Южный федеральный университет

М. Ю. Жуков, Е. В. Ширяева

Использование пакета конечных элементов FreeFem++ для задач гидродинамики, электрофореза и биологии

Ростов-на-Дону
2008

УДК 519.63; 532.5; 537.364
ББК 22.25
Ж 86

Печатается по решению
кафедры вычислительной математики и математической физики
и Совета факультета математики, механики и компьютерных наук ЮФУ
от 15.09.2007

Рецензенты
Я. М. Ерусалимский, Н. В. Петровская

Жуков М. Ю., Ширяева Е. В.

Использование пакета конечных элементов FreeFem++ для задач гидродинамики, электрофореза и биологии. — Ростов н/Д: Изд-во ЮФУ, 2008. — 256 с., ил.

Учебное пособие предназначено для практического освоения языка FreeFem++, используемого для численного решения задач математической физики методом конечных элементов. Содержит подробные программные коды и большой набор задач. Рекомендуется студентам факультета математики, механики и компьютерных наук ЮФУ.

Оглавление

Предисловие

FreeFem++ это интегрированная среда разработки (IDE) со своим собственным высокоуровневым языком программирования, предназначенная для численного решения дифференциальных уравнений в частных производных методом конечных элементов. FreeFem++ позволяет исследовать стационарные и нестационарные математические модели (в физике, химии, биологии, инженерных приложениях и т. п.) в 2D-пространственном случае и является прекрасным инструментом как для обучения методу конечных элементов, так и для решения научных проблем.

FreeFem++ — свободно распространяемое математическое обеспечение, которое используется на различных платформах (Unix OS, Windows 9x, 2000, NT, XP и MacOS X). Авторы FreeFem++ — F. Hecht, O. Pironneau, A. Le Hyaric, K. Ohtsuka — сотрудники лаборатории Ж.-Л. Лионса университета Пьера и Марии Кюри (Париж VI) и Французской Академии Наук. В комплект установки FreeFem++[1], помимо самой интегрированной среды разработки, включено большое количество примеров и подробное руководство пользователя [1]. Заметим, что хотя FreeFem++ и является средой разработки, по существу, это все-таки компилятор специального языка, существенно ориентированного на работу с методом конечных элементов и поэтому в дальнейшем удобно говорить именно о ***языке*** FreeFem++.

Коротко перечислим возможности FreeFem++:

1. Построение численного решения стационарных и нестационарных, линейных и нелинейных двумерных краевых задач.

2. Непосредственно, т. е. без составления алгоритма, возможно решение только линейных стационарных краевых задач. При этом исходная задача должна быть представлена в слабой (вариационной) формулировке.

3. Решение нестационарных и нелинейных краевых задач требует составления некоторого алгоритма решения, пошагового для нестационарных и итерационного для нелинейных задач, позволяющего сводить исходную задачу к набору линейных краевых задач.

4. Пространственно двумерная область, в которой строится решение, описывается при помощи границ, задаваемых в параметрическом виде удобными аналитическими соотношениями.

5. Для триангуляции области используется встроенный автоматический генератор сеток, основанный на алгоритме Делоне–Вороного, при этом плотность внутренних точек области пропорциональна плотности точек

[1] `http://www.freefem.org/ff++`

на ее границе. При построении сеток применяется анизотропная адаптация, позволяющая эффективно перестраивать сетку в процессе проведения вычислений. Сгенерированные сетки можно сохранять в виде файлов и использовать в дальнейшем.

6. Для построения решения имеется большой набор финитных базисных функций (конечных элементов): кусочно-постоянные, линейные и квадратичные лагранжевы элементы и др. Имеется возможность создавать свои собственные базисные функции.

7. Для проведения расчетов имеется большой набор быстрых алгоритмов для решения эллиптических конечно-разностных задач. Это разнообразные прямые и итеративные алгоритмы решения систем линейных уравнений — LU, Cholesky, Crout, CG, GMRES, UMFPACK, а также алгоритмы для нахождения собственных значений и собственных векторов.

8. Имеется возможность получения информации о решении в графическом виде непосредственно на экране дисплея, а также в виде текстовых и postscript-файлов.

9. Алгоритмы записываются на языке, близком к C++ (знание C++ не требуется). При этом эффективность кода по скорости близка к оптимальной, сравнимой со скоростью программ, непосредственно написанных на языке C++.

10. В случае необходимости имеется возможность доступа к внутренним векторам и матрицам, что, в частности, позволяет создавать собственные алгоритмы решения задач взамен имеющихся стандартных.

По сравнению с коммерческими пакетами, предназначенными для решения уравнений в частных производных, например, FemLab или FlexPDE, использование FreeFem++ требует бóльшей «умственной» работы от пользователя. В частности, невозможно записать задачу в исходном виде (FlexPDE предоставляет такую возможность, а FemLab содержит список задач, в котором можно задавать численные значения параметров) — требуется преобразование к так называемой слабой формулировке. Менее развит и интерфейс среды FreeFem++. Однако эти минусы почти полностью компенсируются простотой языка FreeFem++, возможностью доступа ко всем внутренним данным и возможностью создания собственных алгоритмов.

Иными словами, FreeFem++ это не «черный ящик», решающий задачи, а инструмент для исследовательской работы, которая, зачастую, требует конструирования уникальных методов решения. Одновременно с этим простота языка FreeFem++, как уже говорилось, позволяет использовать его для обучения способам применения метода конечных элементов.

Предлагаемая книга предназначена читателям, желающим использовать метод конечных элементов для решения своих задач. В ней содержится большое количество примеров применения средств языка FreeFem++ к решению разнообразных задач математической физики. Книга ни в коем случае не заменяет руководства по использованию FreeFem++ [1]. Более того, предполагается, что чтение данной книги и [1] будет осуществляться параллельно. В [1] читатель найдет большое количество примеров и сможет изучить практически все возможности языка FreeFem++.

Коротко опишем структуру материала в книге, который разбит на три

части. В первой части (гл. 1, 2) дано краткое изложение метода конечных элементов для одномерного и двумерного случаев. Вторая часть — это алгоритмы решения на языке FreeFem++ различных конкретных задач. Спектр этих задач весьма широк: простейшие задачи для уравнения Лапласа в гл. 3, 4, задачи для уравнения переноса в гл. 5, задача об окраске шкур животных в гл. 6. Именно в этих главах содержатся основные алгоритмы и приемы, которые затем активно используются. Гл. 7–15 содержат задачи, связанные с течением жидкости: вихревые течения в гл. 7, тепловая конвекция в гл. 8, течения в каналах и обтекание препятствий в гл. 9, течение жидкости со свободной границей в гл. 14, течение Куэтта-Тейлора в гл. 15. Гл. 10–13 содержат задачи о переносе примесей в жидкости.

Структура всех глав второй части примерно одинакова. Во-первых, это подробная постановка математической задачи с необходимыми сведениями из физики, биологии и т. п. Во-вторых, это алгоритм решения и код программы на языке FreeFem++. В-третьих, это иллюстрированные результаты вычислительного эксперимента. Такая схема, помимо изучения языка, позволяет, в частности, использовать книгу в качестве справочника для получения информации о постановке задач и результатах их численного решения. Наличие работоспособных кодов позволяет читателю без труда воспроизвести приводимые результаты расчетов и, изменив параметры, получить ответы на интересующие его вопросы.

Третья часть содержит детальное описание языка FreeFem++ и предназначена для углубленного изучения его возможностей. В первую очередь это описание синтаксиса в гл. 16, описание приемов управления триангуляцией области в гл. 17, описание принципов выбора базисных функций для построения приближенного решения в гл. 18. В этой части, в гл. 19, описан выбор методов численного решения эллиптических конечно-разностных задач (систем линейных алгебраических уравнений) и изложены способы решения краевых задач, отличные от приведенных во второй части. В гл. 20 изложен способ решения краевых задач на собственные значения при помощи языка FreeFem++. Наконец, гл. 21 посвящена возможностям визуализации результатов расчета.

Неоценимую помощь авторам оказали студенты факультета математики, механики и компьютерных наук Южного федерального университета Ю. К. Самадова, Д. Д. Дуброва и И. И. Чернухина (подготовка части рисунков и тестирование алгоритмов), а также сотрудники университета Н. М. Жукова, прочитавшая рукопись и лишившая будущего читателя возможности обнаружить большое количество опечаток, И. В. Ширяева, сделавшая ряд важных замечаний о синтаксисе языка при написании гл. 16.

Работа выполнена благодаря поддержке внутренних грантов Южного федерального университета, а также при поддержке грантов РФФИ (07-01-00389а), Европейского научного объединения «Регулярная и хаотическая гидродинамика» (07-01-9213) и гранта Президента поддержки ведущих научных школ Российской Федерации (НШ.5747.2006.1).

Часть I
Основы метода конечных элементов

Метод конечных элементов (МКЭ) является популярным и эффективным методом численного решения различных задач математической физики. Привести список огромного количества литературы, посвященной основам метода и проблемам, связанным с его применением, — практически невыполнимая задача. Метод давно стал классическим и в настоящее время его описание традиционно включается во многие учебники по численным методам [2–5] (см. также [6–14]).

Сравнительная простота реализации МКЭ привела к созданию большого количества программного обеспечения, дающего возможность использовать метод без специальной математической подготовки. Не претендуя на полноту, упомянем лишь такие коммерческие программные продукты, как FemLab и FlexPDE, позволяющие записать задачу в привычных математических обозначениях и получить ее численное решение. Собственно говоря, и данный учебник посвящен описанию использования подобного программного обеспечения. Такой подход, не предполагающий специальных знаний, зачастую оправдан. Однако для успешного, эффективного и правильного применения численного метода (МКЭ) знание его математических основ является все же обязательным. Во многих случаях это позволяет избежать неверных интерпретаций результатов расчета.

В этой части книги даются лишь самые необходимые сведения о МКЭ. Читателю достаточно лишь знакомства с основами уравнений математической физики, линейной алгебры и численных методов. При необходимости все специальные понятия приводятся непосредственно в тексте. Изложение материала дается на простых примерах, позволяющих, по мнению авторов, получить ясное представление о методе и его возможностях.

Скажем несколько слов об используемой терминологии. Несмотря на то, что метод конечных элементов известен очень давно, до сих пор в литературе используются различные термины. Так, сам метод называется — метод конечных элементов, вариационно-разностный метод, проекционно-сеточный метод и т. д. Для названия базисных функций используются термины: тестовые функции, пробные функции и т. п. Это связано с широкой применимостью метода для решения различного рода задач — задач теории упругости, гидродинамики и переноса, электродинамики, инженерных задач и пр. В каждой из упомянутых областей принята своя специфическая терминология, зачастую не очень математически строгая.

Глава 1

Метод конечных элементов в одномерном случае

Основная область применения метода конечных элементов это, конечно же, решение уравнений в частных производных для двумерных и трехмерных областей, особенно, когда область имеет сложную геометрическую форму. Однако, для понимания идей метода и большинства его наиболее важных особенностей, удобнее начать с рассмотрения одномерных краевых задач. Традиционно принято излагать основы метода, используя вариационную формулировку задачи. Однако, на взгляд авторов, это существенно облегчает лишь проведение строгих математических доказательств, но не вносит ничего нового в понимание сути метода. Намного проще изначально использовать проекционный подход, трактуя метод конечных элементов как разновидность метода Галеркина.

1.1 Сильное и слабое решение задачи

Рассмотрим первую краевую задачу (задачу Дирихле) для определения функции $u(x)$

$$-\frac{d}{dx}\left(p(x)\frac{du(x)}{dx}\right)+q(x)u(x)=f(x), \qquad 0<x<1, \tag{1.1}$$

$$u(0)=0, \qquad u(1)=0, \tag{1.2}$$

где $p(x)$, $q(x)$, $f(x)$ — известные функции.

Предположим, что функции $q(x)$, $f(x)$ непрерывны, а функция $p(x)$ — непрерывно дифференцируема. Решение задачи (1.1), (1.2), являющееся дважды непрерывно дифференцируемой функцией, будем называть ***сильным решением***.

