметим, что любые два элемента могут иметь общими либо целиком сторону (ребро), либо вершину. На каждой стороне элемента располагаются два узла интерполяции.

На правом рис. 1 приведен пример разбиения области на шести узловые конечные элементы: в ω_τ включается кроме вершин треугольника τ также средние точки его сторон, $P_\tau = P_2 = \{p : p = c_1 + c_2x_1 + c_3x_2 + c_4x_1^2 + c_5x_1x_2 + c_6x_2^2\}$ — множество полиномов не выше второй степени; на каждой стороне элемента располагаются три узла интерполяции. В обоих случаях область Ω_h совпадает с Ω, произвольная функция из P_τ однозначно определяется своими значениями в точках ω_τ.

2.2. Система МКЭ

Тождество (2.6), 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 (N зависит от h) и каждому узлу $a_i \in \omega_h$ поставим в соответствие базисную функцию $\varphi_i \in S_h$ так, что $\varphi_i(a_j) =$

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

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

Разобьем множество индексов $I = \{1, 2, \ldots, N\}$ 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, $u_i = u_{D\,i}$; необходимо определить значения u_h в точках с индексами из i_n. В $\overline{\Omega}_h$ справедливы разложения

$$u_h(x) = \sum_{i \in i_n} u_i \varphi_i(x) + \sum_{i \in i_d} u_{D\,i} \varphi_i(x), \quad v_h(x) = \sum_{i \in i_n} v_i \varphi_i(x), \tag{2.7}$$

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

$$a_{ij} = \int_{\Omega_h} (c \nabla \varphi_j \cdot \nabla \varphi_i + b \cdot \nabla \varphi_j \varphi_i + a \varphi_j \varphi_i)\, dx + \int_{\Gamma_1^h} \sigma \varphi_j \varphi_i\, dx, \tag{2.8}$$

$$\phi_i = \int_{\Omega_h} f\, \varphi_i\, dx + \int_{\Gamma_1^h} \mu\, \varphi_i\, dx, \tag{2.9}$$

Теперь, подставляя в (2.6) вместо u_h его разложение (2.7), а вместо v_h — поочередно φ_i, $i \in i_n$, придем к системе уравнений[2)]

$$\sum_{j \in i_n} a_{ij} u_j = F_i, \quad i \in i_n, \quad F_i = \phi_i - \sum_{j \in i_d} a_{ij} u_{D\,j}. \tag{2.10}$$

Решая эту систему находим неизвестные u_j, $j \in i_n$, а приближенное решение нашей задачи (2.5) — по первой формуле в (2.7).

[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 $2h$.

[2)] система (2.10) эквивалентна тождеству (2.6). Чтобы убедиться в этом достаточно умножить i-тое равенство в (2.10) на v_i и просуммировать по всем $i \in i_n$ и учесть (2.7), (2.8) и (2.9).

Запишем систему (2.10) в матричном виде. Для этого упорядочим неизвестные по возрастанию индекса. 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 (2.10), очевидно, запишется в виде

$$K_0 u_0 = F_0. \tag{2.11}$$

Формулы (2.8), (2.9) дают некоторый способ вычисления элементов матрицы K_0 и вектора F_0, но ими непосредственно не пользуются при практических вычислениях, т. к. известен более удобный и более экономичный метод, который мы и рассмотрим. По традиции *K_0 называют глобальной матрицей жесткости, F_0 — глобальным вектором сил.*

УПРАЖНЕНИЯ.

Обратите внимание, что матрица K_0 и вектор F_0 определяются как уравнением, так и краевыми условиями.

1. Дайте определение K_0 и F_0 в случае задачи Дирихле для уравнения (2.1).

2. Дайте определение K_0 и F_0 в случае, когда на всей границе Γ задано однородное краевое условие Неймана $c\,\nabla u \cdot \nu = 0$.

3. Дайте определение K_0 и F_0 в случае, когда на всей границе Γ задано краевое условие третьего рода.

Указание. Определите V_h, V_h^0 и получите аналоги формул (2.7), (2.8) и (2.9) в каждом случае.

5. Докажите справедливость разложений (2.7).

§ 3. Алгоритм формирования системы МКЭ

Система алгебраических уравнений (2.11) определяется матрицей A и вектором Ф. Поэтому сначала рассмотрим принятый в МКЭ способ их вычисления.

