Содержание

Введение

Расчетное задание посвящено решению краевой задачи в частных производных методом конечных элементов. Основной акцент в данном пособии делается на конечно-элементную аппроксимацию и итерационные методы решения системы линейный алгебраических уравнений (СЛАУ). В качестве основного итерационного метода предлагается использовать предобусловленный метод сопряженных градиентов, а в качестве предобусловливателей к нему – многосеточный метод и метод неполной факторизации.

Цель работы – дать студентам представление о современных методах решения задач в частных производных, и выработать навыки их практической реализации.

При выполнении задания студенты должны реализовать предложенные численные методы на языках Фортран или Си. Отчет по выполненному заданию должен включать в себя:

1) постановку задачи;
2) краткое описание теории и алгоритмов;
3) результаты тестовых расчетов и численных экспериментов;
4) выводы о применимости реализованных методов для решения предложенной задачи.

§1 Постановка краевой задачи

Рассмотрим задачу Дирихле

$$-\nabla\cdot\left(a(x,y)\nabla u\right)=f(x,y),\quad (x,y)^T\in\Omega\subset R^2, \tag{1.1}$$

$$u\big|_{\partial\Omega}=g(x,y), \tag{1.2}$$

которая описывает распределение скалярной физической величины во многих областях физики и механики, например, стационарное поле температуры, электростатическое поле, поле давления в тонком зазоре. В качестве расчетной области возьмем единичный квадрат $\Omega=[0,1]\times[0,1]$.

В первую очередь определимся с тем, в каких именно пространствах будем искать решение этой задачи.

Коэффициенты уравнения (1.1) во множестве практических задач являются разрывными функциями. Так, например, в некоторых задачах теплопроводности значения коэффициента $a(x,y)$ в разных подобластях могут различаться на несколько порядков. А при решении задач по определению поля давления в тонких зазорах (типично для задач газовой смазки) правая часть может определяться как дивергенция разрывного векторного поля $\mathbf{U}$: $f(x,y)=\nabla\cdot(\mathbf{U})$. В этих случаях в классическом смысле решения задачи (1.1), (1.2) существовать не будет, то есть $u\notin C^2(\Omega)$. Поэтому искать решение будем в обобщенном смысле – в пространстве Соболева $H^1(\Omega)$.

Обобщенным решением задачи (1.1), (1.2) называется функция $u\in H^1(\Omega)$, удовлетворяющая равенству

$$\int_\Omega a(x,y)\nabla u\cdot\nabla v\,dxdy=\int_\Omega f(x,y)v\,dxdy \quad \text{при любой } v\in H_0^1(\Omega), \tag{1.3}$$

и след которой на $\partial\Omega$ равен $g(x,y)$.

Нетрудно проверить, что для множества практических задач выполняются условия: $a(x,y)\in L_\infty(\Omega)$, $f(x,y)\in H^{-1}(\Omega)$, $g(x,y)\in H^{1/2}(\partial\Omega)$. А значит, согласно известной теории [5, С.118-122] для таких задач обобщенное решение существует и единственное.

§2 Метод конечных элементов

Естественным способом решение задачи (1.3) является метод конечных элементов. Суть этого метода заключается в том, что решение ищется не в пространстве Соболева $H^1(\Omega)$, а в его конечномерном подпространстве $V^h\subset H^1(\Omega)$, которое определяется при помощи базисных функций, задаваемых на ячейках расчетной сетки. Конечным элементом при этом называется сово-

купность геометрических характеристик ячейки и заданных на ней базисных функций.

Приближенное решение по методу конечных элементов определяется с помощью подхода Галеркина как функция $u^h \in V^h$, удовлетворяющая равенству

$$\int_\Omega a(x,y)\nabla u^h \cdot \nabla v^h \, dxdy = \int_\Omega f(x,y)v^h \, dxdy \quad \text{при любой } v^h \in V_0^h \subset H_0^1(\Omega), \qquad (2.1)$$

значение которой на $\partial\Omega$ равно $g(x,y)$; далее функции v^h будем называть весовыми.

В настоящей работе будем рассматривать только кусочно-линейную аппроксимацию на треугольных сетках. Для этого проведем триангуляцию T^h исходной области Ω (рис. 8.1), на которой введем пространство кусочно-линейных функций V^h (линейных в треугольниках $T \in T^h$ и непрерывных в области $\Omega_h \equiv \Omega = \bigcup_{T \in T^h} \overline{T}$). Буква h означается характерный размер треугольников (на рис. 8.1 h – длина катета прямоугольного треугольника).

В качестве базисных функций v_i^h для кусочно-линейного пространства V^h удобно брать функции, значение которых равно 1 в i-ом узле сетки и 0 во всех остальных узлах:

$$v_i^h(x_j, y_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}. \qquad (2.2)$$

Формула для этой функции будет приведена ниже (2.8).

После выбора пространства V^h и его базисных функций приближенное решение запишется в виде $u^h = \sum_{j=1}^{n} u_j v_j^h$, где v_j^h – базисные функции пространства V^h, а u_j – значения приближенного решения в узлах сетки, которые требуется найти. Далее, подставляя это выражение в (2.1) и взяв в качестве весовой функции v_i^h, получим

$$\sum_{j=1}^{n}\left(\int_{\Omega} a(x,y)\nabla v_j^h \cdot \nabla v_i^h \, dxdy\right) u_j = \int_{\Omega} f(x,y) v_i^h dxdy \text{, для всех } v_i^h \in V_0^h . \qquad (2.3)$$

В итоге, с учетом граничных условий, задача сводится к СЛАУ

$$Au = F , \qquad (2.4)$$

где

$$A_{i,j} = \int_{\Omega} a(x,y)\nabla v_j \cdot \nabla v_i \, dxdy , \; F_i = \int_{\Omega} f(x,y) v_i \, dxdy \qquad (2.5)$$

для внутренних i-ый узлов и $A_{i,j} = \delta_{i,j}$ ($\delta_{i,j}$ – символ Кронекера), $F_i = g(x_i, y_i)$ – для граничных. Эта система аппроксимирует дифференциальное уравнение с точностью $O(h^2)$.