Можно попытаться переформулировать задачу (1.1), (1.2) с целью снижения требований, накладываемых на непрерывность функций $p(x)$, $q(x)$, $f(x)$ и функции $u(x)$. Для этого умножим уравнение (1.1) на некоторую

функцию $v(x)$ и проинтегрируем на отрезке $[0,1]$

$$-\int_0^1 \frac{d}{dx}\left(p(x)\frac{du(x)}{dx}\right)v(x)\,dx+\int_0^1 q(x)u(x)v(x)\,dx=\int_0^1 f(x)v(x)\,dx.$$

Используя формулу интегрирования по частям, имеем

$$\int_0^1 p\frac{du}{dx}\frac{dv}{dx}\,dx-p\frac{du}{dx}v\Big|_0^1+\int_0^1 quv\,dx=\int_0^1 fv\,dx.$$

Потребуем, чтобы функция $v(x)$ удовлетворяла тем же краевым условиям, что и $u(x)$, т. е.

$$v(0)=0,\quad v(1)=0. \tag{1.3}$$

Тогда

$$\int_0^1\left(p\frac{du}{dx}\frac{dv}{dx}+quv-fv\right)dx=0,\quad v(0)=0,\quad v(1)=0. \tag{1.4}$$

Если соотношение (1.4) выполнено для любых $v(x)$, то оно называется ***слабой формой*** (или вариационной формой) записи задачи (1.1), (1.2). На самом деле, следует дополнительно указать какому классу принадлежат функции $v(x)$, например, $v\in C^1$.

Сам способ получения соотношения (1.4) показывает, что если функция $u(x)$ является решением задачи (1.1), (1.2), то эта же функция будет решением (1.4). Обратное же утверждение в общем случае неверно. Это видно уже из того, что для функций $p(x)$, $q(x)$, $f(x)$ и $u(x)$, входящих в (1.4), можно требовать гораздо меньших ограничений на гладкость, чем для соответствующих функций задачи (1.1), (1.2). Так, например, можно считать функции $p(x)$, $q(x)$, $f(x)$ непрерывными ($p,q,f\in C^0$), а функции $u(x)$, $v(x)$ непрерывно дифференцируемыми ($u,v\in C^1$). На самом деле, можно даже требовать от $p(x)$, $q(x)$, $f(x)$ кусочной непрерывности, а от $u(x)$, $v(x)$ — кусочной гладкости. Заметим также, что для функции $u(x)$, входящей в (1.4), не требуется даже выполнения краевых условий (1.2).

Функцию $u(x)$, являющуюся решением задачи (1.4), будем называть ***слабым решением*** задачи (1.1), (1.2). Из вышесказанного ясно, что сильное и слабое решения задачи (1.1), (1.2) в общем случае не совпадают. Подчеркнем, что сильные и слабые решения могут различаться и в случае, когда для функций $p(x)$, $q(x)$, $f(x)$ сохранены такие же требования на гладкость, как в исходной задаче (1.1), (1.2).

Проблемы связи между собой сильных и слабых решений задачи являются очень важными для метода конечных элементов. Дело в том, что традиционно метод используется для получения приближенных решений именно задач в слабой формулировке (т. е. слабых решений)[1]. Различие

[1] В частности, FreeFem++ предназначен для решения именно таких задач.

между слабым и сильным решениями может быть одной из причин, по которой приближенное решение, полученное методом конечных элементов, может не иметь никакой связи с решением исходной задачи.

Далее вопросы, связанные со сходимостью слабых решений к сильному, не рассматриваются, т. к. основная цель книги — это описание работы с FreeFem++. Ограничимся лишь ссылкой на [3, 5], где эти вопросы исследуются применительно к методу конечных элементов и в которых имеется достаточно обширная библиография.

1.2 Построение приближенного решения задачи

1.2.1 Слабое решение

Будем искать приближенное решение задачи (1.4) в виде

$$u^h(x) = \sum_{k=1}^{n-1} c_k \varphi_k(x), \tag{1.5}$$

где $u^h(x)$ — приближенное решение, функции $\varphi_k(x)$ предполагаются известными, линейно независимыми и называются *базисными функциями* (другие названия — *тестовые* функции, *пробные* функции), c_k — коэффициенты, подлежащие определению.

Формула (1.5) задает аппроксимацию функции $u(x)$ в виде некоторого сужения (проектирования) на *конечномерное* пространство, определяемое базисом $\varphi_k(x)$, $k = 1, \ldots, n-1$. Точность аппроксимации, естественно, будет зависеть от того, насколько «хорошо» выбранное конечномерное пространство приближает исходное пространство, которому должна принадлежать функция $u(x)$. Заметим, что величина u^h при заданных $\varphi_k(x)$ полностью определяется набором чисел $(c_1, c_2, \ldots, c_{n-1})$, т. е. можно считать, что $u^h = (c_1, c_2, \ldots, c_{n-1})$ является вектором, принадлежащим $\mathbb{R}^{n-1}$.

Подставляя (1.5) в (1.4), получим

$$\int_0^1 \left(p \sum_{k=1}^{n-1} c_k \varphi_k' v' + q \sum_{k=1}^{n-1} c_k \varphi_k v - f v \right) dx = 0.$$

Учитывая, что (1.4) должно выполняться для любых $v(x)$, принадлежащих некоторому классу, выберем в качестве $v(x)$ некоторый набор функций, а именно $v(x) = \varphi_i(x)$, $i = 1, \ldots, n-1$. В соответствии с требованиями $v(0) = 0$, $v(1) = 0$ (см. (1.4)) будем считать, что

$$\varphi_k(0) = 0, \quad \varphi_k(1) = 0, \quad k = 1, \ldots, n-1. \tag{1.6}$$

В этом случае имеем

$$\int_0^1 \left(p \sum_{k=1}^{n-1} c_k \varphi_k' \varphi_i' + q \sum_{k=1}^{n-1} c_k \varphi_k \varphi_i - f \varphi_i \right) dx = 0, \quad i = 1, \ldots, n-1.$$

Перепишем это соотношение в виде

$$\sum_{k=1}^{n-1} A_{ik} c_k = b_i, \quad i = 1, \ldots, n-1, \tag{1.7}$$

где введены следующие обозначения

$$A_{ik} = \int_0^1 \left(p\varphi_i'\varphi_k' + q\varphi_i\varphi_k\right) dx, \quad b_i = \int_0^1 f\varphi_i \, dx, \quad i, k = 1, \ldots, n-1. \tag{1.8}$$

Таким образом, для определения c_k имеем систему линейных алгебраических уравнений. Решая эту систему и подставляя c_k в (1.5), получим приближенное решение задачи (1.4). Подчеркнем, что приближенное решение (1.5) в силу условий (1.6) автоматически удовлетворяет краевым условиям (1.2) для исходной задачи.

1.2.2 Метод Галеркина. Сильное решение

Разыскивать решение в виде (1.5) можно и для исходной задачи. Подставляя (1.5) в (1.1), имеем

$$\delta(x) = -\frac{d}{dx}\left(p\sum_{k=1}^{n-1} c_k\varphi_k'\right) + q\sum_{k=1}^{n-1} c_k\varphi_k - f$$

или

$$\delta(x) = -p'\sum_{k=1}^{n-1} c_k\varphi_k' - p\sum_{k=1}^{n-1} c_k\varphi_k'' + q\sum_{k=1}^{n-1} c_k\varphi_k - f. \tag{1.9}$$

Здесь $\delta(x)$ — невязка, возникающая после подстановки приближенного решения $u^h(x)$ в уравнение (1.1).

Умножим соотношение (1.9) на φ_i и, проинтегрировав на $[0, 1]$, потребуем выполнения равенств

$$\int_0^1 \delta(x)\varphi_i(x)\, dx = -\int_0^1 \left(\sum_{k=1}^{n-1} c_k \left(p'\varphi_k' + p\varphi_k'' - q\varphi_k\right) + f\right)\varphi_i \, dx = 0 \tag{1.10}$$

или

$$\sum_{k=1}^{n-1} c_k \int_0^1 \left(-p'\varphi_k'\varphi_i - p\varphi_k''\varphi_i + q\varphi_k\varphi_i\right) dx - \int_0^1 f\varphi_i \, dx = 0, \quad i = 1, \ldots, n-1.$$

Таким образом, вновь получена система линейных алгебраических уравнений для определения c_k

$$\sum_{k=1}^{n-1} \tilde{A}_{ik} c_k = b_i, \quad i = 1, \ldots, n-1,$$

где

$$\tilde{A}_{ik} = \int_0^1 (-p'\varphi_k'\varphi_i - p\varphi_k''\varphi_i + q\varphi_k\varphi_i)\,dx, \quad b_i = \int_0^1 f\varphi_i\,dx. \tag{1.11}$$

Заметим, что функция (1.5) будет являться приближением решения исходной задачи (1.1), (1.2), если на базисные функции $\varphi_k(x)$ будут наложены дополнительные ограничения (1.6) — требования удовлетворения граничным условиям (1.2).

Укажем на основное различие между матрицами A_{ik} и $\tilde{A}_{ik}$. Для вычисления матрицы A_{ik} (см. (1.8)) от функций φ_k требуется, чтобы φ_k' была кусочно-непрерывной. В случае матрицы $\tilde{A}_{ik}$ (см. (1.11)) на функции φ_k следует накладывать более сильные ограничения — требуется, чтобы φ_k'' была кусочно-непрерывной. Конечно, имеется в виду, что матрица $\tilde{A}_{ik}$ вычисляется непосредственно по приведенной формуле и операция интегрирования по частям в (1.11) не используется. Если же в (1.11) произвести операцию интегрирования по частям, то вновь получится слабая формулировка задачи и соотношения (1.11) совпадут с (1.7).

Как уже говорилось, формула (1.5) определяет аппроксимацию функции $u(x)$ в виде некоторого сужения (проектирования) на некоторое *конечномерное* пространство, определяемое базисом $\varphi_k(x)$, $k = 1, \ldots, n-1$. Соотношения (1.10) при условии выполнения (1.6) задают *проекцию* невязки $\delta(x)$ (см. (1.9)) исходной задачи (1.1), (1.2) на *то же самое* пространство. Конечно, это вовсе не обязательно (в случае более сложных задач — просто неправильно) и соотношения (1.10) следует заменить более общими, задающими проектирование невязки $\delta(x)$ на конечномерное пространство, определяемое иным базисом, например, $\psi_k(x)$, $k = 0, \ldots, n-1$

$$\int_0^1 \delta(x)\psi_i(x)\,dx = -\int_0^1 \left(\sum_{k=1}^{n-1} c_k\,(p'\varphi_k' + p\varphi_k'' - q\varphi_k) + f\right)\psi_i dx = 0. \tag{1.12}$$

Подчеркнем, что в этом случае матрица $\tilde{A}_{ik}$ будет определяться соотношениями, отличными от (1.11), и переход от $\tilde{A}_{ik}$ к A_{ik} будет невозможен. Во избежание недоразумений, заметим, что способ проектирования (1.10) или (1.12) при помощи скалярного произведения, определяемого интегрированием, уже подразумевает некоторую слабую формулировку исходной задачи. Действительно, если, например, в формулировку исходной задачи (1.1), (1.2) включено требование принадлежности функций p', q, f, u'' классу непрерывных функций, то при построении приближенного решения с использованием соотношений (1.10) уже можно ограничиваться требованием принадлежности функций p', q, f, φ_k'' (а следовательно и u'') лишь классу кусочно-непрерывных функций.