3.1. Алгоритм вычисления матрицы A и вектора Φ

Из формулы (2.8) следует, что матрица A есть сумма двух квадратных матриц размерности $N \times N$: $A = K + H$,[1)]

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

Аналогично,

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

1. Вычисление K и F. Согласно определению

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

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

$$K\,u \cdot v = \sum_{\ell=1}^{n_t} \int_{\tau_\ell} (c\nabla u_h \cdot \nabla v_h + b \cdot \nabla u_h v_h + a u_h v_h)\, dx. \tag{2.12}$$

Рассмотрим представление u_h и v_h на элементе $\tau_\ell \in \mathcal{T}_h$. Пусть на τ_ℓ имеется m_τ узлов интерполяции 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

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

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

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

Подставляя эти разложения в (2.12), получим

$$K\,u\cdot v=\sum_{\ell=1}^{n_t}\sum_{\alpha,\beta=1}^{m_\tau}k_{\alpha\beta}^{\ell}u_{t_{\beta\ell}}v_{t_{\alpha\ell}}. \tag{2.13}$$

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

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

называется матрицей жесткости элемента τ_ℓ (локальной матрицей жесткости). Преобразуем равенство (2.13). Введем в рассмотрение матрицу $\tilde{K}^\ell$ размера $N\times N$, состоящую из нулей, за исключением m_τ^2 элементов, которые определим так:

$$\tilde{K}^{\ell}_{t_{\alpha\ell},t_{\beta\ell}}=k_{\alpha\beta}^{\ell},\quad \alpha,\beta=1,\ldots,m_\tau.$$

Тогда формула (2.13) примет вид

$$K\,u\cdot v=\sum_{\ell=1}^{n_t}\sum_{\alpha,\beta=1}^{m_\tau}\tilde{K}^{\ell}_{t_{\alpha\ell},t_{\beta\ell}}u_{t_{\beta\ell}}v_{t_{\alpha\ell}}=\sum_{\ell=1}^{n_t}\sum_{i,j=1}^{N}\tilde{K}_{ij}^{\ell}u_jv_i=$$

$$=\sum_{\ell=1}^{n_t}\tilde{K}^{\ell}y\cdot v=\Big(\sum_{\ell=1}^{n_t}\tilde{K}^{\ell}\Big)u\cdot v,$$

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 расширенных локальных матриц жесткости. Поскольку нули не имеет смысла суммировать, приходим к следующему алгоритму, известному как алгоритм сборки матрицы жесткости.

Алгоритм сборки матрицы жесткости.

- Положить $K = 0$ (K — матрица размера $N \times N$).
- Для каждого $\ell = 1, 2, \ldots, n_t$:
  - вычислить K^ℓ (размера $m_\tau \times m_\tau$).
  - для $\alpha, \beta = 1, \ldots, m_\tau$ 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.

- Положить $F = 0$ (F — вектор столбец длины N).
- 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

$$F^\ell_\alpha = \int_{\tau_\ell} f\, \varphi_{t_{\alpha\ell}}\, dx.$$

2. 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

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

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

$$h^k_{\alpha\beta} = \int_{e_k} \sigma\, \varphi_{e_{\beta k}} \varphi_{e_{\alpha k}}\, dx.$$

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

$$G^k_\alpha = \int_{e_k} \mu\, \varphi_{e_{\alpha k}}\, dx.$$

Алгоритм сборки матрицы H и вектора G определяется так же, как и выше. Они определяются краевым условием третьего рода и учитывают вклады от граней элементов, принадлежащих Γ^h_1, в матрицу A и вектор Φ соответственно.

УПРАЖНЕНИЯ.

1. Сформулируйте *алгоритм формирования системы МКЭ* в случаях, когда a) решается краевая задача Дирихле; b) решается однородная краевая задача Неймана.

2. Пусть задана разреженная (sparse) матрица K размера $N \times N$. Приведите MatLab код нескольких различных алгоритмов, которые позволяют получить из K матрицу K_0 вычеркиванием строк и столбцов с номерами $i = [i_1, i_2, \ldots, i_k]$.

3.2. Алгоритм решения задачи

На основе предыдущего, приходим к следующей последовательности действий для нахождения приближенного решения задачи (2.5):

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

2) определение коэффициентов и правых частей уравнения и 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 $K_0 u_0 = F_0$; образование вектора узловых параметров u решения u_h по u_0;

7) обработка решения u_h (например, построение графика u_h).

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

Задача построения подходящих для МКЭ триангуляций областей является непростой и самостоятельной задачей. Для ее решения мы воспользуемся функцией initmesh из *pde toolbox*. Следующая глава полностью посвящена этой проблеме.

3.3. Решение модельной задачи в *pde toolbox*

Набор программ для решения задач из некоторой предметной области в MatLab называются *toolbox*-ами. Они поставляются отдельно от MatLab. Для решения разнообразных краевых задач для уравнений в частных производных в двумерных областях (как стационарных, так и нестационарных), имеется *pde toolbox*. Он включает графическое приложение *pdetool*, используя который пользователь в удобной форме может решать конкретные краевые задачи. Все функции *pdetool* доступны также для вызова из программы пользователя, так что их можно использовать при решении самых разнообразных задач. В частности, *pde toolbox* позволяет решать рассматриваемую нами краевую задачу в случае, когда вектор-функция $b \equiv 0$.