При вычислении интегралов (2.5) функций $a(x)$ и $f(x)$ можно заменить следующими приближениями в пространстве V^h:

$$a^h(x,y) = \sum_{k=1}^{n} a_k v_k^h , \;\; f^h(x,y) = \sum_{k=1}^{n} f_k v_k^h , \qquad (2.6)$$

где $a_k = a(x_k, y_k)$, $f_k = f(x_k, y_k)$. Кроме того, интеграл по области Ω можно представить в виде суммы интегралов по всем треугольникам сетки. А значит, интерес, прежде всего, вызывает нахождение соответствующих интегралов на каждом треугольнике в отдельности.

Рассмотрим треугольник с локальной нумерацией узлов против часовой стрелки $T = \left\{(x_1, y_1)^T, (x_2, y_2)^T, (x_3, y_3)^T\right\}$. Площадь такого треугольника определяется по формуле

$$S = \frac{1}{2}(x_2 y_3 - x_3 y_2 + x_1 y_2 - x_2 y_1 + x_3 y_1 - x_1 y_3). \qquad (2.7)$$

Базисная функция, удовлетворяющая свойству (2.2), запишется в виде

$$v_k^h = \frac{1}{2S}(\alpha_k x + \beta_k y + \gamma_k), \qquad (2.8)$$

где

$$\alpha_1 = y_2 - y_3,\ \beta_1 = x_3 - x_2,\ \gamma_1 = x_2 y_3 - x_3 y_2,$$
$$\alpha_2 = y_3 - y_1,\ \beta_2 = x_1 - x_3,\ \gamma_2 = x_3 y_1 - x_1 y_3, \qquad (2.9)$$
$$\alpha_3 = y_1 - y_2,\ \beta_3 = x_2 - x_1,\ \gamma_3 = x_1 y_2 - x_2 y_1.$$

Подставляя (2.8) в интегралы (2.5) по одному треугольнику, получим следующие вычислительные формулы

$$A_{i,j}^{(T)} = \int_T a^h(x,y)\nabla v_j^h \cdot \nabla v_i^h \, dxdy = \frac{a_1 + a_2 + a_3}{3} \frac{\alpha_i \alpha_j + \beta_i \beta_j}{4S}, \qquad (2.10)$$

$$F_i^{(T)} = \int_T f^h(x,y) v_i \, dxdy = \sum_{j=1}^{3} f_j \int_T v_j^h \, v_i^h \, dxdy, \text{ где } \int_T v_j^h \, v_i^h \, dxdy = \begin{cases} S/6, & i = j \\ S/12, & i \neq j \end{cases}. \qquad (2.11)$$

Эти формулы определяют матрицу $A^{(T)}$ и вектор правой части $F^{(T)}$ элемента (треугольного в нашем случае).

Для получения матрицы A и правой части F системы (2.4) интеграл нужно взять по всей области Ω. В силу выбора базисных функций СЛАУ можно собрать с помощью конечно-элементной сборки из матриц $A^{(T)}$ и правых частей $F^{(T)}$ элементов. Это осуществляется последовательностью следующих алгоритмов.

Алгоритм 2.1. Конечно-элементная сборка СЛАУ

(1) *For* $T = 1..N_T$ *Do* // N_T *– число треугольников сетки*

(2) *Вычисляем матрицу* $A^{(T)}$ *и вектор правой части* $F^{(T)}$ *на треугольнике*

(3) *For* $i = 1..3$ *Do*

(4) $I = get\_global\_node\_index\_by\_local(T,\ i)$

(5) *For* $j = 1..3$ *Do*

(6) $J = get\_global\_node\_index\_by\_local(T,\ j)$

(7) $A(I,J) = A(I,J) + A^{(T)}(i,j)$

(8) *EndDo*

(9) $F(I) = F(I) + F^{(T)}(i)$

(10) EndDo

(11) EndDo

Фактически, этот алгоритм собирает систему из уравнений вида

$$\sum_{j=1}^{n}\left(\int_{\Omega} a^h(x,y)\nabla v_j^h \cdot \nabla v_i^h \, dxdy\right) u_j = \int_{\Omega} f^h(x,y) v_i^h dxdy \text{ для всех } v_i^h \in V^h, \quad (2.12)$$

которая отличается от (2.3) прежде всего тем, что весовые базисные функции берутся не из подпространства V_0^h, а из всего пространства V^h. То есть полученная система не учитывает граничные условия (1.2). Чтобы их учесть, требуется применить следующий алгоритм.

Алгоритм 2.2. Учет граничный условий первого рода

(1) $For\ \ i = 1..n\ \ Do$

(2) $If\ \ node\ i\ \ belong\ to\ boundary\ \ Then$

(3) $A_{i,j} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$ *для всех* $j = 1..n;\quad F_i = g(x_i, y_i);$ *continue for next i*

(4) EndIf

(5) $For\ \ j = 1..n\ \ Do$

(6) $If\ \ node\ j\ \ belong\ to\ boundary\ \ Then$

(7) $F_i = F_i - A_{i,j} g(x_j, y_j);\quad A_{i,j} = 0$

(8) EndIf

(9) EndDo

(10) EndDo

Определившись с алгоритмами сборки СЛАУ, рассмотрим ряд ее свойств.

Легко заметить, что в силу выбора базисных функций элемент матрицы $A_{i,j}$ будет не равен 0 только в тех случаях, когда i-ый и j-ый узлы сетки принадлежат одному ребру треугольника. А значит, полученная матрица является разреженной, и для решения системы (2.4) целесообразно использовать итерационные методы. При этом известно, что скорость сходимости итерационных

методов сильно зависит от числа обусловленности системы $\mathrm{Cond}(A) \overset{def}{=} \|A\|\|A^{-1}\|$, которое, в свою очередь, линейно растет с увеличением количества узлов сетки. Поэтому на практике решают не исходную, а предобусловленную (в данном случае слева) систему

$$BAu = BF, \tag{2.13}$$

где оператор предобусловливания B строят таким образом, чтобы число обусловленности $\mathrm{Cond}(BA)$ было бы как можно меньше и слабо зависело от количества узлов сетки.