1.2.3 Конечно-разностный метод

Описанные приемы построения приближенного решения, конечно же, не являются специфическими способами решения исходной задачи. При-

ведем, в частности, конечно-разностный (сеточный) метод, важный для дальнейших целей. Для простоты ограничимся случаем, когда $p(x) = 1$, $q(x) = 0$, т. е. задачей

$$-u''(x) = f(x), \quad 0 < x < 1, \quad u(0) = 0, \quad u(1) = 0. \tag{1.13}$$

Разобьем отрезок $[0, 1]$ на интервалы с одинаковой длиной $[x_k, x_{k+1}]$, $k = 0, \ldots, n-1$, $x_i = ih$, $h = 1/n$. Рассмотрим уравнение (1.13) в точке x_k и аппроксимируем производную $u''(x_k)$ центральной конечной разностью (см. [4]). Тогда, с учетом краевых условий, для определения $u_k = u(x_k)$, $k = 0, \ldots, n$ получим систему линейных алгебраических уравнений

$$-u_{k-1} + 2u_k - u_{k+1} = h^2 f(x_k), \quad k = 1, \ldots, n-1, \quad u_0 = 0, \quad u_n = 0. \tag{1.14}$$

Матрица коэффициентов для этой системы имеет трехдиагональный вид

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \ldots & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & \ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \ldots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \ldots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} h^2 f(x_1) \\ h^2 f(x_2) \\ h^2 f(x_3) \\ h^2 f(x_4) \\ \ldots \\ h^2 f(x_{n-2}) \\ h^2 f(x_{n-1}) \end{pmatrix}. \tag{1.15}$$

1.3 Выбор базисных функций

Проведем некоторый сравнительный анализ метода, использованного для построения слабого решения, и конечно-разностного метода (аналогично можно сделать и для метода Галеркина). В дальнейшем, если не оговорено противное, и для метода Галеркина, и для метода построения слабого решения, изложенного в п. 1.2.1, будем использовать общее название — *проекционные методы*, имея ввиду тот факт, что при соответствующих требованиях на гладкость и проведении интегрирования по частям матрицы A_{ik} и $\tilde{A}_{ik}$ будут, конечно же, совпадать.

При построении приближенного решения u^h по формуле (1.5) необходимо задавать набор базисных функций φ_1, φ_2,..., φ_{n-1}, удовлетворящих условиям (1.6). Если какая-либо дополнительная информация о задаче неизвестна, то обычно в качестве такого набора выбирают полиномиальные или тригонометрические функции. Например, для рассматриваемой задачи это могут быть

$$\varphi_1(x) = x(1-x), \quad \varphi_2(x) = x^2(1-x)^2, \ldots$$

или

$$\varphi_k(x) = \sin \pi k x.$$

Вычислительная практика показывает, что при удачном выборе базисных функций бывает достаточно ограничиться их небольшим количеством.

В общем же случае, когда число n велико, для определения коэффициентов c_k придется решать систему линейных уравнений (1.7) с большой размерностью $(n-1)\times(n-1)$ и сильно заполненной матрицей A_{ik}, определенной формулами (1.8). При решении такой системы могут возникнуть значительные трудности — большие погрешности при вычислении, увеличение времени расчетов и т. п.

Напротив, при использовании конечно-разностной схемы (1.14) матрица в (1.15) трехдиагональна и для нахождения решения $u(x_0)$, $u(x_1)$,..., $u(x_n)$ существуют эффективные алгоритмы. Более того, значения функции $u(x)$ непосредственно определяются в точках x_k сразу же после решения системы (1.14), тогда как для нахождения приближенного решения по формуле (1.5) следует сначала, решая (1.7), найти c_k, и лишь затем использовать формулу (1.5).

На самом деле такие преимущества конечно-разностный метод имеет лишь для сравнительно простых уравнений, краевых условий и областей, в которых решается задача. Уже в двумерном случае использование конечно-разностного метода приводит к серьезным проблемам при конструировании краевых условий в областях сложной формы (например, отличных от прямоугольника). Кроме того, часто возникают проблемы сохранения различных свойств исходных задач.

Поясним это на следующем примере. Рассмотрим связанный с задачей (1.1), (1.2) дифференциальный оператор L (строго говоря, в определение следует еще включать требования гладкости)

$$(Lu)(x) \stackrel{def}{=} -\frac{d}{dx}\left(p(x)\frac{du(x)}{dx}\right)+q(x)u(x), \quad 0<x<1, \tag{1.16}$$

$$u(0)=0, \quad u(1)=0.$$

Дифференциальный оператор называется *симметричным*, если для любых $u(x)$, $v(x)$ из допустимого класса функций, в частности, удовлетворяющих краевым условиям (1.2) и требуемым условиям гладкости, выполнены соотношения

$$(Lu,v)=(u,Lv), \tag{1.17}$$

$$(Lu,v) \stackrel{def}{=} \int_0^1 \left\{-\frac{d}{dx}\left(p(x)\frac{du(x)}{dx}\right)+q(x)u(x)\right\} v(x)\,dx. \tag{1.18}$$

Введем также обозначение, которое будем называть ***билинейной формой*** (т. е. *линейной* по u и v)

$$A(u,v) \stackrel{def}{=} \int_0^1 \left(p\frac{du}{dx}\frac{dv}{dx}+quv\right)dx. \tag{1.19}$$

Очевидно, что условие симметричности (1.17) для задачи (1.1), (1.2) выполнены (см. п. 1.1)

$$(Lu,v)=A(u,v), \quad (u,Lv)=A(v,u), \quad A(u,v)=A(v,u).$$

Аналогом свойства симметричности в конечномерном случае, естественно, является симметричность матрицы системы линейных уравнений (1.7). С учетом введенного обозначения (1.19) легко убедиться, что матрица (1.7) симметрична

$$A_{ik} = A(\varphi_i, \varphi_k), \quad A_{ki} = A(\varphi_k, \varphi_i), \quad A_{ik} = A_{ki}.$$

Таким образом, при построении приближенного решения такое важное свойство исходной задачи, как симметричность, сохраняется. Особенно подчеркнем, что *матрица A_{ik} будет симметрична при любом выборе базисных функций.*

По иному обстоит дело в случае конечно-разностного метода. Опуская довольно утомительные выкладки, приведем систему линейных уравнений для приближенного решения задачи (1.13) в случае, когда при аппроксимации не делается предположение об одинаковой длине отрезков $[x_k, x_{k-1}]$, $k = 1, \ldots, n-1$ (ср. с (1.14))

$$-\alpha_k u_{k-1} + 2u_k - \beta_k u_{k+1} = (x_k - x_{k-1})(x_{k+1} - x_k) f(x_k),$$
$$u_0 = 0, \quad u_n = 0, \quad \alpha_k = \frac{2(x_k - x_{k-1})}{(x_{k+1} - x_{k-1})}, \quad \beta_k = \frac{2(x_{k+1} - x_k)}{(x_{k+1} - x_{k-1})}.$$

Теперь соответствующая матрица коэффициентов не является симметричной (ср. с (1.15)), хотя, по-прежнему, остается трехдиагональной

$$\begin{pmatrix} 2 & -\beta_1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ -\alpha_2 & 2 & -\beta_2 & 0 & \ldots & 0 & 0 & 0 \\ 0 & -\alpha_3 & 2 & -\beta_3 & \ldots & 0 & 0 & 0 \\ 0 & 0 & -\alpha_4 & 2 & \ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & -\alpha_{n-2} & 2 & -\beta_{n-2} \\ 0 & 0 & 0 & 0 & \ldots & 0 & -\alpha_{n-1} & 2 \end{pmatrix}.$$

Приведенный пример показывает, что проекционный метод построения слабого решения является предпочтительнее конечно-разностного метода, т. к. он позволяет автоматически сохранять такое важное свойство задачи, как симметричность.

1.3.1 Финитные базисные функции

Оказывается, что существует эффективный способ устранения «главного недостатка» проекционных методов — сильной заполненности матрицы системы линейных уравнений. Специальный выбор базисных функций позволяет сделать матрицу A_{ik} сильно разреженной, а во многих случаях трехдиагональной, блочно-диагональной или ленточной. Более того, это можно сделать не только для одномерных задач, но и в случаях многомерных задач в области с достаточно сложной геометрической формой.

Иными словами, возможно сохранить все преимущества проекционных методов и конечно-разностных методов (разреженность матриц). Алгоритмы, позволяющие это осуществить, уместно называть ***проекционно-сеточными методами*** (проекционно-разностными, вариационно-разностными). Это другое название ***метода конечных элементов***, предложенное, в частности, в [3].

Напомним некоторые определения.

Определение 1.3.1. ***Носителем функции*** *называется замкнутая область, вне которой функция тождественно обращается в нуль. Для носителя функции* $\varphi(x)$ *используется обозначение:* $\operatorname{supp}\varphi$.

Определение 1.3.2. *Функция называется* ***финитной****, если ее носитель является ограниченной областью (более точно, компактной).*

В методе конечных элементов в качестве базисных функций предлагается выбирать финитные функции с размерами носителей, меньшими (как правило, существенно меньшими), чем размеры области, в которой рассматривается решаемая задача.

Для решения задачи (1.4) возможен следующий способ выбора базисных функций. Разобьем отрезок $[0,1]$ на интервалы $[x_{k-1}, x_k]$, $k = 1, \ldots, n$, $x_0 = 0$, $x_n = 1$ (длины интервалов не предполагаются одинаковыми, см. рис. 1.1).

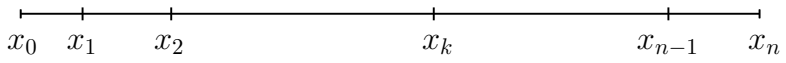

Рис. 1.1. Разбиение отрезка

В качестве $\varphi_k(x)$, $k = 1, \ldots, n-1$ выберем *финитные кусочно-линейные функции*, носителем которых будет отрезок $[x_{k-1}, x_{k+1}]$ (см. рис. 1.2)

$$\varphi_k(x) = \begin{cases} 0, & x < x_{k-1}, \\ \dfrac{x - x_{k-1}}{x_k - x_{k-1}}, & x_{k-1} < x < x_k, \\ \dfrac{x - x_{k+1}}{x_k - x_{k+1}}, & x_k < x < x_{k+1}, \\ 0, & x > x_{k+1}. \end{cases} \tag{1.20}$$

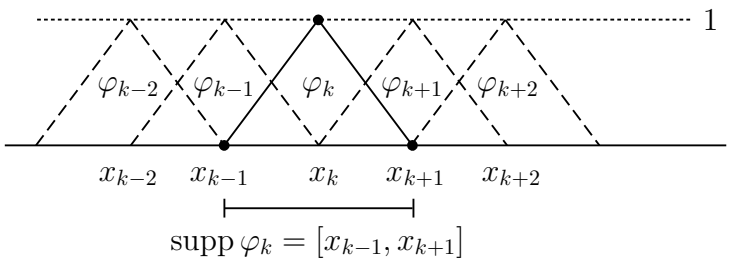

Рис. 1.2. Набор базисных функций $\varphi_k(x)$

Обычно носители базисных функций называют ***конечными элементами***. Часто тот же самый термин используется и для самих базисных функций.

Подчеркнем, что все функции $\varphi_k(x)$ $k = 1, \ldots, n-1$ удовлетворяют условиям (1.6).

Сделаем ряд важных замечаний.

1. Носители базисных функций полностью заполняют отрезок $[0, 1]$, т. е.

$$\bigcup_{k=1}^{n-1} \operatorname{supp} \varphi_k = [0, 1]. \tag{1.21}$$

Иными словами, конечные элементы полностью покрывают область, в которой разыскивается решение задачи. В противном случае, нашлась бы такая точка отрезка $[0, 1]$, в которой приближенное решение $u^h(x)$, задаваемое формулой (1.5), было бы неопределено.

2. Пересечения носителей различных базисных функций не являются пустыми, но таких пересечений достаточно мало. Более точно (очевидно, что при указании области изменения индексов предполагается, что они лежат в интервале от 1 до $n-1$)

$$\operatorname{supp} \varphi_k \bigcap \operatorname{supp} \varphi_i \neq \varnothing, \quad i = k-1, k, k+1, \tag{1.22}$$

$$\operatorname{supp} \varphi_k \bigcap \operatorname{supp} \varphi_i \neq \varnothing, \quad i = k-2, k+2, \tag{1.23}$$

$$\operatorname{mes}(\operatorname{supp} \varphi_k \bigcap \operatorname{supp} \varphi_i) = 0,$$

$$\operatorname{supp} \varphi_k \bigcap \operatorname{supp} \varphi_i = \varnothing, \quad i < k-2, \quad i > k+2. \tag{1.24}$$

Соотношения (1.22)–(1.24) являются наиболее важными для эффективности метода конечных элементов. В конечном итоге, именно они определяют структуру матрицы A_{ik}, задаваемой соотношениями (1.8). Благодаря (1.23), (1.24), большинство интегралов в (1.8) обращается в нуль и матрица A_{ik} становится сильно разреженной.