Рассмотрим кратко, как, используя функции *pde toolbox*, можно решить следующую краевую задачу в единичном круге Ω:

$$-\triangle u(x) = 1, \quad x \in \Omega, \quad u(x) = 0, \quad x \in \Gamma. \tag{2.14}$$

Она соответствует рассматриваемой нами задаче (2.1), если принять $c \equiv 1$, $a = b = u_D \equiv 0$, $f \equiv 1$, $\Gamma_1 = \emptyset$ и имеет точное решение

$$u(x) = -(x_1^2 + x_2^2 - 1)/4.$$

В *pde toolbox* используются простейшие трехузловые конечные элементы (P_1 элементы). Шаги 4–6, отмеченные в предыдущем пункте, реализованы в функции assempde (далее в комментариях мы отмечаем номер шага, к которому относится соответствующий оператор). Результаты выполнения представленных ниже команд приведены на рис. 2. Из результатов вычислений следует, что максимальная погрешность приближенного решения в равномерной норме не превосходит $5 \cdot 10^{-4}$.

```
g='circleg';       % 1) set domain geometry
bc='circleb1';     % 2) set the boundary conditions
c=1; a=0; f=1;     % 2) set pde coefficients

[p,e,t]=initmesh(g,'Hmax',0.1); % 3) 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);    % 4-6) assembles and solves the FEM system

figure;                         % 7) 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;                         % 7) 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 реализовать в MatLab для рассматриваемой нами задачи $\mathcal{P}$. Кроме того, в редких случаях, наиболее трудных для реализации, мы будем пользоваться программами (функциями) *pde toolbox*.

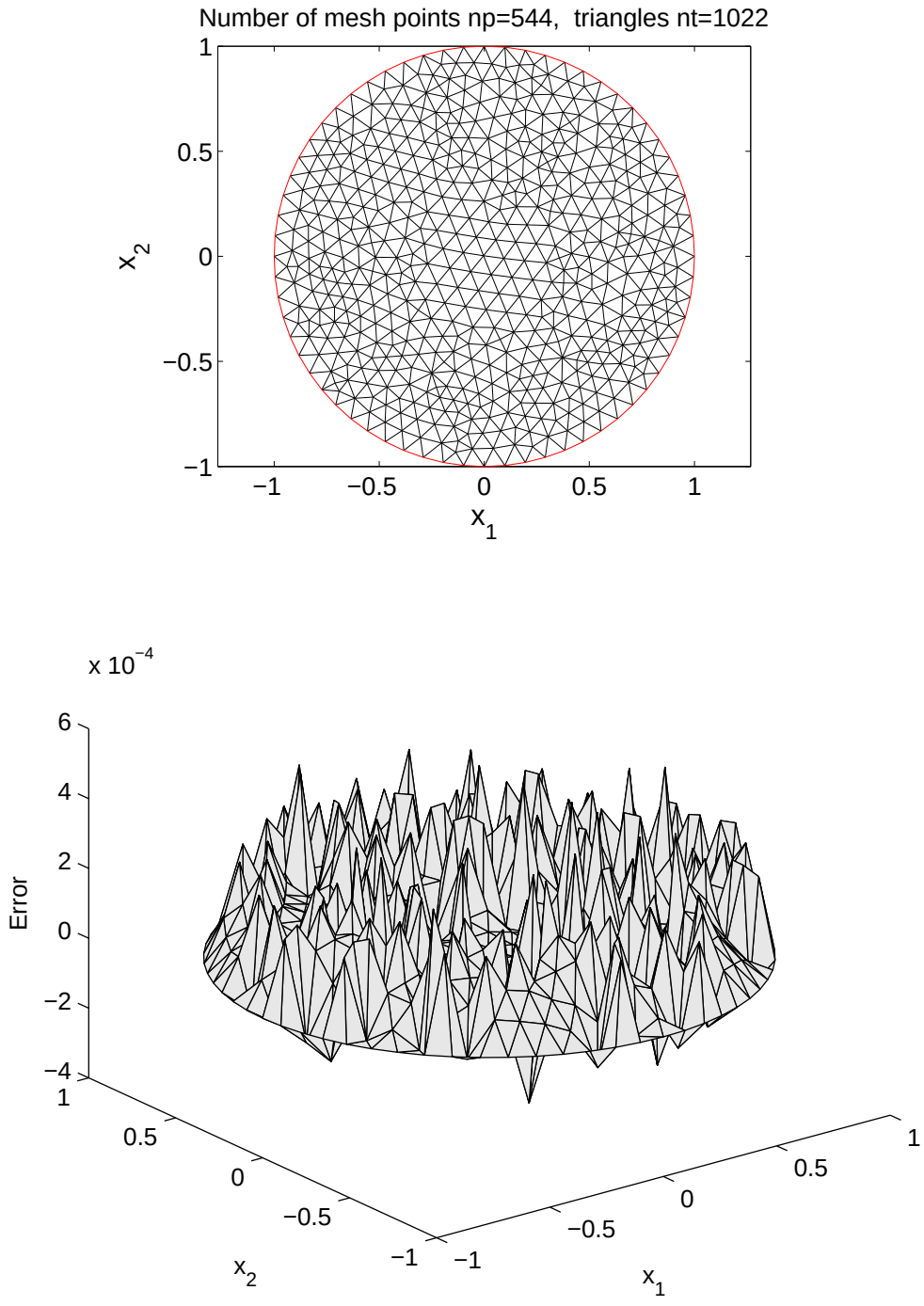

Рис. 2. P_1 триангуляция области и погрешность решения задачи (2.14) методом конечных элементов.

Глава 3

ПОСТРОЕНИЕ ДВУМЕРНЫХ СЕТОК В MATLAB

§ 1. 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.

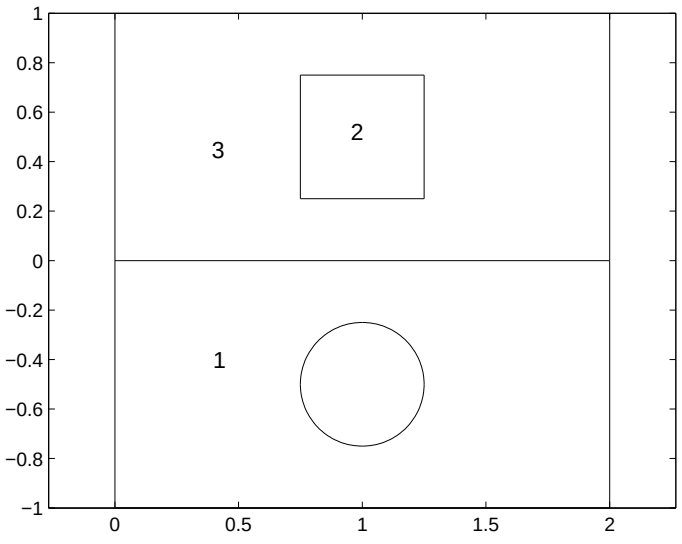

Рис. 1. Геометрия исходной области.

Для примера возьмем область, изображенную на рис. 1: она состоит из трех подобластей, отмеченных на рисунке цифрами 1, 2, 3

(метками). [1] 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). Различают внутренние сегменты (на рисунке 1 это сегменты прямоугольника с номером 2) и граничные (образующие границу всей области).[2] Сегменты не могут пересекаться или быть замкнутыми; они могут иметь лишь общие граничные точки. 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 три 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.

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.

Первый способ задания геометрии. Следующая матрица, формируемая функцией gtm, определяет геометрию области для нашего

[1] Предполагается, что в этих подобластях коэффициенты дифференциального уравнения определяются разными формулами.

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

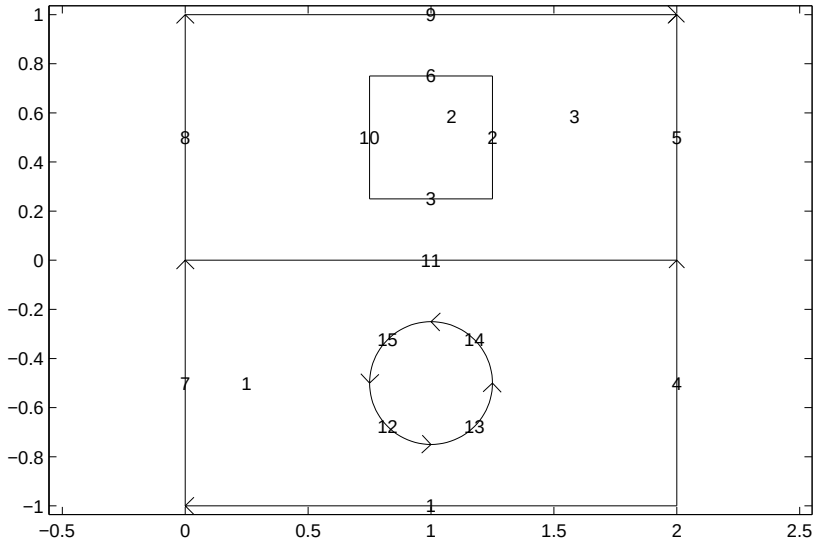

Рис. 2. Определение граничных сегментов и их номеров.

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); вторая и третья (четвертая и пятая) строки определяют x (y) координаты начальной и конечной точки соответственно; шестая и седьмая 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 ];
```