§3 Метод сопряженных градиентов

В качестве основного итерационного метода решения СЛАУ выберем широко известный метод сопряженных градиентов (МСГ). Суть его заключается в следующем. Пусть имеется самосопряженный положительно определенный линейный оператор $A: V \to V$, заданный на конечномерном пространстве V размерности n. Исходя из теоремы Гамильтона-Кэли, его обратный оператор A^{-1} можно представить как полином от исходного оператора A степени $\le n-1$

$$A^{-1} = P_{n-1}(A), \quad \deg(P_{n-1}) \le n-1. \tag{3.1}$$

Отсюда следует, что решение уравнения

$$Au = f \tag{3.2}$$

можно записать в виде

$$u = A^{-1}f = P_{n-1}(A)f. \tag{3.3}$$

Или, другими словами, $u \in V_n$, где V_k обозначает пространство Крылова порядка k

$$V_k = \mathrm{Span}\left(f,\ Af,\ A^2 f, \ldots, A^{k-1} f \right). \tag{3.4}$$

На основе этого факта и строится метод сопряженных градиентов, суть которого заключается в построении последовательности приближений Галеркина исходной задачи (3.2) в пространствах V_k:

$$(Au_k, v) = (f, v), \quad \forall v \in V_k, \ k = 0,1,2,\ldots. \tag{3.5}$$

Заметим, что $u_n = u = A^{-1} f$. А это означает, что МСГ является точным методом. Однако на практике его применяют именно как итерационный, поскольку скорость сходимости процесса построения приближений u_k $(k = 0,1,\ldots)$ достаточно высока, и он может завершиться при $k << n$. Для k-того приближения справедлива оценка

$$\|u - u_k\|_A \le 2\left(\frac{\sqrt{\mathrm{cond}(A)} - 1}{\sqrt{\mathrm{cond}(A)} + 1}\right)^k \|u - u_0\|_A. \tag{3.6}$$

Однако помимо хорошей скорости сходимости метод должен быть эффективным и простым в применении, то есть на каждой итерации задача (3.5) должна быстро решаться. Для этого определим A-ортогональное дополнение пространства Крылова V_k до V_{k+1}

$$V_k^{\perp} = \{w \in V_{k+1} : \ (Aw, v) = 0, \ \forall v \in V_k\}. \tag{3.7}$$

Предположим, что u_k найдено, и рассмотрим невязку для k-го приближения $r_k = f - Au_k$. Если $r_k = 0$, то $u_k = u$, и задача (3.2) решена. В противном случае, если $r_k \neq 0$, согласно (3.5) получим, что $r_k \notin V_k$ и $r_k \in V_{k+1}$. Или другими словами, $V_k^{\perp} \neq 0$. Следовательно

$$V_{k+1} = V_k \oplus V_k^{\perp}. \tag{3.8}$$

В результате $(k+1)$-ое приближение может быть представлено в виде

$$u_{k+1} = u_k + \alpha_k p_k, \tag{3.9}$$

где $p_k \neq 0$, $p_k \in V_k^{\perp}$. При этом параметр α_k легко найти, используя простую подстановку выражения (3.9) в $(k+1)$-ое вариационное равенство (3.5):

$$\alpha_k = \frac{(r_k, p_k)}{(Ap_k, p_k)}. \tag{3.10}$$

В итоге остается найти ненулевой элемент $p_k \in V_k^{\perp}$. Для этого возьмем $p_0 = r_0$ и предположим, что $p_1, \ldots, p_{k-1}$ уже найдены. Очевидно, что

$$\left(A\,p_i, p_j\right) = 0 \quad \text{для } i \neq j\,. \tag{3.11}$$

То есть последовательность $\{p_0,\ldots,p_{k-1}\}$ является ортогональным базисом пространства V_k. В свою очередь $r_k \notin V_k$ и $r_k \in V_{k+1}$. А значит, r_k можно A-ортогонализировать до элемента $p_k \in V_k^{\perp}$ следующим образом

$$p_k = r_k - \sum_{i=1}^{k-1} \frac{\left(A\,p_i, r_k\right)}{\left(A\,p_i, p_i\right)} p_i\,. \tag{3.12}$$

Заметим, что $A\,p_i \in V_k$ для $i \le k-2$. А значит, исходя из (3.5), получим, что $\left(A\,p_i, r_k\right) = 0$ для $i \le k-2$. В результате (3.12) перепишется в виде

$$p_k = r_k - \frac{\left(A\,p_{k-1}, r_k\right)}{\left(A\,p_{k-1}, p_{k-1}\right)} p_{k-1}. \tag{3.13}$$

Сделав дополнительные преобразования, позволяющие сократить количество элементарных вычислительных операций, запишем окончательный вид МСГ.

Алгоритм 3.1. Метод Сопряженных Градиентов

(1) Задаем начальное приближение u_0. *Вычисляем* $p_0 = r_0 = f - Au_0$.

(2) Для $k = 0,1,\ldots$

$$\begin{aligned} u_{k+1} &= u_k + \alpha_k p_k\,, \\ r_{k+1} &= r_k - \alpha_k A\,p_k\,, \\ p_{k+1} &= r_{k+1} + \beta_k p_k\,, \end{aligned} \tag{3.14}$$

где

$$\alpha_k = \frac{\left(r_k, r_k\right)}{\left(A\,p_k, p_k\right)}, \quad \beta_k = \frac{\left(r_{k+1}, r_{k+1}\right)}{\left(r_k, r_k\right)}. \tag{3.15}$$

Поскольку скорость сходимости (3.6) МСГ сильно зависит от числа обусловленности оператора A, то, как отмечалось выше, зачастую вместо (3.2) решают предобусловленное уравнение $BAu = Bf$, где предобусловливающий оператор B является самосопряженным и положительно определенным. При этом оператор BA в общем случае не является самосопряженным относительно

исходного скалярного произведения, и поэтому напрямую использовать МСГ для предобусловленной системы нельзя. Чтобы преодолеть этот недостаток, вводится энергетическое скалярное произведение $[u,v]=(Au,v)$, относительно которого оператор BA является самосопряженным. Далее, в алгоритме 3.1 исходное скалярное произведение заменяется энергетическим, и получается МСГ для предобусловленной системы уравнений. Сделав некоторые преобразования, можно значительно упростить формулы метода, записав расчетные формулы относительно исходного, а не энергетического скалярного произведения.

Алгоритм 3.2. Предобусловленный Метод Сопряженных Градиентов

(1) Задаем начальное приближение u_0. *Вычисляем* $r_0 = f - Au_0$, $p_0 = Br_0$.