3. Справедливо соотношение

$$u^h(x_s) = \sum_{k=1}^{n-1} c_k \varphi_k(x_s) = c_s. \tag{1.25}$$

Таким образом, коэффициенты c_s — это значения приближенного решения в точках x_s. В частности, это означает, что при решении системы (1.7) приближенное решение $u^h(x)$ **в узлах сетки** будет получено сразу, без использования формулы (1.5). Напомним, что в случае конечно-разностного метода, приближенное решение в узлах сетки также получалось непосредственно после решения системы линейных уравнений (см. (1.14)).

4. Базисные функции $\varphi_k(x)$ выбраны кусочно-линейными из следующих соображений. Во-первых, кусочно-линейные функции допустимо использовать в задаче (1.4), т. к. производные $\varphi_k'(x)$ в этом случае кусочно-постоянны и интегрирование в (1.4) и последующих соотношениях (1.6)–(1.8) может быть выполнено.

Во-вторых, кусочно-линейные функции являются наиболее простейшими финитными кусочно-полиномиальными функциями, использование которых позволяет учесть все члены в интегралах (1.8). Нельзя выбирать в качестве $\varphi_k(x)$ кусочно-постоянные функции, т. к. в этом случае $\varphi_k'(x)$ обращаются в нуль и члены вида $\varphi_k'(x)\varphi_i'(x)$ будут отсутствовать в интегралах (1.8), что приведет к исчезновению части информации об исходной задаче. По этой же причине, нельзя использовать кусочно-линейные функции непосредственно в выражениях для вычисления матрицы $\tilde{A}_{ik}$, т. к. интегралы (1.11) содержат вторые производные базисных функций.

На самом деле, ситуация более сложная, т. к. при дифференцировании кусочно-постоянных функций будут возникать δ-функции Дирака и при определенных дополнительных предположениях кусочно-постоянные базисные функции все-таки могут быть использованы.

1.3.2 Вычисление элементов матрицы A_{ik} и вектора b_i

Уже на основании свойств носителей базисных функций (конечных элементов) (1.22)–(1.24) можно сделать вывод о том, что матрица A_{ik} будет трехдиагональной. Однако, с учетом явных выражений (1.20) для базисных функций полезно привести конкретные формулы для элементов матрицы A_{ik} и вектора b_i системы (1.7).

С учетом (1.24) ясно, что носитель функции φ_k не пересекается с носителями функций φ_i при $i > k+2$ и $i < k-2$ (см. также рис. 1.2). Это означает, что произведение $\varphi_i\varphi_k = 0$ при $i > k+2$ и $i < k-2$. Это же относится и к производным функции φ_i, заданной формулой (1.20). На самом деле, из (1.20) следует даже бóльшее — $\varphi_i\varphi_k = 0$ при $i \geqslant k+2$ и $i \leqslant k-2$.

Учитывая сказанное, запишем формулы для вычисления интегралов, входящих в (1.8). Заметим, что вместо интегрирования по всей области $[0,1]$ можно ограничиться интегрированием лишь по области, являющейся объединением носителей подинтегральных функций.

Имеем следующие соотношения (пересечение носителей функций либо пусто, либо имеет меру нуль)

$$\int_0^1 p\varphi_k'\varphi_i'\,dx = 0, \quad i \geqslant k+2, \quad i \leqslant k-2. \tag{1.26}$$

Далее, заменяем интегрирование по всей области интегрированием по пересечению носителей (см. рис. 1.2)

$$\int_0^1 p\varphi_k'\varphi_k'\,dx = \int_{x_{k-1}}^{x_{k+1}} p\varphi_k'\varphi_k'\,dx,$$

$$\int_0^1 p\varphi_k'\varphi_{k+1}'\,dx = \int_{x_k}^{x_{k+1}} p\varphi_k'\varphi_{k+1}'\,dx, \quad \int_0^1 p\varphi_k'\varphi_{k-1}'\,dx = \int_{x_{k-1}}^{x_k} p\varphi_k'\varphi_{k-1}'\,dx. \tag{1.27}$$

Аналогично для других членов, входящих в формулу (1.8)

$$\int_0^1 q\varphi_k\varphi_i\,dx = 0, \quad k-2 \leqslant i \leqslant k+2, \tag{1.28}$$

$$\int_0^1 q\varphi_k\varphi_k\,dx = \int_{x_{k-1}}^{x_{k+1}} q\varphi_k\varphi_k\,dx, \quad \int_0^1 q\varphi_k\varphi_{k+1}\,dx = \int_{x_k}^{x_{k+1}} q\varphi_k\varphi_{k+1}\,dx,$$

$$\int_0^1 q\varphi_k\varphi_{k-1}\,dx = \int_{x_{k-1}}^{x_k} q\varphi_k\varphi_{k-1}\,dx, \quad \int_0^1 f\varphi_i\,dx = \int_{x_{i-1}}^{x_{i+1}} f\varphi_i\,dx. \tag{1.29}$$

Таким образом, матрица A_{ik} является трехдиагональной матрицей.

Пример 1.1. В простых случаях интегралы (1.27), (1.29) легко вычисляются. Рассмотрим краевую задачу (см. (1.13))

$$-u''(x) = 1, \quad u(0) = 0, \quad u(1) = 0.$$

Сравнивая с (1.1), запишем

$$p = 1, \quad q = 0, \quad f(x) = 1.$$

Производя вычисление интегралов, получим следующие результаты (в окончательном ответе, для простоты, полагаем $x_k = kh$)

$$\int_{x_{k-1}}^{x_{k+1}} \varphi_k'\varphi_k'dx = \int_{x_{k-1}}^{x_k} \varphi_k'\varphi_k'dx + \int_{x_k}^{x_{k+1}} \varphi_k'\varphi_k'dx = \int_{x_{k-1}}^{x_k} \frac{1}{(x_k - x_{k-1})^2}dx + \int_{x_k}^{x_{k+1}} \frac{1}{(x_k - x_{k+1})^2}dx =$$

$$= \frac{x_k - x_{k-1}}{(x_k - x_{k-1})^2} + \frac{x_{k+1} - x_k}{(x_{k+1} - x_k)^2} = \frac{1}{x_k - x_{k-1}} + \frac{1}{x_{k+1} - x_k} = \frac{2}{h}.$$

$$\int_{x_k}^{x_{k+1}} \varphi_k'\varphi_{k+1}'dx = \int_{x_k}^{x_{k+1}} \frac{1}{x_k - x_{k+1}}\frac{1}{x_{k+1} - x_k}dx = \frac{x_{k+1} - x_k}{(x_k - x_{k+1})(x_{k+1} - x_k)} = \frac{1}{x_k - x_{k+1}} = -\frac{1}{h}.$$

$$\int_{x_{k-1}}^{x_k} \varphi_k'\varphi_{k-1}'dx = \int_{x_{k-1}}^{x_k} \frac{1}{x_k - x_{k-1}}\frac{1}{x_{k-1} - x_k}dx = \frac{x_k - x_{k-1}}{(x_k - x_{k-1})(x_{k-1} - x_k)} = \frac{1}{x_{k-1} - x_k} = -\frac{1}{h}.$$

$$\int_0^1 \varphi_i\,dx = \int_{x_{i-1}}^{x_{i+1}} \varphi_i\,dx = \int_{x_{i-1}}^{x_i} \varphi_i\,dx + \int_{x_i}^{x_{i+1}} \varphi_i\,dx.$$

$$\int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{x_i - x_{i-1}}\,dx = \left| \begin{array}{c} x - x_{i-1} = s \\ dx = ds \end{array} \right| = \frac{1}{x_i - x_{i-1}} \int_0^{x_i - x_{i-1}} s\,ds = \frac{1}{x_i - x_{i-1}} \cdot \frac{s^2}{2}\Big|_0^{x_i - x_{i-1}} =$$

$$= \frac{(x_i - x_{i-1})^2}{2(x_i - x_{i-1})} = \frac{x_i - x_{i-1}}{2} = \frac{h}{2}.$$

Аналогично, имеем

$$\int_{x_i}^{x_{i+1}} \varphi_i \, dx = \frac{h}{2}.$$

Используя полученные соотношения, запишем систему (1.7)

$$\underbrace{A_{ii-1}}_{-1/h} c_{i-1} + \underbrace{A_{ii}}_{2/h} c_i + \underbrace{A_{ii+1}}_{-1/h} c_{i+1} = \underbrace{b_i}_{h}$$

или

$$-\frac{c_{i-1} - 2c_i + c_{i+1}}{h^2} = 1.$$

С учетом (1.25) запишем

$$-\frac{u^h(x_{i-1}) - 2u^h(x_i) + u^h(x_{i+1})}{h^2} = 1.$$

Сравнивая полученное выражение с (1.14), видим, что вновь получена обычная конечно-разностная схема (при $f(x) = 1$).

Конечно, такие простые соотношения получаются лишь для простых примеров. В более общем случае для вычисления интегралов (1.27), (1.29) следует использовать те или иные квадратурные формулы. От правильного выбора квадратурной формулы будет зависеть точность вычисления интеграла и, в конечном итоге, точность представления матрицы A_{ik}. Заметим, что FreeFem++ предлагает специальные возможности для использования различных квадратурных формул (см. гл. 18 и [1]).

1.4 Естественные и главные краевые условия

При использовании метода конечных элементов важное значение имеет правильный учет краевых условий рассматриваемой краевой задачи. В случае задачи (1.1), (1.2) при построении приближенного решения в виде (1.5) выполнение краевых условий (1.2) обеспечивалось выбором базисных функций. Действительно, для базисных функций требовалось выполнение краевых условий (1.6), которые и были соблюдены при задании φ_k соотношениями (1.20).

В общем случае для задач с краевыми условиями, отличными от (1.2), не всегда удается удовлетворить краевым условиям, выбирая подходящие базисные функции. Более точно, такой выбор может быть сопряжен с большими техническими трудностями — функции могут оказаться слишком сложными, матрица системы линейных уравнений (1.7) не будет сильно разреженной и пр.

Рассмотрим следующую краевую задачу, которая отличается от задачи (1.1), (1.2) (при $p = 1$, $q = 0$) лишь заданием краевого условия третьего рода при $x = 0$

$$-u''(x) = f(x), \qquad 0 < x < 1, \tag{1.30}$$

$$u'(0) - \alpha u(0) = 0, \qquad u(1) = 0, \tag{1.31}$$

где $f(x)$ — известная функция, α — заданное число.

Следуя процедуре, описанной в п. 1.1, умножим (1.30) на $v(x)$ и проинтегрируем от 0 до 1 с использованием интегрирования по частям

$$\int_0^1 \frac{du}{dx}\frac{dv}{dx}\,dx - \frac{du}{dx}v\Big|_0^1 = \int_0^1 fv\,dx. \tag{1.32}$$

Потребуем, чтобы $v(x)$ при $x = 1$ удовлетворяла тому же краевому условию, что и $u(x)$, т. е.

$$v(1) = 0. \tag{1.33}$$

Заметим, что нельзя требовать выполнения условия $v(0) = 0$, т. к. в этом случае (1.32) примет вид (1.4) (при $p = 1$, $q = 0$), что, очевидно, будет соответствовать решению задачи (1.30), (1.31) с краевым условием $u(0) = 0$ вместо краевого условия $u'(0) - \alpha u(0) = 0$.

Соотношение (1.32) с учетом (1.33) примет вид

$$\int_0^1 \left(\frac{du}{dx}\frac{dv}{dx} - fv\right)dx + u'(0)v(0) = 0, \quad v(1) = 0.$$

Используя краевое условие (1.31) при $x = 0$, получим (ср. с (1.4))

$$\int_0^1 \left(\frac{du}{dx}\frac{dv}{dx} - fv\right)dx + \alpha u(0)v(0) = 0, \quad v(1) = 0. \tag{1.34}$$

Функцию $u(x)$, являющуюся решением задачи (1.34), будем называть ***слабым решением*** задачи (1.30), (1.31).