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

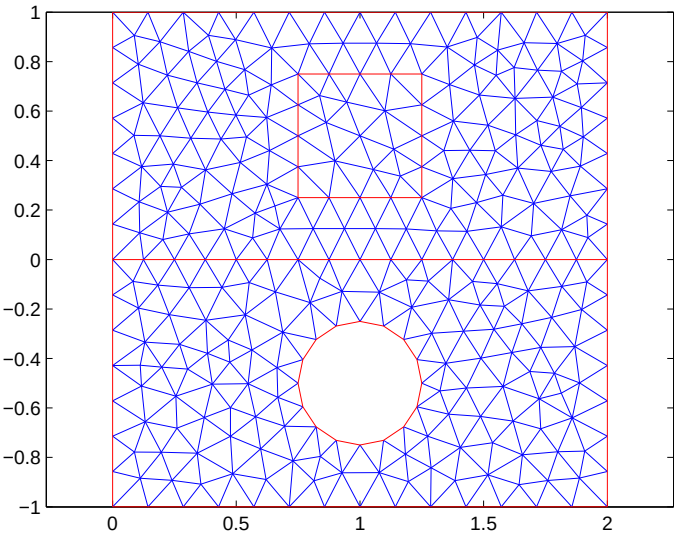

Рис. 3. Пример триангуляции области.