(2) Для $k = 0,1,\ldots$

$$\begin{aligned} u_{k+1} &= u_k + \alpha_k p_k, \\ r_{k+1} &= r_k - \alpha_k A p_k, \\ p_{k+1} &= B r_{k+1} + \beta_k p_k, \end{aligned} \tag{3.16}$$

где

$$\alpha_k = \frac{(Br_k, r_k)}{(Ap_k, p_k)}, \quad \beta_k = \frac{(Br_{k+1}, r_{k+1})}{(Br_k, r_k)}. \tag{3.17}$$

(3) Итерационный процесс (3.16) продолжается до тех пор, пока не выполнится условие $\eta^{k+1} < \varepsilon$, *где* η^{k+1} *вычисляется по одной из формул*

$$\eta^{k+1} = \frac{\left\| Br^{k+1} \right\|}{\|BA\| \left\| u^{k+1} \right\| + \|Bf\|}, \tag{3.18}$$

$$\eta^{k+1} = \frac{\left\| Br^{k+1} \right\|}{\|Bf\|}. \tag{3.19}$$

Однако при выполнении расчетного задания можно ограничиться более простым критерием выхода

$$\eta^{k+1} = \left\| Br^{k+1} \right\| < \varepsilon, \tag{3.20}$$

где в качестве нормы рекомендуется взять $\|x\| = \|x\|_\infty \equiv \max|x_i|$.

§4 Многосеточный предобусловливатель

Для построения многосеточного предобусловливателя обратимся к симметричному методу Гаусса-Зейделя. Известно, что сходимость этого метода сильно зависит от числа обусловленности, которое, в свою очередь, зависит от шага сетки (количества узлов).

Но даже при больших числах обусловленности метод Гаусса-Зейделя обладает одним замечательным свойством: всего за несколько итераций он подавляет высокочастотные составляющие ошибки (сглаживает приближение решения). На основе этого свойства и строятся многосеточные методы. В геометрическом многосеточном методе строится последовательность вложенных сеток. На самой мелкой (расчетной) сетке гасится высокочастотная составляющая ошибки, а средние и низкие частоты гасятся на последовательности более грубых вложенных сеток. И чем сетка грубее, тем более низкие частоты устраняется из ошибки.

Опишем процесс подробнее. Для этого введем набор вложенных сеток $T_1^h, T_2^h, \ldots, T_J^h \equiv T^h$ (пример двух вложенных треугольных сеток приведен на рис. 9.1). На этих сетках зададим конечномерные пространства

$$V_1^h \subset V_2^h \subset \ldots \subset V_J^h \equiv V^h, \tag{4.1}$$

на которых будем искать решение задачи

$$\int_\Omega a^h(x,y)\nabla u^h \cdot \nabla v^h \, dxdy = \int_\Omega f^h(x,y) v^h \, dxdy, \text{ для всех } v_i^h \in V^h \tag{4.2}$$

с учетом граничных условий. Введем обозначение

$$A\left(u^h, v^h\right) = \int_\Omega a^h(x,y)\nabla u^h \cdot \nabla v^h \, dxdy.$$

Поскольку билинейная форма $A\left(u^h, v^h\right)$ является непрерывной и эллиптичной, то существует такой оператор A^h, что $A\left(u^h, v^h\right) = \left(A^h u^h, v^h\right)$ для любых u^h, v^h. В результате (4.2) перепишется в виде $\left(A^h u^h, v^h\right) = \left(f^h, v^h\right)$ или

$$A^h u^h = f^h. \tag{4.3}$$

Для нахождения решений на наборе пространств (4.1) определим оператор проектирования $Q_k^h : V^h \to V_k^h$:

$$\left(Q_k^h u^h, v_k^h\right) = \left(u^h, v_k^h\right), \qquad \forall v_k^h \in V_k^h \tag{4.4}$$

и $A_k^h : V_k^h \to V_k^h$ – операторы сужения A^h на V_h^k:

$$\left(A_k^h u_k^h, v_k^h\right) = A\left(u_k^h, v_k^h\right), \quad \forall u_k^h, v_k^h \in V_k^h. \tag{4.5}$$

При этом будем полагать, что выполнено следующее условие

$$\left\| \left(I - Q_{k-1}^h\right) v^h \right\|^2 \le \mathrm{Const} \cdot \rho\left(A_k^h\right)^{-1} A\left(v^h, v^h\right), \quad \forall v^h \in V^h, \tag{4.6}$$

которое определяет качество операторов проектирования. Если константа в условии будет слишком большой, то набор пространств (4.1) окажется непригодным для использования в многосеточном алгоритме.

Введя необходимые обозначения, можно записать многосеточный алгоритм для операторного уравнения (4.3). Суть его заключается в том, что на мелкой сетке производится несколько сглаживающих итераций, после чего полученная невязка проектируется на следующий грубый уровень, где процесс рекуррентно повторяется. После коррекции на грубом уровне к решению (на мелкой сетке) добавляется поправка с грубой сетки и делается еще несколько сглаживающих итераций.

Алгоритм 4.1. Многосеточный предобусловливатель для операторного уравнения, V-цикл

Пусть $B r \equiv B_J r_J$, *и эта операция задается рекурсивным способом:*

(1) ***Предварительное сглаживание:*** *положив* $e_k^0 = 0$, *вычисляем для* $l = 1,2,\ldots,m(k)$ $\quad e_k^l = e_k^{l-1} + R_k^h\left(r_k^h - A_k^h e_k^{l-1}\right)$, *где* R_k^h – *сглаживающий оператор.*

(2) ***Коррекция на грубом уровне:*** *вычислим приближенно решение* $e_{k-1} \in V_{k-1}^h$ *задачи* $A_{k-1}^h e = r_{k-1}^h \equiv Q_{k-1}^h\left(r_k^h - A_k^h e_k^{m(k)}\right)$ *по формуле* $e_{k-1} = B_{k-1}^h r_{k-1}^h$, *где оператор* $B_{k-1}^h : V_{k-1}^h \to V_{k-1}^h$ *для* $k > 2$ *определяется рекурсивно по шагу (1) алгоритма, а для* $k = 2$: $B_{k-1}^h \equiv B_1^h = \left(A_1^h\right)^{-1}$.

(3) ***Окончательное сглаживание:*** *положив* $e_k^{m(k)+1} = e_k^{m(k)} + e_{k-1}$, *вычисляем для* $l = m(k)+2,\ldots,2m(k)+1$ $e_k^l = e_k^{l-1} + R_k^h\left(r_k^h - A_k^h e_k^{l-1}\right)$.

(4) В итоге определяем $B_k^h r_k^h = e_k^{2m(k)+1}$.