Приближенное решение задачи (1.34) строим в виде, аналогичном (1.5), ***добавив к базисным функциям еще одну дополнительную функцию*** $\varphi_0(x)$, линейно не зависящую от прежних базисных функций

$$u^h(x) = \sum_{k=0}^{n-1} c_k\varphi_k(x). \tag{1.35}$$

Необходимость введения дополнительной базисной функции достаточно очевидна. Сохранив прежний набор базисных функций, невозможно удовлетворить краевым условиям при $x = 0$, т. к. в силу (1.6) $\varphi_k(0) = 0$, $\varphi_k(1) = 0$, $k = 1, \ldots, n-1$. Понятно также, что, выбирая $\varphi_0(x)$, следует потребовать выполнения условия $\varphi_0(1) = 0$. В противном случае нарушится выполнение краевого условия при $x = 1$ для приближенного решения, т. е. будет $u^h(1) \neq 0$. (Конечно, кроме тривиального случая, когда $c_0 = 0$, что соответствует просто прежнему набору базисных функций.)

Зададим дополнительную базисную функцию $\varphi_0(x)$ в виде *финитной кусочно-линейной функции* с носителем $\operatorname{supp}\varphi_0 = [x_0, x_1]$ (ср. с (1.20) и см. рис. 1.3 и 1.2)

$$\varphi_0(x) = \begin{cases} 1, & x = x_0, \\ \dfrac{x - x_1}{x_0 - x_1}, & x_0 < x < x_1, \\ 0, & x > x_1. \end{cases} \tag{1.36}$$

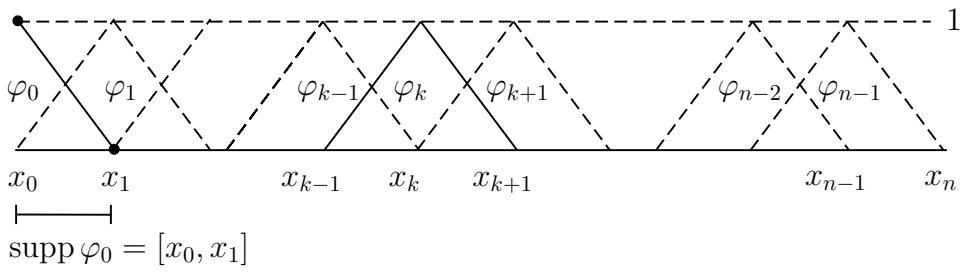

Рис. 1.3. Набор базисных функций $\varphi_k(x)$

Выбор $\varphi_0(x)$ в форме (1.36) продиктован следующими соображениями. Новая базисная функция не должна «сильно отличаться» от прежних, по-крайней мере, принадлежать тому же классу функций (в данном случае, быть кусочно-линейной). Желательно сохранение условия (1.25), т. е. $u^h(x_0) = c_0$, что приводит к условию $\varphi_0(0) = 1$. Желательно также сохранение условий, аналогичных (1.22)–(1.24):

$$\operatorname{supp}\varphi_0 \bigcap \operatorname{supp}\varphi_1 \neq \varnothing, \quad \operatorname{supp}\varphi_0 \bigcap \operatorname{supp}\varphi_i = \varnothing, \quad i > 2, \tag{1.37}$$

$$\operatorname{supp}\varphi_0 \bigcap \operatorname{supp}\varphi_2 \neq \varnothing, \quad \operatorname{mes}(\operatorname{supp}\varphi_0 \bigcap \operatorname{supp}\varphi_2) = 0.$$

Особенно подчеркнем, что функция $\varphi_0(x)$ не удовлетворяет краевому условию (1.31) при $x = 0$. Заманчиво было бы, конечно, выбрать $\varphi_0(x)$ в виде $\varphi_0(x) = 1 + \alpha x$, например, на отрезке $[x_0, x_1]$, и $\varphi_0(x) = 0$ — вне этого отрезка. Это привело бы к $\varphi_0'(0) - \alpha\varphi_0(0) \equiv 0$. Однако, при этом теряется некоторая «универсальность» базисных функций — φ_0 будет зависеть от параметра задачи α. Более того, детальный анализ показывает, что такой выбор существенно ухудшает процесс построения решения.

После того как выбор дополнительной базисной функции осуществлен, схема построения системы линейных уравнений для определения c_k будет такая же, как в п. 1.2.1.

Подставляя (1.35) в (1.34), получим ($k = 0, 1, \ldots, n-1$)

$$\int_0^1 \left(\sum_{k=0}^{n-1} c_k \varphi_k' v' - fv \right) dx + \alpha \sum_{k=0}^{n-1} c_k \varphi_k(0) v(0) = 0.$$

Выбирая в качестве $v(x)$ набор функций $\varphi_i(x)$, $i = 0, \ldots, n-1$, имеем

$$\int_0^1 \left(\sum_{k=0}^{n-1} c_k \varphi_k' \varphi_i' - f\varphi_k \right) dx + \alpha \sum_{k=0}^{n-1} c_k \varphi_k(0) \varphi_i(0) = 0, \quad i = 0, \ldots, n-1.$$

Для определения c_k имеем систему линейных уравнений

$$\sum_{k=0}^{n-1} B_{ik} c_k = b_i, \quad i = 0, \ldots, n-1, \tag{1.38}$$

где введены следующие обозначения

$$\int\limits_0^1 \varphi_i'\varphi_k' dx + \alpha\varphi_i(0)\varphi_k(0) = B_{ik}, \quad \int\limits_0^1 f\varphi_i\, dx = b_i, \quad i,k = 0,\ldots,n-1. \tag{1.39}$$

Нетрудно показать, что элементы матрицы B_{ik} для $i,k = 1,\ldots,n-1$ совпадают с элементами матрицы A_{ik} (при $p = 1$, $q = 0$), определяемыми формулами (1.8). Действительно, базисные функции $\varphi_1, \ldots, \varphi_{n-1}$ остались прежними и $\varphi_k(0) = 0$, $k = 1,\ldots,n-1$.

В силу (1.37) при вычислении интегралов в (1.39) получим (пересечение носителей соответствующих функций имеет нулевую меру)

$$B_{0k} = 0, \quad B_{k0} = 0, \quad k = 2,\ldots,n-1.$$

Используя вид функций φ_0 и φ_1 (см. (1.20) и (1.36)) в случае, когда длины всех отрезков $[x_k, x_{k+1}]$ одинаковы и равны h, получим

$$B_{00} = \frac{1+\alpha h}{h}, \quad B_{10} = B_{01} = -\frac{1}{h}.$$

Таким образом, матрица B_{ik} размерности $n \times n$ отличается от матрицы A_{ik} размерности $(n-1)\times(n-1)$ лишь левым столбцом и верхней строкой. Система уравнений (1.38), для удобства умноженная на h, в матричной форме имеет вид (выделена часть, соответствующая матрице A_{ik})

$$hBc = \left(\begin{array}{c|cccccc} 1+\alpha h & -1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ \hline -1 & 2 & -1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \ldots & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & \ldots & 0 & 0 & 0 \\ \ldots & \multicolumn{7}{c}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ 0 & 0 & 0 & 0 & \ldots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \ldots & 0 & -1 & 2 \end{array}\right)\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \ldots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} hb_0 \\ hb_1 \\ hb_2 \\ hb_3 \\ \ldots \\ hb_{n-2} \\ hb_{n-1} \end{pmatrix}.$$

Интересно посмотреть, чему соответствует первое уравнение системы (1.38). При $i = 0$, с учетом $c_0 = u(x_0) = u(0)$, $c_1 = u(x_1) = u(x_0+h) = u(h)$, имеем

$$(1+\alpha h)u(0) - u'(h) = h\int\limits_0^h f(x)\left(\frac{h-x}{h}\right)dx = \int\limits_0^h f(x)(h-x)\,dx.$$

Раскладывая в ряд при $h \to 0$, выводим

$$(\alpha u(0) - u'(0))h - \frac{1}{2}u''(0)h^2 + \mathcal{O}(h^3) = \frac{1}{2}f(0)h^2 + \mathcal{O}(h^3).$$

Если предполагать выполнение уравнения (1.30) при $x = 0$, то имеем $u''(0) = -f(0)$, и получится, что краевое условие (1.31) при $x = 0$ выполняется с точностью до членов порядка $\mathcal{O}(h^2)$

$$u'(0) - \alpha u(0) = \mathcal{O}(h^2), \quad h \to 0.$$

Этот факт служит косвенным подтверждением правильности выбора базисной функции $\varphi_0(x)$.

Рассмотренные примеры показывают, что имеются различные возможности удовлетворения краевым условиям. В первом случае все краевые условия для задачи (1.1), (1.2) были выполнены *за счет выбора базисных функций* (см. формулу (1.20)). Во втором случае, для задачи (1.30), (1.31) краевое условие (1.31) при $x = 1$ по прежнему выполнено за счет выбора базисных функций. Краевое условие (1.31) при $x = 0$ учитывается «естественным образом». Имеется в виду следующее: в результате преобразования уравнений в (1.34) возникает дополнительный член $\alpha u(0)v(0)$. Затем к базисным функциям добавляется новая функция φ_0 (не удовлетворяющая краевому условию!) и задача приводится к решению системы линейных уравнений (1.38) с матрицей B_{ik}, включающей дополнительные, по сравнению с матрицей A_{ik}, члены $\alpha\varphi_i(0)\varphi_k(0)$.

Определение 1.4.1. *Краевые условия, которым можно удовлетворить за счет выбора базисных функций, называются* ***главными краевыми условиями.***

Определение 1.4.2. *Краевые условия, удовлетворение которых возможно за счет преобразования задачи, а не за счет выбора базисных функций, называются* ***естественными краевыми условиями.***

Таким образом, в задаче (1.1), (1.2) оба краевых условия (1.2) являются ***главными***. В задаче (1.30), (1.31) краевое условие (1.31) при $x = 1$ будет ***главным***, а краевое условие (1.31) при $x = 0$ является ***естественным***.

Понятие о главных и естественных краевых условиях является весьма важным для метода конечных элементов. В случае естественных краевых условий, в некотором смысле, можно не заботиться о выборе базисных функций. В случае же главных краевых условий приходится ставить дополнительные ограничения на выбор базисных функций. Заметим, что большое количество примеров задач с естественными и главными краевыми условиями имеется в [3], где подробно разъясняются всевозможные математические проблемы, возникающие в связи с численной реализацией различных граничных условий.

Скажем несколько слов о случае, когда для задачи (1.30), (1.31) параметр $\alpha = 0$ (краевое условие второго рода). Никаких проблем в данном случае в связи с численной реализацией не возникает — выражения (1.39) остаются справедливыми и при $\alpha = 0$. На первый взгляд может показаться, что задача (1.4) (при $p = 1$, $q = 0$) и задача (1.34) при $\alpha = 0$ одинаковы. На самом деле, это, конечно же, разные задачи. В случае (1.4) требуется выполнение двух условий: $v(0) = 0$ и $v(1) = 0$, тогда как для (1.34) нужно выполнение лишь одного условия $v(1) = 0$ и значение $v(0)$ может быть произвольным.

Глава 2

Метод конечных элементов в двумерном случае

С точки зрения подхода к построению решения метод конечных элементов для пространственно многомерных задач ничем не отличается от одномерного случая. Также, как и в одномерном случае, используется проекционный метод и получается аналог, например, задачи (1.4). Конструируется набор финитных базисных функций и приближенное решение разыскивается в виде их линейной комбинации аналогично (1.5). Затем получается система линейных уравнений вида (1.7) с достаточно разреженной матрицей и решение этой системы определяет значения приближенного решения в некоторых точках области.

Серьезные математические и вычислительные трудности возникают реально на этапе реализации метода конечных элементов, например, при триангуляции плоской области, при выборе подходящего для конкретной задачи набора финитных базисных функций, при решении систем линейных уравнений большой размерности (несмотря на сильную разреженность матриц, это достаточно серьезная проблема) и при доказательстве сходимости приближенного решения к точному решению задачи.

Конечно, вышесказанное в первую очередь относится к решению сложных задач математической физики для областей со сложной геометрической формой, например, к решению уравнений Навье–Стокса для несжимаемой жидкости при больших числах Рейнольдса, к решению уравнений переноса заряженных примесей электрическим полем, к решению уравнений магнитной гидродинамики и т. п. В случае же таких «простых» задач, как, например, краевая задача для уравнения Лапласа в прямоугольной области, все проблемы метода конечных элементов в настоящее время практически решены и эта задача идеально подходит для изложения основ метода в двумерном случае (см. любой учебник по методу конечных элементов, например, [3, 4]).