Следующие команды строят сетку, представленную на рис. 3:

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

Второй способ задания геометрии. К рис. 3 приводят также команды

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

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 их 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: первая и вторая (третья и четвертая) 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. Для построения сеток наибо-

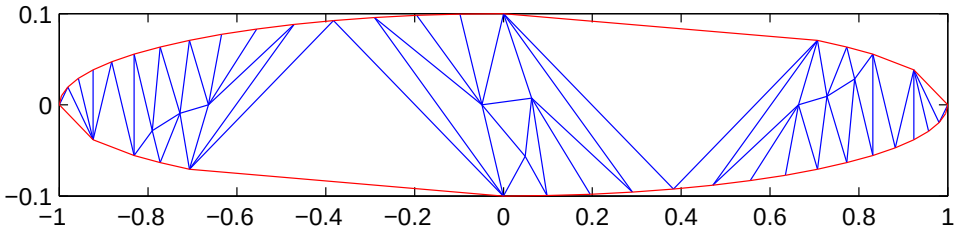

Рис. 4. Неудачная параметризация граничных сегментов.

лее 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

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
```

Видно, что сетка весьма некачественная; это объясняется тем, что функция initmesh не может распределить равномерно узлы на границе области из-за неудачной угловой параметризации, использованной

нами. 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

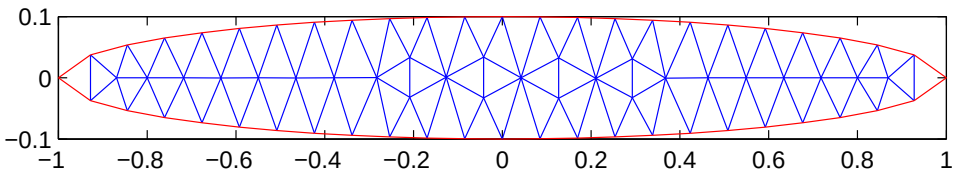

Рис. 5. Неудачная параметризация исправляется функцией pdearcl.

gell1 отличается от gell0 тем, что сначала выбираются 100 точек (xt,yt) на сегменте, которые следующей командой равномерно распределяются на нем по длине дуги (естественно, можно брать меньше

или больше 100 точек, в зависимости от длины сегмента). Функция pdearcl является функцией *pde toolbox*.

§ 2. Кодировка P_1 сеток

Рассмотрим геометрию области, изображенной слева на рис. 6, а также треугольную сетку на ней (рис. справа). Файл геометрии

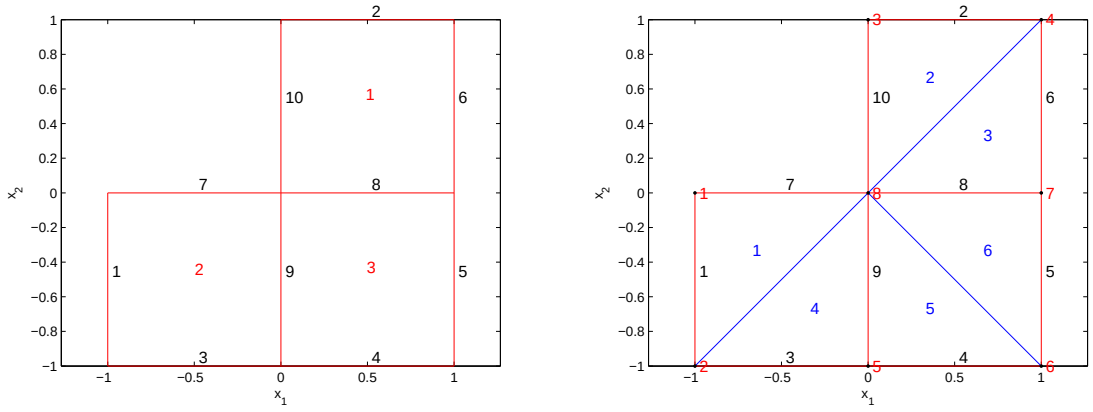

Рис. 6. L-образная область (слева) и его грубая триангуляция (справа).

'lshapeg' этой области находится в *pde toolbox*. На левом рисунке указаны 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); plotP1mesh(p,e,t,[1,1,1],12);
```

Текст функции plotP1mesh приведен ниже.