Однако на практике решается не операторное уравнение (4.3), а СЛАУ (2.4). И в этом случае на 3-м шаге алгоритма 4.1 нельзя будет записать $e_k^{m(k)+1} = e_k^{m(k)} + e_{k-1}$, поскольку $e_k^{m(k)}$ и e_{k-1} – вектора разной размерности. А значит, потребуется определить матрицу продолжения P_{k-1}^k: $R^{n_{k-1}} \to R^{n_k}$, где $n_k = \dim\left(V_k^h\right)$. С ее помощью можно продолжить поправку e_{k-1} с крупной сетки на более мелкую:

$$e_k^{m(k)+1} = e_k^{m(k)} + P_{k-1}^k e_{k-1}.$$

Матрица проектирования Q в этих терминах определяется как $\left(P_{k-1}^k\right)^T$. В результате,

$$A_{k-1} = \left(P_{k-1}^k\right)^T A_k P_{k-1}^k, \quad r_{k-1} = \left(P_{k-1}^k\right)^T r_k, \tag{7}$$

и многосеточный алгоритм перепишется в виде.

Алгоритм 4.2. Многосеточный предобусловливатель для СЛАУ, V-цикл

Как и прежде $B\,r \equiv B_J\,r_J$, *и эта операция задается рекурсивным способом:*

(1) ***Предварительное сглаживание:*** *положив* $e_k^0 = 0$, *вычисляем для* $l = 1,2,\ldots,m(k)$ $\quad e_k^l = e_k^{l-1} + R_k\left(r_k - A_k e_k^{l-1}\right)$, *где* R_k – *сглаживающий оператор.*

(2) ***Коррекция на грубом уровне:*** *вычислим приближенно решение* $e_{k-1} \in V_h^{k-1}$ *задачи* $A_{k-1} e = r_{k-1} \equiv \left(P_{k-1}^k\right)^T \left(r_k - A_k e_k^{m(k)}\right)$ *по формуле* $e_{k-1} = B_{k-1} r_{k-1}$, *где оператор* $B_{k-1} : V_h^{k-1} \to V_h^{k-1}$ *для* $k > 2$ *определяется рекурсивно по шагу (1) алгоритма, а для* $k = 2$: $B_{k-1} \equiv B_1 = A_1^{-1}$.

(3) ***Окончательное сглаживание:*** *положив* $e_k^{m(k)+1} = e_k^{m(k)} + P_{k-1}^k e_{k-1}$, *вычисляем для* $l = m(k)+2,\ldots,2m(k)+1$ $e_k^l = e_k^{l-1} + R_k\left(r_k - A_k e_k^{l-1}\right)$.

(4) *В итоге определяем* $B_k r_k = e_k^{2m(k)+1}$.

В качестве сглаживающего оператора рекомендуется использовать симметричный предобусловливатель Гаусса-Зейделя

$$R = \left(D + \overline{A}\right)^{-1} D \left(D + \underline{A}\right)^{-1}, \tag{4.8}$$

где $A = \underline{A} + D + \overline{A}$. Действие этого оператора равносильно применению одной итерации прямого метода Гаусса-Зейделя, а затем одной итерации обратного.

§5 Методы неполной факторизации

В качестве еще одной техники предобусловливания следует отметить так называемые методы неполной (приближенной) факторизации.

Неполной факторизацией матрицы A называется следующее разложение

$$A = LU + R \quad \to \quad B = \left(LU\right)^{-1}, \tag{5.1}$$

где L – нижняя, U – верхняя треугольные матрицы. “Остаточная” матрица R определяет точность факторизации. При создании алгоритмов неполной факторизации стараются минимизировать $\|R\|$ и шаблон матриц L и U. Под шабло-

ном матрицы понимается множество пар индексов, соответствующих ненулевым элементам матрицы: $NZ(A)=\{(i,j):\ A_{i,j}\neq 0\}$.

Классическим и самым простым является следующий алгоритм.

Алгоритм 5.1. ILU-факторизация

(1) $For\ i=2..n\ \ Do$

(2) $\quad For\ k=1..i-1\ and\ (i,k)\in NZ(A)\ Do$

(3) $\quad\quad Compute\ \ A_{i,k}=A_{i,k}/A_{k,k}$

(4) $\quad\quad For\ \ j=k+1..n\ \ and\ (i,j)\in NZ(A)\ Do$

(5) $\quad\quad\quad Compute\ \ A_{i,j}=A_{i,j}-A_{i,k}A_{k,j}$

(6) $\quad\quad EndDo$

(7) $\quad EndDo$

(8) $EndDo$

(9) $L=\underline{A}+I,\ \ U=D+\overline{A},\ \ где\ \ A=\underline{A}+D+\overline{A},\ \ I\ -единичная\ матрица$

Достоинством этого алгоритма является простота реализации и то, что $NZ(L)+NZ(U)=NZ(A)$. Однако точность этой факторизации небольшая, поскольку в матрице R многие элементы могут оказаться большими по модулю.

Одним из способов уточнения ILU факторизации является расширение шаблонов матриц L и U с помощью матрицы уровней lev.

Алгоритм 5.2. ILU(p)- факторизация

(1) $lev_{i,j}=\begin{cases}0 & if\ \ A_{i,j}\neq 0\ \ or\ \ i=j\\ \infty & else\end{cases}$

(2) $For\ i=2..n\ \ Do$

(3) $\quad For\ k=1..i-1\ \ and\ \ lev_{i,k}\le p\ \ Do$

(4) $\quad\quad Compute\ \ A_{ik}=A_{ik}/A_{kk}$

(5) $\quad\quad For\ \ j=k+1..n\ \ Do$

(6) $\quad\quad\quad Compute\ \ A_{i,j}=A_{i,j}-A_{i,k}A_{k,j}\ and\ lev_{i,j}=\min(lev_{i,j},lev_{i,k}+lev_{k,j}+1)$

(7) $\quad\quad EndDo$

(8) *EndDo*

(9) *Replace any element in row* i *with* $lev_{i,j} > p$ *by zero*

(10) $L = \underline{A} + I$, $U = D + \overline{A}$

(11) EndDo

Качества этого алгоритма определяет параметр p. Чем он больше, тем точнее получается факторизация. Недостатком метода является то, что "мощность" шаблонов матриц L и U (количество ненулевых элементов матриц) при $p \geq 1$ заранее неизвестна и даже при больших p факторизация может оказаться неточной. Это может произойти из-за того, что в строке (9) алгоритма 5.2 из матрицы L или U будет отброшен большой по модулю элемент.