Ниже, как раз на примере задачи Дирихле для уравнения Лапласа в квадратной области, показан алгоритм построения решения — задача переформулирована для построения слабого (обобщенного) решения, рассмотрен пример разбиения области на конечные элементы и описан один из вариантов выбора финитных базисных функций.

2.1 Задача Дирихле для уравнения Лапласа. Слабое решение

Рассмотрим задачу Дирихле для уравнения Лапласа (первую краевую задачу) для определения функции $u(x,y)$ в случае квадратной области:

$$-\Delta u = f(x,y), \quad (x,y) \in D = (0,1)\times(0,1), \qquad \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \quad (2.1)$$

$$u\big|_\Gamma = 0. \quad (2.2)$$

Здесь $f(x,y)$ — заданная в области D функция, $\Gamma = \partial D$ — граница области D (стороны квадрата), Δ — оператор Лапласа.

Краевые условия выбраны однородными для простоты изложения.

Прежде чем записать задачу (2.1), (2.2) в форме, аналогичной (1.4), напомним некоторые основные формулы и обозначения.

Формула Грина (аналог интегрирования по частям для одномерного случая) в двумерном случае имеет вид

$$\iint\limits_D v\Delta u\,dx\,dy = -\iint\limits_D \nabla u\cdot\nabla v\,dx\,dy + \int\limits_\Gamma v\frac{\partial u}{\partial n}\,ds, \quad (2.3)$$

где

$$\nabla v = \left(\frac{\partial v}{\partial x}, \frac{\partial v}{\partial y}\right), \quad \nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right), \quad \nabla u\cdot\nabla v = \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y}, \quad (2.4)$$

$$\boldsymbol{n} = (n_x, n_y), \quad \frac{\partial v}{\partial n} = \boldsymbol{n}\cdot\nabla v = n_x\frac{\partial v}{\partial x} + n_y\frac{\partial v}{\partial y}. \quad (2.5)$$

Здесь ∇u — градиент функции u (другое обозначение $\operatorname{grad} u$), $\boldsymbol{n}$ — вектор нормали (внешней к области D) с компонентами n_x, n_y, $\partial v/\partial n$ — производная по нормали.

Для вывода соотношения аналогичного (1.4) умножим (2.1) на функцию $v(x,y)$ и проинтегрируем по области D

$$-\iint\limits_D v\Delta u\,dx\,dy = \iint\limits_D vf\,dx\,dy. \quad (2.6)$$

Используя формулу Грина (2.3), получим

$$\iint\limits_D \nabla u\cdot\nabla v\,dx\,dy - \int\limits_\Gamma v\frac{\partial u}{\partial n}\,ds - \iint\limits_D fv\,dx\,dy = 0. \quad (2.7)$$

Потребуем выполнения для функции $v(x,y)$ краевого условия (2.2)

$$v\big|_\Gamma = 0. \quad (2.8)$$

Окончательно, из (2.7) с учетом (2.8) получим аналог (1.4):

$$\forall\, v, \quad \iint\limits_D (\nabla u\cdot\nabla v - fv)\,dx\,dy = 0, \quad v\big|_\Gamma = 0. \quad (2.9)$$

Соотношение (2.9) должно выполняться при ***любом*** v. Решение задачи (2.9) называется слабым (или обобщенным) решением исходной задачи (2.1), (2.2).

Напомним, что для строгой постановки задачи необходимо дополнительно указать каким функциональным пространствам принадлежат функции u, v, например, являются ли они непрерывными, дифференцируемыми и т. п. Далее считаем, что все дополнительные условия поставлены (подробнее см., например, [3, 4]).

2.2 Построение приближенного решения

Разобьем область $\overline{D} = [0,1] \times [0,1]$ на квадратные ячейки с длиной стороны h и вершинами в узлах

$$x_i = ih, \quad y_k = kh, \quad i,k = 0,\dots,n, \quad nh = 1. \tag{2.10}$$

Каждую квадратную ячейку $(x,y) \in [ih, ih+h] \times [kh, kh+h]$ дополнительно разобьем диагональю, проходящей через вершины (ih, kh) и $(ih+h, kh+h)$ (см. рис. 2.1). Таким образом, вся область $\overline{D}$ будет разбита на треугольники, которые обозначим T_m. Такое разбиение области называется ***триангуляцией*** области $\overline{D}$. В рассматриваемом случае объединение треугольников полностью покрывает область $\overline{D}$

$$\bigcup_m T_m = \overline{D}. \tag{2.11}$$

Отметим, что пересечение треугольников T между собой возможно только по их границам ∂T (сторонам и вершинам) и мера пересечений треугольников равна нулю

$$\operatorname{mes}(T_j \cap T_m) = 0, \quad j \neq m. \tag{2.12}$$

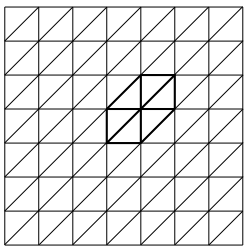

Рис. 2.1. Пример триангуляции области $\overline{D} = [0,1] \times [0,1]$

Определим вспомогательную кусочно-линейную функцию (функция Куранта, [3])

$$\varphi(\xi,\eta) = \begin{cases} 1-\xi+\eta, & 0 \leqslant \xi \leqslant 1, & \xi-1 \leqslant \eta \leqslant 0, & (T_1), \\ 1-\xi, & 0 \leqslant \xi \leqslant 1, & 0 \leqslant \eta \leqslant \xi, & (T_2), \\ 1-\eta, & 0 \leqslant \xi \leqslant 1, & \xi \leqslant \eta \leqslant 1, & (T_3), \\ 1+\xi-\eta, & -1 \leqslant \xi \leqslant 0, & 0 \leqslant \eta \leqslant 1+\xi, & (T_4), \\ 1+\xi, & -1 \leqslant \xi \leqslant 0, & \xi \leqslant \eta \leqslant 0, & (T_5), \\ 1+\eta, & -1 \leqslant \xi \leqslant 0, & -1 \leqslant \eta \leqslant \xi, & (T_6). \end{cases} \tag{2.13}$$

Здесь T_s соответствуют треугольникам на рис. 2.3.

Вид функции (2.13) показан на рис. 2.2.

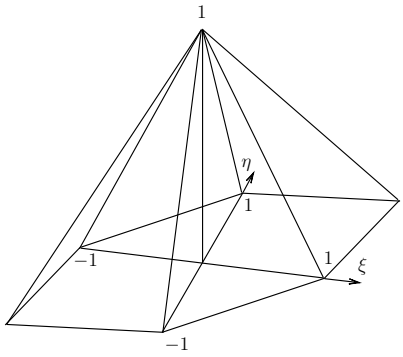

Рис. 2.2. Вид финитной кусочно-линейной базисной функции $\varphi_{ik}(x, y)$

Для каждого внутреннего узла $i, k = 1, \ldots, n-1$ определим финитную кусочно-линейную базисную функцию с носителем в виде шестиугольника S^{ik}, состоящего из треугольников с общей вершиной в точке (x_i, y_k), которые обозначим T_1, T_2, T_3, T_4, T_5, T_6 (см. рис. 2.3)

$$\varphi_{ik}(x, y) = \varphi\left(\frac{x - x_i}{h}, \frac{y - y_k}{h}\right), \quad \operatorname{supp}\varphi_{ik} = \bigcup_{s=1}^{6} T_s = S^{ik}. \tag{2.14}$$

Иногда, для того, чтобы подчеркнуть наличие у треугольников общей вершины (x_i, y_k), будет использовано обозначение T_s^{ik}.

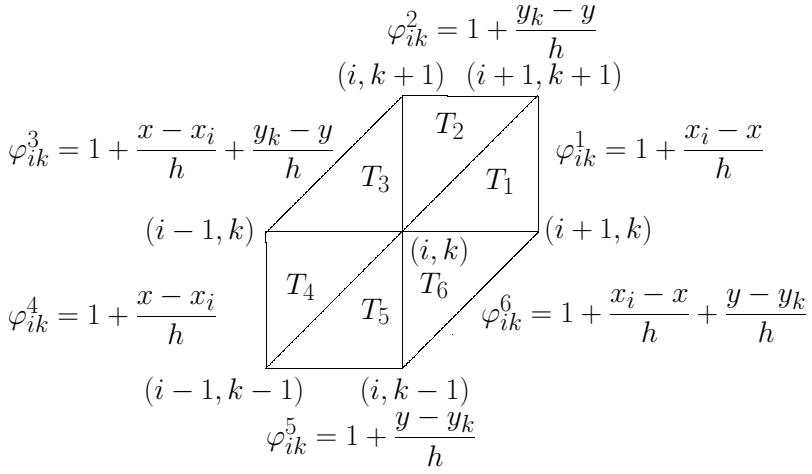

Рис. 2.3. Носитель функции $\varphi_{ik}(x, y)$ (конечный элемент) и ее значения в T_k

Значения функции $\varphi_{ik}(x, y)$ в каждом треугольнике T_s, $s = 1, \ldots, 6$ для удобства приведены на рис. 2.3 (см. также рис. 2.2). Заметим, что функция $\varphi_{ik}(x, y)$ обращается в нуль во всех узлах за исключением узла (x_i, y_k)

$$\varphi_{ik}(x_i, y_k) = 1. \tag{2.15}$$

Кроме этого, на всех сторонах шестиугольника функция $\varphi_{ik}(x,y)$, очевидно, равна нулю (см. рис. 2.2).

Приближенное решение задачи (2.9) ищем в виде линейной комбинации базисных функций по всем внутренним узлам области D

$$u^h(x,y) = \sum_{i,k=1}^{n-1} u_{ik}\varphi_{ik}(x,y), \quad u_{ik} = u^h(x_i,y_k). \tag{2.16}$$

Как уже говорилось, на границах носителя базисные функции обращаются в нуль. Задавая в (2.16) суммирование по всем внутренним узлам области D, удается удовлетворить краевым условиям (2.2) для $u^h(x,y)$ *почти на всей границе* области D. Иными словами, краевые условия (2.2), так же как и для задачи (1.1), (1.2), являются главными, т. е. их удается выполнить за счет выбора базисных функций (см. определение 1.4.1, с. 28).

Заметим, что объединение носителей всех базисных функций не покрывает полностью область D — значения $u^h(x,y)$ не определены в левом верхнем и правом нижнем треугольниках триангулированной области (см. рис. 2.1). Это означает, что на части границы краевые условия все-таки не выполнены. К счастью, можно показать, что для рассматриваемой задачи это не играет существенной роли. Однако, данный факт означает, что во избежание ошибок к выбору базисных функций следует относиться внимательно.