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

- в матрице 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 матрице e, размером $7 \times ne$, хранится 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 $e(6 : 7, k)$ —

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

номера подобластей слева и справа от ребра, если следовать параметризации. Напомним, что дополнение ко всей области имеет номер 0;

- в матрице 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      t =[1  4  7  2  5  6
    0 -1  1  1 -1 -1  0  0];         2  3  4  5  6  7
                                     8  8  8  8  8  8
                                     2  1  1  2  3  3];
```

```
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 plotP1mesh, в которой использованы MatLab-функции plot и text. Функция

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

выводит на текущие оси текстовую строку abc так, что первая буква строки имеет координаты x, y (остальные параметры необязательны).

```
function h=plotP1mesh(p,e,t,opt,fs)
% PLOTP1MESH: 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 в двумерных областях. Например, 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 рассматривается другой способ кодировки P_1 сеток (§ 3), а также P_2 сетки.

§ 3. Сопряженная кодировка сетки

Материал данного параграфа далее непосредственно не используется, однако является полезным при решении разнообразных задач, связанных с сетками.

Рассмотренная ранее кодировка сетки, представленная массивами (p, e, t), ориентирована на алгоритмы, в которых конечные элементы (треугольники, граничные ребра) обрабатываются независимо друг от друга. Однако, имея только матрицы (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?

Для быстрого решения подобных задач необходимо дополнить данные (p, e, t) другими.

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

Рассмотрим первую 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). Нетрудно 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 T' равны 1), то сможем легко решить первую 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 втором пункте* и аналогичных им, введем два дополнительных массива ee и te. Определим их следующим образом:

- в матрице ee, размером $7 \times K$, хранится 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; в $e(6:7,k)$ — 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 матрице 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);
```

Функция рисования имеет следующий вид:

```
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);
```

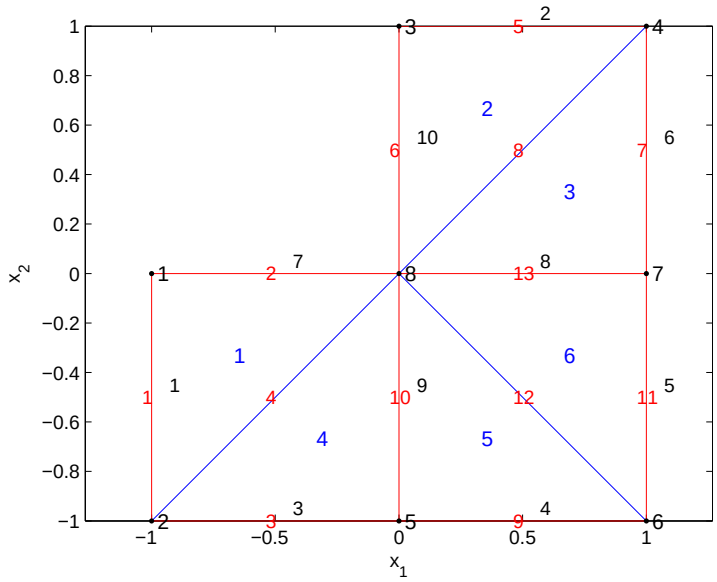

Рис. 7. Нумерация узлов, ребер и элементов (сопряженная P_1 сетка).

```
    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).

Глава 4

ПРОГРАММИРОВАНИЕ МКЭ НА ОСНОВЕ P_1 ЭЛЕМЕНТОВ

§ 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 плотных и разреженных матриц могут различаться.

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 заполнение 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 векторе $i\big(p(j) : p(j+1)-1\big)$, а их значения в векторе $a\big(p(j) : p(j+1)-1\big)$; 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 второй способ создания разреженных матриц, учитывая внутреннее хранение A. Рассмотрим команды

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

Команда 1) определяет матрицу A размера $n \times n$, 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 векторов (i, p, a) копируются, вставляется новый элемент и т. д. Этот способ, как видно, оказывается дорогим, если $nnz(A)$ — число ненулевых элементов A велико.

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

Очевидный метод реализации третьего способа основывается на упорядочении триплета (i, j, a) по возрастанию индекса i, суммирования копий, и т.д. Наибольшую сложность здесь представляет процедура сортировки целочисленного массива, сложность остальных шагов порядка $nnz(A)$.

§ 2. Программирование рассылки элементов

2.1. Рассылка элементов локальных матриц жесткости.

Для примера рассмотрим сборку глобальной матрицы жесткости для линейных трехузловых конечных элементов для конечно-элементной области, образованной разбиением сторон квадрата на n_x частей ($2n_x^2$ треугольников). 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 векторе $t(1:3, it)$ хранятся глобальные номера узлов на элементе с номером it.

Первый алгоритм (bad1), очевидный, 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
```

Второй алгоритм (bad2), векторизованная версия предыдущего. Подходит для небольших сеток.

```
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
```

Третьий 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.

Четвертый 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 ((i,j) элемент локальной матрицы жесткости вычисляется сразу для всех элементов). 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 можно 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 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 square 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 $A = locsm(it, p, t)$.

```
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(nt,dofe,dofe);  % 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
```

2.2. 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.

§ 3. 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.

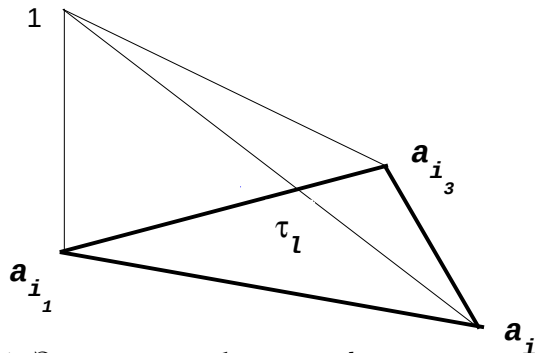

Рис. 1. Элемент τ_ℓ и базисная функция φ_{i_1}.