Следующий алгоритм позволяет отчасти преодолеть эти недостатки.

Алгоритм 5.3. ILUT(τ, p)-факторизация

(1) *For* $i = 1..n$ *Do*

(2) $w = A_{i,*}$ // $A_{i,*}$ *– i-ая строка матрицы*

(3) *For* $k = 1..i-1$ *and* $w_k \neq 0$ *Do*

(4) *Compute* $w_k = w_k / A_{k,k}$ *and Apply dropping rule to* w_k

(5) *If* $w_k \neq 0$ *Then Compute* $w = w - w_k U_{k,*}$

(6) *EndDo*

(7) *Apply dropping rule to row* w

(8) $L_{i,j} = w_j$ *for* $j = 1..i-1$

(9) $U_{i,j} = w_j$ *for* $j = i..n$

(10) EndDo

В строке (4) алгоритма 5.3 отбрасываются (обнуляются) те w_k, величина которых по модулю меньше $\tau_i = \tau \left\| A_{i,*} \right\|$. В строке (7) отбрасывают сначала величины по модулю меньшие τ_i, а затем в строке w оставляют p максимальных по модулю элементов, соответствующих матрице L, и p максимальных по мо-

дулю элементов, соответствующих матрице U. Тем самым из факторизации отбрасываются прежде всего малые по модулю величины. При этом мощность шаблонов матриц L и U будет заранее известна.

Недостатком алгоритма 5.3 является то, что трудно определить, какими именно должны быть параметры τ и p. Как правило, это определяется экспериментально. При этом в общем случае алгоритм не сохраняет симметричность: если A – симметричная матрица, то LU может быть несимметричной. Поэтому для симметричных матриц применяют, например, алгоритм ILUS.

§6 Этапы численного решения краевой задачи

Изложим основные этапы решения краевой задачи в расчетном задании.

Исходная задача

$$-\nabla\cdot\left(a(x,y)\nabla u\right)=f(x,y) \text{ в области } \Omega, \quad u\big|_{\partial\Omega}=g(x,y)$$

Метод Конечных Элементов

$\Downarrow$ (1 этап)

$$Au=F$$

Предобусловливание:

1) многосеточный V - цикл $(e=Br)$,

2) ILUS или AINV (2 этап)

$\Downarrow$

$$BAu=BF$$

Метод сопряженных градиентов

$\Downarrow$ (3 этап)

РЕШЕНИЕ

Первый этап включает в себя аппроксимацию задачи, которая осуществляется с помощью алгоритмов 2.1 и 2.2.

Второй этап – это построение предобуловливателя для полученной СЛАУ. В работе должны быть реализованы два предобусловливателя: многосеточный V-цикл – алгоритм 4.2 и метод неполной факторизации – алгоритм 5.1.

На третьем этапе происходит непосредственное решение предобусловленной системы методом сопряженных градиентов – алгоритм 3.2.

§7 Форматы хранения разреженных матриц

Поскольку матрица жесткости в методе конечных элементов является разреженной, то в программе требуется использовать специальные форматы хранения ее в памяти.

Одним из самых простых в применении форматов является представление разреженной матрицы в виде двух прямоугольных матриц (7.1): вещественной val, в которой хранятся значения ненулевых элементов разреженной матрицы, и целочисленной ind, в которой хранятся номера вторых индексов соответствующих ненулевых элементов. Размерность этих матриц $n \times width$, где n – размерность исходной матрицы, а $width$ – максимальное количество ненулевых элементов на строке. В результате, доступ к элементу $A_{i,j}$ осуществляется следующим образом. В i-ой строке матрицы индексов ind ищется значение, равное j. Если значение найдено в k-ом столбце матрицы ind, то в качестве значения $A_{i,j}$ возвращается $val_{i,k}$. Иначе $A_{i,j}$ равно 0.

Ниже приведен пример хранения матрицы в этом формате с нумерацией элементов от 1 до n. Изначально матрицы val и ind являются нулевыми. Ненулевые элементы при сборке добавляются в соответствующие строки матриц val и ind слева направо.

$$A = \begin{pmatrix} 10 & 3 & 1 & 0 & 0 & 2 \\ 0 & 6 & 0 & 3 & 2 & 0 \\ 2 & 0 & 4 & 0 & 0 & 0 \\ 1 & 1 & 0 & 5 & 0 & 3 \\ 0 & 5 & 0 & 0 & 7 & 0 \\ 4 & 0 & 0 & 5 & 0 & 9 \end{pmatrix} \Rightarrow val = \begin{pmatrix} 10 & 1 & 2 & 3 \\ 6 & 3 & 2 & 0 \\ 4 & 2 & 0 & 0 \\ 5 & 1 & 1 & 3 \\ 7 & 5 & 0 & 0 \\ 9 & 4 & 5 & 0 \end{pmatrix} \quad ind = \begin{pmatrix} 1 & 3 & 6 & 2 \\ 2 & 4 & 5 & 0 \\ 3 & 1 & 0 & 0 \\ 4 & 1 & 2 & 6 \\ 5 & 2 & 0 & 0 \\ 6 & 1 & 4 & 0 \end{pmatrix} \qquad (7.1)$$

Данный формат хорош тем, что позволяет легко реализовать конечно-элементную сборку. Но в качестве недостатка можно отметить то, что формат

неэффективно расходует память в случае, если число ненулевых элементов на каждой строке сильно разнится. Более эффективными с этой точки зрения являются так называемые сжатые форматы хранения данных. Одним из наиболее популярных форматов является CRS (Compressed Row Storage). Данные в этом формате хранятся в виде 3 векторов. В первом (val) хранятся значения ненулевых элементов матрицы, во втором (col_ind) – соответствующие вторые индексы, а в третьем (row_ptr, размерности размерности $n+1$) – смещения, которые указывают на начала строк в первых двух векторах. Последний элемент третьего вектора задает конец матрицы (размерность первых массивов + 1).

val	10	3	1	2	6	3	2	2	4	1	1	5	3	5	7	4	5	9	
col_ind	1	2	3	6	2	4	5	1	3	1	2	4	6	2	5	1	4	6	
row_ptr	1				5			8		10				14		16			19

CRS (Compressed Row Storage)

Также иногда используется формат CCS (Compressed Column Storage). Он аналогичен CRS. Отличие заключается в том, что элементы хранятся не по строкам, а по столбцам.