Далее, процедура определения неизвестных величин u_{ik} в формуле (2.16) полностью следует [4] и п. 1.2.1. Подставляя (2.16) в (2.9) и выбирая в качестве функций $v(x,y)$ базисные функции $\varphi_{pq}(x,y)$, $p,q = 1,\ldots,n-1$, получим (проекция (2.9) на линейное пространство, задаваемое базисными функциями)

$$\sum_{i,k=1}^{n-1} u_{ik} \iint\limits_D \left(\frac{\partial\varphi_{ik}}{\partial x}\frac{\partial\varphi_{pq}}{\partial x} + \frac{\partial\varphi_{ik}}{\partial y}\frac{\partial\varphi_{pq}}{\partial y} \right) dx\,dy = \int\limits_D f\varphi_{pq}\,dx\,dy. \tag{2.17}$$

Для определения u_{ik} имеем линейные алгебраические уравнения

$$\sum_{i,k=1}^{n-1} A_{pq,ik}u_{ik} = b_{pq}, \quad p,q = 1,\ldots,n-1, \tag{2.18}$$

где

$$A_{pq,ik} = \iint\limits_D \left(\frac{\partial\varphi_{ik}}{\partial x}\frac{\partial\varphi_{pq}}{\partial x} + \frac{\partial\varphi_{ik}}{\partial y}\frac{\partial\varphi_{pq}}{\partial y} \right) dx\,dy, \quad b_{pq} = \int\limits_D f\varphi_{pq}\,dx\,dy. \tag{2.19}$$

В рассматриваемом случае элементы матрицы $A_{pq,ik}$ легко определяются. Действительно, т. к. мера пересечения шестиугольника S_{pq} с шестиугольниками S_{ik} равна нулю при $i \geqslant p+2$, $k \geqslant q+2$, то

$$A_{pq,ik} = \iint\limits_D (\ldots)\,dx\,dy = \iint\limits_{S_{ik}\cap S_{pq}} (\ldots)\,dx\,dy = 0, \quad i \geqslant p+2, \quad k \geqslant q+2. \tag{2.20}$$

Далее заметим, что для любой базисной функции независимо от индексов i, k производные определяются соотношениями

$$\frac{\partial\varphi_{ik}^m}{\partial x}=0,\quad m=3,6;\quad \frac{\partial\varphi_{ik}^m}{\partial x}=-\frac{1}{h},\quad m=1,2;\quad \frac{\partial\varphi_{ik}^m}{\partial x}=\frac{1}{h},\quad m=4,5;$$
$$\frac{\partial\varphi_{ik}^m}{\partial y}=0,\quad m=2,5;\quad \frac{\partial\varphi_{ik}^m}{\partial y}=-\frac{1}{h},\quad m=3,4;\quad \frac{\partial\varphi_{ik}^m}{\partial y}=\frac{1}{h},\quad m=1,6.$$

Тогда, например, для $A_{pq,pq+1}$ с учетом того, что пересечение $\operatorname{supp}\varphi_{pq}$ и $\operatorname{supp}\varphi_{pq+1}$ является лишь пересечениями треугольников $T_4^{pq}\cap T_6^{pq+1}$ и $T_3^{pq}\cap T_1^{pq+1}$, имеем

$$A_{pq,pq+1}=\iint\limits_D\left(\frac{\partial\varphi_{pq+1}}{\partial x}\frac{\partial\varphi_{pq}}{\partial x}+\frac{\partial\varphi_{pq+1}}{\partial y}\frac{\partial\varphi_{pq}}{\partial y}\right)dx\,dy=$$
$$=\iint\limits_{T_4^{pq}\cap T_6^{pq+1}}\left(0\cdot\frac{1}{h}-\frac{1}{h}\cdot\frac{1}{h}\right)dx\,dy+\iint\limits_{T_3^{pq}\cap T_1^{pq+1}}\left(-\frac{1}{h}\cdot 0-\frac{1}{h}\cdot\frac{1}{h}\right)dx\,dy=-1.$$

Аналогично для остальных элементов получим

$$A_{pq,p-1q}=\iint\limits_{T_5^{pq}\cap T_1^{p-1q}}(\dots)\,dx\,dy+\iint\limits_{T_4^{pq}\cap T_2^{p-1q}}(\dots)\,dx\,dy=-1.$$
$$A_{pq,p-1q-1}=\iint\limits_{T_5^{pq}\cap T_3^{p-1q-1}}(\dots)\,dx\,dy+\iint\limits_{T_6^{pq}\cap T_2^{p-1q-1}}(\dots)\,dx\,dy=0.$$
$$A_{pq,pq-1}=\iint\limits_{T_6^{pq}\cap T_4^{pq-1}}(\dots)\,dx\,dy+\iint\limits_{T_1^{pq}\cap T_3^{pq-1}}(\dots)\,dx\,dy=-1.$$
$$A_{pq,p+1q}=\iint\limits_{T_2^{pq}\cap T_4^{p+1q}}(\dots)\,dx\,dy+\iint\limits_{T_1^{pq}\cap T_5^{p+1q}}(\dots)\,dx\,dy=-1.$$
$$A_{pq,p+1q+1}=\iint\limits_{T_3^{pq}\cap T_5^{p+1q+1}}(\dots)\,dx\,dy+\iint\limits_{T_2^{pq}\cap T_6^{p+1q+1}}(\dots)\,dx\,dy=0.$$
$$A_{pq,pq}=\iint\limits_D(\dots)\,dx\,dy=\iint\limits_{T_1^{pq}\cup T_2^{pq}\cup T_3^{pq}\cup T_4^{pq}\cup T_5^{pq}\cup T_6^{pq}}(\dots)\,dx\,dy=4.$$

Таким образом, система (2.18) записывается в виде

$$(u_{pq,p-1q}-2u_{pq,pq}+u_{pq,p+1q})+(u_{pq,pq-1}-2u_{pq,pq}+u_{pq,pq+1})=\iint\limits_D f\varphi_{pq}\,dx\,dy.$$

Вновь, как и в п. 1.3.2, имеем обычную конечно-разностную схему для задачи (2.1), (2.2), если для интеграла в правой части использовать приближенное значение (аналог формулы центральных прямоугольников)

$$\iint\limits_D f\varphi_{pq}\,dx\,dy\approx f(x_p,y_q)\varphi_{pq}(x_p,y_q)h^2=f(x_p,y_q)h^2.$$

Часть II
Обучение на примерах

В этой части рассмотрены примеры построения численного решения различных задач методом конечных элементов с использованием языка FreeFem++. Авторы выбирали задачи, руководствуясь следующими соображениями. Во-первых, это ряд классических задач курса математической физики — стационарные и нестационарные краевые задачи для уравнения Лапласа и задачи переноса. Во-вторых, это задачи, принадлежащие области научных интересов авторов, — задачи о различных течениях несжимаемой жидкости, задачи о переносе примесей жидкостью и электрическим полем, задачи о конвективных движениях жидкости, а также математические модели в биологии, в частности, задачи для уравнений реакции-диффузии. В-третьих, это сравнительно простые задачи, для решения которых не требуются специальные численные методы и, по крайней мере, в некотором интервале изменения параметров задачи (коэффициентов диффузии, вязкости, теплопроводности и т. д.) применение метода конечных элементов достаточно эффективно. Наконец, в-четвертых, авторы, по возможности, старались избежать пересечений с множеством задач, приведенных в [1]. По последней причине, в частности, отсутствуют примеры задач теории упругости — области, в которой применение метода конечных элементов очень популярно.

Изложение материала ориентировано на тот случай, когда математик, физик, биолог, химик, инженер и т. д., желающий для исследования некоторой математической модели, представленной уравнениями в частных производных с краевыми условиями (и начальными для нестационарной задачи), использовать метод конечных элементов, устанавливает на компьютере FreeFem++ (см. приложение А). Наличие ряда примеров, в том числе, содержащихся в [1], позволяет ему, не вдаваясь в подробности метода конечных элементов, записать задачу на языке FreeFem++ и убедиться в том, что данный численный метод работает и дает исследователю некоторую информацию о задаче. Конечно, предполагается, что пользователь FreeFem++ обладает некоторой математической подготовкой и владеет основами программирования.

Подчеркнем, что не требуется никаких специальных знаний о методе — достаточно некоторых *формальных* действий для преобразования (не очень сложной) математической задачи в операторы языка FreeFem++. В каком-то смысле, приводимые ниже примеры являются разговорником

иностранного языка — можно быстро составить фразу (математическая задача → FreeFem++) с надеждой быть понятым (получить результаты вычислений), хотя и необязательно адекватно. Используя эту аналогию, скажем, что краткий учебник *иностранного языка* FreeFem++ приведен в части 15.4, а более полный, конечно же, в [1].

Приведем формальную схему, которая используется ниже при рассмотрении конкретных примеров решения задач. Эта схема, начиная с п. 3, фактически, является структурой программы на языке FreeFem++ (последовательность действий в некоторых случаях может быть и иной).

1. Математическая постановка задачи — уравнения в частных производных, краевые условия (и начальные условия для нестационарных задач), $2D$ область, в которой решается задача.
2. Слабая (вариационная) формулировка задачи — в большинстве случаев, умножение исходных уравнений на (тестовую) функцию с последующим интегрированием по частям при помощи формулы Грина. Для нестационарных задач дополнительно используется алгоритм аппроксимации временных производных.
3. Задание на языке FreeFem++ области D, в которой решается задача — запись параметрических уравнений границы области.
4. Триангуляция области D — замена области D конечным набором треугольников, как можно более плотно покрывающих исходную область.
5. Выбор конечных элементов — выбор способа аппроксимации решения (кусочно-постоянными, кусочно-линейными, кусочно-квадратичными и т. п. функциями).
6. Запись на языке FreeFem++ слабой формулировки задачи — интегралы по области, интегралы по границе, краевые условия.
7. Выбор способа решения — решение системы линейных алгебраических уравнений методом Краута, методом LU разложения и т. п.
8. Для нестационарных задач дополнительно необходима организация цикла движения по времени.
9. Визуализация решения и вывод дополнительной информации о задаче и процессе решения.

Практически каждый пример решения задачи сопровождается полным работоспособным кодом на языке FreeFem++, который может быть просто скопирован и выполнен (см. также большое количество примеров, имеющееся в каталогах, в которых установлен FreeFem++, и их описание в [1]). Файлы с кодами FreeFem++ должны иметь расширение `edp`, так как компилятор языка `FreeFem++.exe` работает именно с такими файлами.

Каждая задача сопровождается результатами вычислительного эксперимента. При этом преследовались, по крайней мере, две цели. Одна из них заключается в том, что пользователь имеет возможность, скопировав код, сравнить результаты собственных вычислений с некоторым эталоном и убедиться в их правильности. Вторая цель — демонстрация весьма интересных и важных для понимания описываемых физических процессов результатов решения некоторой прикладной задачи.

Дополнительные математические сведения, в частности, об аппроксимации по времени, о решении уравнений переноса и т. п., приведены, по необходимости, отдельно или при описании решения задач. Кроме этого, в некоторых случаях, помимо формальной постановки задачи, приводятся достаточно подробные сведения либо о постановке аналогичных задач, либо о сущности описываемых физических (или иных) процессов.

Скажем несколько слов о точности приводимых вычислительных экспериментов. Основная цель примеров — это демонстрация *простейших возможностей* языка FreeFem++, а не исследование решения конкретных задач. По этой причине контроль точности выполняемых расчетов *не проводится* (точнее, он не содержится в текстах программ за исключением п. 4.6), хотя язык FreeFem++ и имеет много возможностей для осуществления такого контроля. Простейший способ проверки адекватности вычислительных экспериментов тем процессам, которые описывают математические модели, имеется у каждого читателя — можно повторить расчеты при измененном количестве треугольников, используемых для триангуляции области.

Во избежание недоразумений, особенно подчеркнем, что алгоритмы расчетов дееспособны только в случаях, когда порядки величин параметров задач мало отличаются от приведенных в тексте. К таким параметрам относятся коэффициенты теплопроводности, диффузии, вязкости, размеры области, значения известных функций в начальный момент времени и на границах, и пр. Рекомендованный способ проверки путем изменения (уменьшения) размеров треугольников может оказаться неэффективным, т. к. при этом возрастет размерность систем линейных алгебраических уравнений и, как следствие, ухудшается точность их решения.

Тут следует заметить, что метод конечных элементов, на самом деле, требует очень осторожного подхода к решению задач. Во многих случаях для известных задач (процессы теплопроводности, некоторые задачи теории упругости и теории течения жидкости) применимость этого метода математически строго обоснована и, в частности, даны оценки аппроксимации решений конечноэлементными функциями. Однако, для новых задач в каждом отдельном случае (особенно это относится к нелинейным задачам) необходимо обоснование применимости метода, оценки погрешности, уточнение требований на гладкость решений и т. п. (см. [3]).

Глава 3

Решение задач для уравнения Лапласа

Цель настоящей главы — демонстрация использования FreeFem++ для решения 2D стационарной краевой задачи, не приводя подробные объяснения. Наиболее просто это сделать для неоднородного уравнения Лапласа с различными краевыми условиями и в областях различной формы. Выбор для рассмотрения именно уравнения Лапласа не случаен. Во-первых, стационарная краевая задача для уравнения Лапласа является одной из наиболее часто встречающихся в курсе уравнений математической физики. Во-вторых, при помощи этой задачи описывается множество различных стационарных физических процессов, таких как стационарное распределение температуры, распределение концентрации, распределение электрического потенциала, распределение скорости для потенциального течения несжимаемой жидкости и т. п. В-третьих, на примере этой задачи легко показать простейшие возможности FreeFem++ — запись задачи в слабой формулировке, способы задания различных краевых условий, способы задания двумерной области, выбор конечных элементов, способы визуализации решения.