3.1. 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

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

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 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 первом 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:

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

где $|\tau_\ell|$ есть площадь τ_ℓ.

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

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

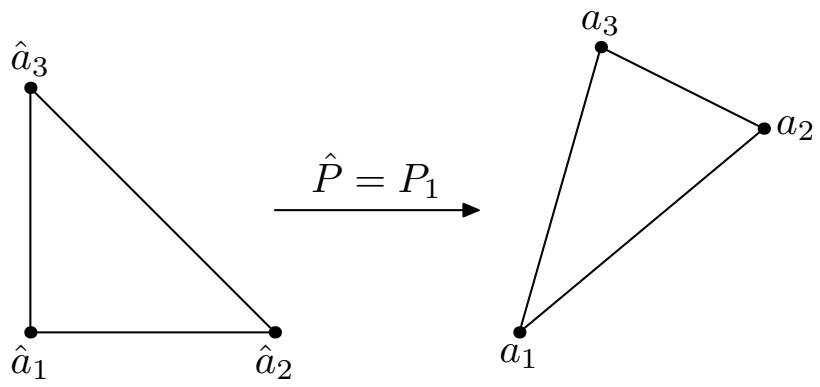

Рис. 2. Базисный элемент Δ (слева) и P_1 элемент τ (справа).

Для получения формул для базисных функций воспользуемся следующим приемом, пригодным и для более сложных треугольных элементов с прямолинейными сторонами (например, для 6-ти узловых P_2 элементов). Идея заключается в том, чтобы рассмотреть подобный τ_ℓ «канонический» (базисный) элемент Δ, 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 трехузловой 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

$$x = a_1\hat{\varphi}_1 + a_2\hat{\varphi}_2 + a_3\hat{\varphi}_3 = a_1 + \hat{x}_1(a_2 - a_1) + \hat{x}_2(a_3 - a_1) \qquad (4.1)$$

задает преобразование элемента Δ на τ. Действительно, это отображение является аффинным и вершины Δ 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

$$x = B_\tau\hat{x} + a_1 \ : \quad \Delta \to \tau. \qquad (4.2)$$

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

$$|\tau| = \int_\tau d\,x = \int_\Delta \det(B_\tau) d\,\hat{x} = |\Delta| \det(B_\tau) = \frac{1}{2} J_\tau. \tag{4.3}$$

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

$$B_\tau^{-1} = \frac{1}{\det(B_\tau)} \begin{pmatrix} (a_3 - a_1)_2 & -(a_3 - a_1)_1 \\ -(a_2 - a_1)_2 & (a_2 - a_1)_1 \end{pmatrix} =: \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}. \tag{4.4}$$

Следовательно,

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

Базисные функция $\varphi_i(x)$ на элементе τ получаются заменой переменных из базисных функций на элементе Δ 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 равны

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

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

$$\nabla\varphi_i(x) = B_\tau^{-T} \hat{\nabla}\hat{\varphi}_i(\hat{x}), \quad \hat{x} = B_\tau^{-1}(x - a_1), \tag{4.5}$$

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

Заметим также, что для элементов τ любых типов

$$\sum_{i=1}^{m_\tau} \varphi_i(x) = 1, \quad \sum_{i=1}^{m_\tau} \nabla\varphi_i(x) = (0, 0)^T, \quad x \in \tau, \tag{4.6}$$

[1)]На базисном элементе это так, а точка $(1/3, 1/3)$ — центр тяжести элемента Δ, при преобразовании координат переходит в x_τ — центр тяжести τ.

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

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

Матрица M^τ с этими элементами называется матрицей масс. 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»

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

а матрица M^τ является положительно определенной.

Аналогично вычисляются компоненты вектора сил элемента τ

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

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

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

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 первом случае

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

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 $\varphi_{i_\alpha}(x_e)=1/2$.

3.2. Способы задания коэффициентов уравнения

Матрицы жесткости и векторы сил элементов зависят соответственно от коэффициентов и правой части дифференциального уравнения c, b, a, f. Прежде чем программировать сборку матрицы жесткости, необходимо решить в каком виде задавать 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 [1)];

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);

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

Поясним эти способы. Напомним, что область Ω может состоять из ряда подобластей, в каждой из которых коэффициенты уравнения могут иметь свой вид. Подобласти предполагаются пронумерованными и с каждой из меток (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.

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

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

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

3) предполагается, что имеется три подобласти; в первой подобласти c постоянна и равна 1; во второй подобласти $c = x_1^2 + x_2$, в третьей подобласти $c = \sin(x_1 + x_2)$:

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

Длинные строки удобно формировать следующим образом: создаем $c1 =' \ldots'$; $c2 =' \ldots'$; $\ldots$, $cm =' \ldots'$ и определяем $c = [\ '''' \ c1 \ '!' \ c2 \ '!' \ \ldots \ '!' \ cm \ '''' \ ]$, отделяя ck-тые подстроки пробелами слева и справа.