В качестве еще одно сжатого формата стоит упомянть JDS (Jagged Diagonal Storage). Суть его заключается в том, что сначала собираются ненулевые элементы матрицы в структуру $NonZero$, после чего строки $NonZero$ сортируются по количеству элементов в них:

$$A=\begin{pmatrix}10&3&1&0&0&2\\0&6&0&3&2&0\\2&0&4&0&0&0\\1&1&0&5&0&3\\0&5&0&0&7&0\\4&0&0&5&0&9\end{pmatrix}\Rightarrow NonZero=\begin{pmatrix}10&3&1&2\\6&3&2&\\2&4&&\\1&1&5&3\\5&7&&\\4&5&9&\end{pmatrix}\overset{sort}{\Rightarrow}\begin{pmatrix}10&3&1&2\\1&1&5&3\\4&5&9&\\6&3&2&\\5&7&&\\2&4&&\end{pmatrix}$$

Далее в качестве Jagged диагоналей обозначаются столбцы отсортированной структуры NonZero. Эти столбцы по очереди добавляются в вектор Jdiag, а в вектор col_ind добавляются вторые индексы соответствующих элементов раз-

реженной матрицы. В векторе jd_ptr хранятся смещения, которые указывают на начала Jagged диагоналей, в векторе perm – данные об изменении нумерации строк при сортировке.

Jdiag	10	1	4	6	5	2	3	1	5	3	7	4	1	5	9	2	2	3	
col_ind	1	1	1	2	2	1	2	2	4	4	5	3	3	4	6	5	6	6	
Jd_ptr	1						7						13				17		19
Perm	1	4	6	2	5	3													

JDS (Jagged Diagonal Storage)

Основное достоинство этого формата заключается в том, что в векторе col_ind, в отличие от формата CRS, вторые индексы “скачут” не так сильно, что положительно сказывается на качестве «кэширования» оперативной памяти при реализации операции умножения разреженной матрицы на вектор.

В заключение отметим, что при работе с симметричными матрицами имеет смысл использовать форматы, учитывающие симметрию. Представим симметричную матрицу в виде $A = \underline{A} + D + \underline{A}^T$, где $\underline{A}$ – нижняя треугольная матица, D – диагональная. Для уменьшения объема расходуемой памяти, очевидно, следует хранить только элементы матриц $\underline{A}$ и D. Если $\underline{A}$ хранится в формате CRS, а диагональ D – отдельно в векторе, то такой формат хранения симметричных матриц называется SSK (Sparse SKyline).

В работе для простоты рекомендуется использовать первый формат (7.1).

§8 Форматы хранения сеток

Любая сетка состоит из множества ячеек (Cells). Ячейки в двухмерном случае задаются совокупностью ребер (Edges). Ребра задаются узлами (Nodes).

Поскольку основой всех элементов сетки являются узлы, то обязательно следует их пронумеровать и задать координаты XY(Nodes).

Далее по мере необходимости требуется определить связи (connectivities) между элементами сетки (узлами, ребрами, ячейками). Так, например, для конечно-элементной сборки требуется задать связь между нумерацией ячеек и уз-

лов Nodes(Cells). В случае треугольных ячеек (рис. 8.1) эту связь удобно задать с помощь матрицы размерности $N_T \times 3$:

$$mNodesOnCells = \begin{pmatrix} 1 & 4 & 5 \\ 1 & 5 & 2 \\ 2 & 5 & 6 \\ 2 & 6 & 3 \\ \dots & \dots & \dots \\ 5 & 9 & 6 \end{pmatrix}.$$

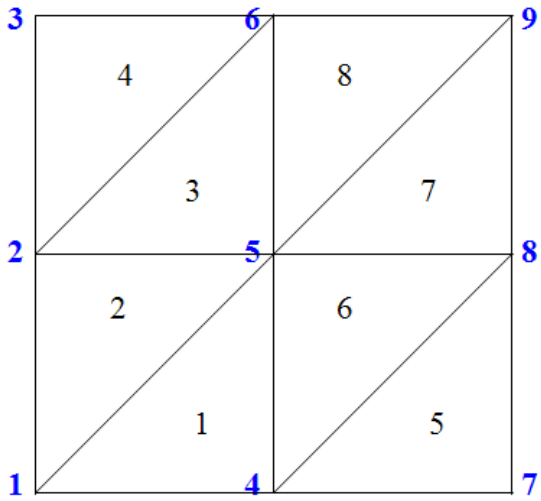

Рис. 8.1 Пример грубой сетки
Синие цифры – номера узлов
Черные цифры – номера ячеек

После определения этой связи алгоритм 2.1 можно записать в виде:

Алгоритм 8.1. Конечно-элементная сборка СЛАУ после введения связи в нумерациях ячеек и узлов Nodes(Cells)

(1) $For\ \ T = 1..N_T\ \ Do$ *// N_T – число треугольников сетки*

(2) *Вычисляем матрицу* $A^{(T)}$ *и вектор правой части* $F^{(T)}$ *на треугольнике*

(3) $For\ \ i = 1..3\ \ Do$

(4) $I = mNodesOnCells(T,\ i)$

(5) $For\ \ j = 1..3\ \ Do$

(6) $J = mNodesOnCells(T,\ j)$

(7) $A(I,J) = A(I,J) + A^{(T)}(i,j)$

(8) *EndDo*

(9) $F(I) = F(I) + F^{(T)}(i)$

(10) *EndDo*

(11) *EndDo*

Таким образом, для аппроксимации задачи методом конечных элементов нужно задать координаты узлов XY(Nodes) и связь Nodes(Cells).

§9 Алгоритм построения набора вложенных сеток

Для реализации многосеточного V-цикла требуется построения набора вложенных сеток.

В качестве самой грубой сетки можно взять сетку, изображенную на рис. 8.1. Ее удобно задать вручную. Следующую сетку получим, измельчив грубую, разбивая каждое ребро пополам, а каждый треугольник на четыре новых (см. рис. 9.1). Для реализации алгоритма измельчения сетки потребуются две вспомогательные связи Nodes(Edges) и Edges(Cells).

Алгоритм 9.1. Измельчение сетки

Исходные данные: XY(NodesC), NodesC(EdgesC) и EdgesC(CellsC) исходной сетки. Буква C обозначается принадлежность к грубой сетке (Coarse), F – к мелкой сетке (Fine).

(1) Ребра исходной сетки на основе связи NodesC(EdgesC) ся пополам. Задаются нумерация и координаты узлов новой сетки XY(NodesF). Собирается часть связи NodesF(EdgesF) для следующей (более мелкой) сетки (в связь добавляются ребра, изображенные сплошной линией на рис. 9.1). Нумерация новых ребер производится таким образом, чтобы номера потомков i-го ребра были бы $2i-1$ и $2i$.

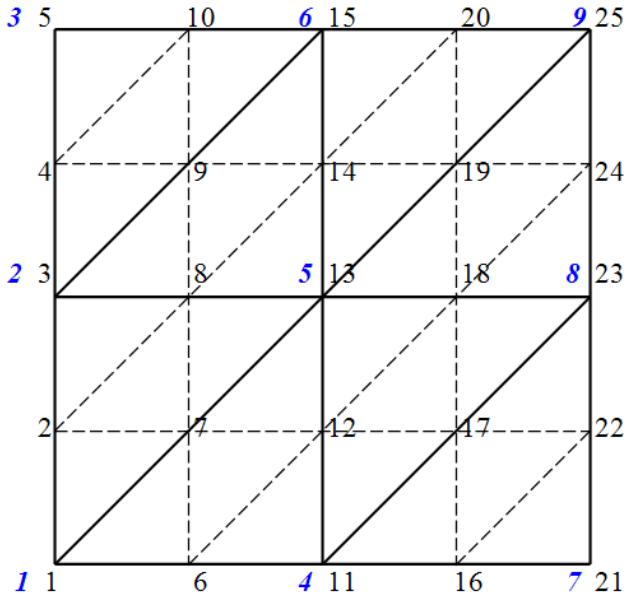

Рис. 9.1 Получение новой вложенной сетки путем измельчения грубой сетки

(2) Далее бежим по треугольникам исходной сетки, разбивая их на 4 новых. С помощью связи EdgesC(CellsC), и зная соотношение между нумерацией ребер грубой сетки и их потомками (ребрами мелкой сетки), можно легко построить связь EdgesF(CellsF) для мелкой сетки. Одновременно дост-

раивается связь NodesF(EdgesF) (в связь добавляются "внутренние" ребра, изображенные пунктирной линией на рис. 9.1)

(3) В итоге на мелкой сетке получаем координаты XY(NodesF) и связи NodesF(EdgesF), EdgesF(CellsF), по которым однозначно определяется связь NodesF(CellsF)= NodesF(EdgesF(CellsF)).

Для реализации многосеточного предобусловливателя остается построить оператор продолжения P_{k-1}^{k} из алгоритма 4.2. Этот оператор однозначно определяется при разбиении ребер в п.1 алгоритма 9.1. При разбиении каждого ребра грубой $(k-1)$ сетки (см. рис. 9.2) к матрице P_{k-1}^{k} нужно сделать следующую добавку:

$$P_{k-1}^{k} = P_{k-1}^{k} + \begin{array}{c} \begin{matrix} i & j \\ \downarrow & \downarrow \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} \\ \leftarrow n \\ \leftarrow l \\ \\ \leftarrow m \end{array}$$

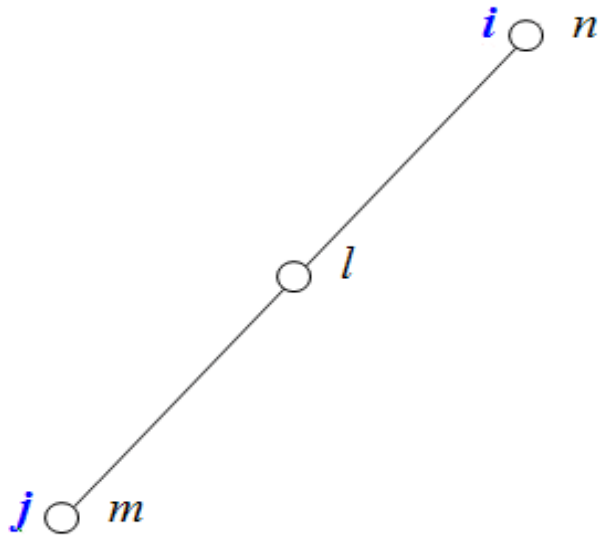

Рис. 9.2 Разбиение ребра

§10 Тестовые и методические расчеты

После реализации вычислительных алгоритмов студенты должны проверить работоспособность своей программы. Для этого нужно определить задачу (1.1), (1.2) с заранее известным решением. В качестве решения следует взять трансцендентную по x и по y функцию, например, $u_{exact}(x,y)=\sin(x+y)$. При

этом правая часть (1.1) получается подстановкой решения $u_{exact}(x,y)$ в левую часть уравнения, что приводит к $f(x,y) = -\nabla \cdot (a(x,y)\nabla u_{exact}(x,y))$. Граничные условия также должны соответствовать заранее известному решению: $g(x,y) = u_{exact}(x,y)$.

Помимо исследования сходимости приближенного решения задачи к точному $u_{exact}(x,y)$ при измельчении шага сетки h, требуется показать, что аппроксимация имеет порядок $O(h^2)$.

Дополнительные численные эксперименты должны быть направлены на изучение скорости сходимости итерационных методов решения СЛАУ – с предобусловливателями и без них.

Литература

[1] Баландин М.Ю., Шурина Э.П. Методы решения СЛАУ большой размерности – Новосибирск: НГТУ, 2000. http://fpmi.ami.nstu.ru/arhive/courses/umf/MethodsSLE.pdf

[2] Василевский Ю.В., Ольшанский М.А. Краткий курс по многосеточным методам декомпозиции области – Москва, 2007. http://www.inm.ras.ru/library/Vassilevski/yuv-olsh-mnogoset-dekomp.pdf

[3] Зенкевич О., Морган К. Конечные элементы и аппроксимация. – Москва: Мир, 1986.

[4] Коннор Дж., Бреббия К. Метод конечных элементов в механике жидкости – Ленинград: Судостроение, 1979.

[5] Михайлов В.П., Гущин А.К. Дополнительные главы курса “Уравнения математической физики” – Москва, 2007. http://www.mi.ras.ru/noc/lectures/07mihailovgushchin.pdf

[6] Шайдуров В.В. Многосеточные методы конечных элементов – Москва: Наука, 1989.