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.

3.3. Вклад элементов в систему МКЭ.

В разделе 3.1, стр. 87, 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 (по числу элементов) достаточно убрать цикл по it, заменить скалярные операции умножения и деления на векторные .* и ./, 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 трех подобластей, изображенную на рис. 1 (см. §1, с. 64). Его геометрия определяется матрицей *gtm*. Замерим время выполнения на трех 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;                   % pointer to 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    0.00996      1.408
  2652    5112    0.0372       5.592
 16459   32438    0.283        64.968
```

Несмотря на невысокую точность замера времени выполнения по CPU, можно уверенно говорить о пользе векторизации при программировании в MatLab, хотя, как мы видели ранее, сравниваемые функции имеют примерно одинаковые времена рассылки элементов при сборке.

3.4. Учет краевых условий. Формирование системы МКЭ

1. 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 третьего 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 задаваться 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 3.2, с. 91, 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., $c = \mathrm{const}$. 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 втором — $u_D = x+y$, 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 третьего 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.

2. 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. (1.24), с. 17):

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

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, все элементы которого равны нулю, кроме элементов с номерами $j \in i_d$, $u_D(j) = u_{D\,j}$.

Матрица K_0 получается из матрицы A вычеркиванием его строк и столбцов с номерами из множества i_d. Чтобы реализовать эту опе-

рацию, удобно ввести матрицу N размера $n_p \times m$, $m = numel(i_n)$, которую определим 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 также, что после решения системы $K_0 u_0 = F_0$, вектор узловых значений решения u_h схемы МКЭ находится по формуле $u = Nu_0 + u_D$ (вектор u_0 дополняется 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.

3. Формирование и решение системы МКЭ. На основании приведенных выше функций можно написать различные 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
```

3.5. Решение тестовой задачи

Применим функцию *assembpde* для решения тестовой задачи. Рассмотрим краевую задачу 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 функции в уравнении 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 четырех 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=[];

exu=@(x,y,sdl) x.^2+y.^2;   % exact solution
for h=[0.5 0.1 0.05 0.02]
  [p,e,t]=initmesh(g,'hmax',h);

  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];
```

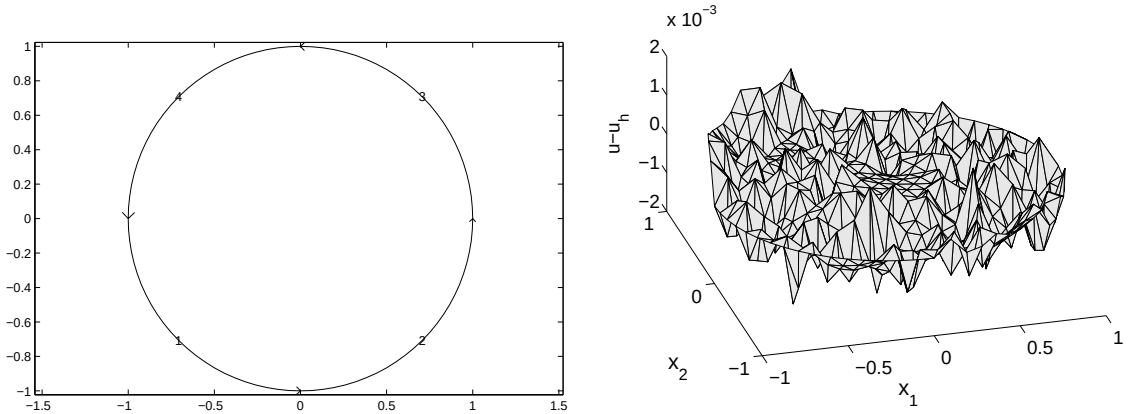

Рис. 3. Геометрия Ω (слева) и график погрешности решения (справа) на сетке с максимальным размером элементов $h = 0.1$ ($np = 544$).

```
  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:

```
 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*.

УПРАЖНЕНИЕ.

1. Напишите функцию, которая определяет зависимость погрешности решения схемы МКЭ в нормах C, L_2 и H_0^1 от шага сетки h и от N в предположении, что точное решение известно и определяется либо неименованной, либо m-функцией.

Указание. Решите тестовую задачу на последовательности из k сеток с шагами $h = [h_1, h_2, \ldots, h_k]$ (с числом узлов $n = [n_1, n_2, \ldots, n_k]$) и вычислите функционалы погрешности $e = [e_1, e_2, \ldots, e_k]$ (например, в норме L_2). Если зависимость погрешности от h степенная, т. е. $e_i = c\,h_i^\alpha$ при некотором α, то построив график функции $e = e(h)$ в двойных логарифмических координатах командой loglog(h,e,'-b.') вы должны увидеть фактически график линейной функции, наклон которой и есть α (почему?). Найдите формулу для вычисления α. Аналогично рассмотрите зависимость $e = e(n)$.

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, 2011. — 237 с.