3.1 Задача о стационарном распределении температуры

3.1.1 Постановка задачи

Рассмотрим задачу о стационарном распределении температуры $u(x, y)$ в некоторой двумерной области D, на фрагментах границы которой задана температура g_1, поток тепла g_2 и условие теплопередачи Ньютона (краевое условие третьего рода).

Пусть дано неоднородное уравнение Лапласа

$$-\Delta u = f(x, y), \quad (x, y) \in D, \qquad \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}. \tag{3.1}$$

Поставим следующие краевые условия:

$$u\big|_{\Gamma_1} = g_1, \tag{3.2}$$

$$\left.\frac{\partial u}{\partial n}\right|_{\Gamma_2} = g_2, \tag{3.3}$$

$$\left.\left(\frac{\partial u}{\partial n} + \beta u\right)\right|_{\Gamma_3} = g_3. \tag{3.4}$$

Здесь $f(x, y)$ — функция, заданная в области D; $g_1(x, y)$, $g_2(x, y)$, $g_3(x, y)$ — функции, заданные на границе области D, β — заданный параметр.

Напомним, что (3.2) называется неоднородным условием Дирихле, (3.3) — неоднородным условием Неймана, а (3.4) называется неоднородным условием Робина (или Фурье, или Ньютона, или смешанным условием).

Форма области D может быть любой, однако предполагается, что граница области $\partial D = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ (возможно несвязная) является достаточно гладкой. Для определенности, считаем, что область $D = (0, a) \times (0, b)$ и фрагмент границы Γ_1 является несвязным: $\Gamma_1 = \Gamma_{11} \cup \Gamma_{12}$ (см. рис. 3.1).

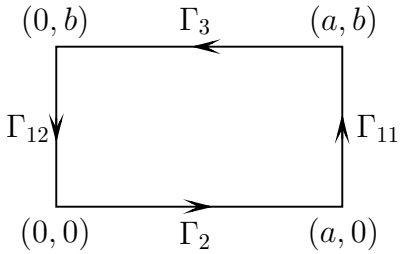

Рис. 3.1. Область $\overline{D}$. Прямоугольник $[0, a] \times [0, b]$

3.1.2 Слабая формулировка задачи

Основная причина, по которой следует переходить от сильной к слабой формулировке задачи — язык FreeFem++ (и метод конечных элементов) ориентирован на решение задач именно в слабой формулировке. Достаточно подробно способ такого перехода для задачи, аналогичной рассматриваемой, описан в гл. 2, где также имеются определения, вводится используемая терминология и т. п. Однако, для дальнейшего изучения материала знакомство с гл. 2 необязательно и приводимые ниже преобразования можно рассматривать как набор формальных действий.

Запишем задачу (3.1)–(3.4) в слабой (вариационной) формулировке. Для этого умножим (3.1) на тестовую (пробную) функцию $v(x, y)$ и проинтегрируем по области D

$$-\iint\limits_D v \Delta u \, dx \, dy = \iint\limits_D v f \, dx \, dy.$$

Используя формулу Грина, получим слабую формулировку исходной задачи (3.1)–(3.4)

$$\iint\limits_D \nabla v \cdot \nabla u \, dx \, dy - \int\limits_\Gamma v \frac{\partial u}{\partial n} \, ds - \iint\limits_D f v \, dx \, dy = 0, \quad \forall\, v(x, y). \tag{3.5}$$

Здесь $\partial u/\partial n$ — производная по внешней нормали к границе области D (по поводу обозначений см., в частности, (2.3)–(2.5)).

Ввиду того, что краевые условия (3.2)–(3.4) на фрагментах контура Γ заданы различными (см. также рис. 3.1), перепишем (3.5) в виде

$$\iint\limits_D \nabla v\cdot\nabla u\,dx\,dy-\int\limits_{\Gamma_1} v\frac{\partial u}{\partial n}\,ds-\int\limits_{\Gamma_2} v\frac{\partial u}{\partial n}\,ds-\int\limits_{\Gamma_3} v\frac{\partial u}{\partial n}\,ds-\iint\limits_D fv\,dx\,dy=0. \quad (3.6)$$

Преобразуем интегралы по контурам Γ_2, Γ_3, исключая производные с учетом краевых условий (3.3)–(3.4). Для контура Γ_3 с учетом краевого условия (3.4) выводим

$$-\int\limits_{\Gamma_3} v\frac{\partial u}{\partial n}\,ds=-\int\limits_{\Gamma_3} v\,(g_3-\beta u)\,ds. \quad (3.7)$$

Аналогично, для контура Γ_2 с учетом краевого условия (3.3) имеем

$$-\int\limits_{\Gamma_2} v\frac{\partial u}{\partial n}\,ds=-\int\limits_{\Gamma_2} vg_2\,ds. \quad (3.8)$$

По терминологии метода конечных элементов (см., в частности, п. 1.4) краевые условия (3.3), (3.4) являются *естественными краевыми условиями*. Грубо говоря, эти краевые условия могут быть выполнены за счет требования обращения в нуль интегралов по контуру (после исключения производных) для *всех произвольных* тестовых функций $v(x,y)$.

Интеграл по контуру Γ_1 не может быть преобразован подобным образом, т. к. условие (3.2) не содержит производных функции $u(x,y)$. Краевое условие (3.2) является *главным краевым условием*. Для того, чтобы краевое условие было выполнено, необходимо накладывать дополнительные требования на функции $v(x,y)$.

Дополнительное ограничение на функции $v(x,y)$ в данном случае аналогично *однородному* условию (3.2) и имеет вид

$$v\big|_{\Gamma_1}=0. \quad (3.9)$$

Окончательно, исходная задача (3.1)–(3.4) в слабой формулировке с учетом (3.6)–(3.8) принимает вид

$$\iint\limits_D \nabla u\cdot\nabla v\,dx\,dy \;-\; \iint\limits_D fv\,dx\,dy-$$

$$-\int\limits_{\Gamma_1} v\frac{\partial u}{\partial n}\,ds \;-\; \int\limits_{\Gamma_2} g_2v\,ds-\int\limits_{\Gamma_3}(g_3-\beta u)\,v\,ds=0,\quad \forall\, v$$

или с учетом (3.9)

$$\iint\limits_D\nabla u\cdot\nabla v\,dx\,dy-\int\limits_{\Gamma_2} g_2v\,ds-\int\limits_{\Gamma_3}(g_3-\beta u)\,v\,ds-\iint\limits_D fv\,dx\,dy=0, \quad (3.10)$$

$$\forall\, v\big|_{\Gamma_1}=0.$$

3.1.3 Слабая формулировка задачи на языке FreeFem++

Прежде чем приводить коды на языке FreeFem++, напомним покоординатную форму записи следующего интеграла

$$\iint\limits_D \nabla v \cdot \nabla u \, dx \, dy = \iint\limits_D \left(\frac{\partial v}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \frac{\partial u}{\partial y} \right) dx \, dy. \tag{3.11}$$

Удобно также использовать обозначения

$$\frac{\partial}{\partial x} = \partial_x, \quad \frac{\partial}{\partial y} = \partial_y. \tag{3.12}$$

С учетом (3.12) формула (3.11) запишется в виде

$$\iint\limits_D \nabla v \cdot \nabla u \, dx \, dy = \iint\limits_D \left(\partial_x v \, \partial_x u + \partial_y v \, \partial_y u \right) dx \, dy. \tag{3.13}$$

Приведем схему, при помощи которой задача в слабой формулировке (3.10) записывается на языке FreeFem++

$$\iint\limits_D \nabla v \cdot \nabla u \, dx \, dy \;\; \rightarrow \;\; \texttt{int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))}$$

$$-\int\limits_{\Gamma_2} g_2 v \, ds \;\; \rightarrow \;\; \texttt{-int1d(Th,Gamma2)(g2 * v)}$$

$$-\int\limits_{\Gamma_3} (g_3 - \beta u) v \, ds \;\; \rightarrow \;\; \texttt{+int1d(Th,Gamma3)(beta*u*v)-int1d(Th,Gamma3)(g3*v)}$$

$$-\iint\limits_D f v \, dx \, dy \;\; \rightarrow \;\; \texttt{-int2d(Th)(f * v)}$$

$$-\int\limits_{\Gamma_1} v \frac{\partial u}{\partial n} \, ds \;\; \rightarrow \;\; \texttt{on(Gamma1, u = g1)}$$

Таким образом, интегралы, содержащиеся в выражении (3.10), практически дословно переписываются в кодах языка FreeFem++. Может быть следующая схема соответствий является даже более удобной

$$\iint\limits_D (\dots) \, dx \, dy \;\; \rightarrow \;\; \texttt{int2d(Th)(...)}$$

$$\int\limits_{\Gamma_i} (\dots) \, ds \;\; \rightarrow \;\; \texttt{int1d(Th,GammaI)(...)}$$

$$\partial_x(\dots) \, \partial_x(\dots) \;\; \rightarrow \;\; \texttt{dx(...)*dx(...)}$$

$$\partial_y(\dots) \, \partial_y(\dots) \;\; \rightarrow \;\; \texttt{dy(...)*dy(...)}$$

Интуитивно понятно, что `Th` в выражении `int2d(Th)` обозначает область интегрирования (более точно, `Th` — триангуляция области D, см.

ниже), `(Th, GammaI)` в выражении `int1d(Th,GammaI)` указывает область и фрагмент ее границы, по которой проводится интегрирование.

По поводу записи интегралов по границе сделаем важное замечание.

✓. *Интегралы* `int1d` *в случае фрагмента границы* Γ_3 *нельзя записывать в виде одного интеграла. Дело в том, что один из них, содержащий произведение* v *на известную функцию, является линейной формой относительно тестовой функции* v*, а другой, содержащий произведение* uv*, является билинейной формой относительно функций* u*,* v*. Язык FreeFem++ не позволяет смешивать эти понятия (выражения).*

Также должно быть ясно, что краевое условие (3.2) требует специальной формы записи. В подынтегральном выражении для интеграла по границе Γ_1 производная $\partial u/\partial n$ не может быть исключена при помощи каких-либо преобразований. Именно поэтому краевое условие (3.2) задается непосредственно в том виде, как оно записано, при помощи ключевого слова `on` в форме: `on(Gamma1, u = g1)`.

Фрагмент программы для решения задачи на языке FreeFem++ выглядит следующим образом (см. также полный текст программы, приведенный ниже на с. 43)

```
24 solve Poisson(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )
25                      - int1d(Th,Gamma2)( g2 * v )
26                      + int1d(Th,Gamma3)( beta/alpha * u*v )
27                      - int1d(Th,Gamma3)( g3/alpha * v )
28                      - int2d(Th)( f * v )
29                      + on( Gamma11, u = g1 )
30                      + on( Gamma12, u = g1 ) ;
```

Обратим внимание, что нумерация строк использована для удобства ссылок и при использовании кода должна быть удалена, т. к. FreeFem++ ***не требует*** нумерации строк.

3.1.4 Задание области D на языке FreeFem++

Для того, чтобы определить область D в FreeFem++ достаточно указать границу области D. Это возможно сделать при помощи параметрического задания отдельных частей границы с использованием ключевого слова `border`. Приведем пример задания области D в случае прямоугольника $[0, a] \times [0, b]$ (см. рис. 3.1)

```
6 border Gamma2  (t=0,1) { x=a*t;     y=0; };
7 border Gamma11 (t=0,1) { x=a;       y=b*t; };
8 border Gamma3  (t=0,1) { x=a*(1-t); y=b; };
9 border Gamma12 (t=0,1) { x=0;       y=b*(1-t); };
```

Граница $\Gamma_1 = \Gamma_{11} \cup \Gamma_{12}$ предполагается несвязной. Именно поэтому для каждого ее связного фрагмента использованы различные идентификаторы `Gamma11` и `Gamma12`.

Интуитивно понятно, что, например, строка 6 соответствует параметрической форме записи для отрезка прямой линии $x \in [0, a]$, $y = 0$

$$x = at, \quad y = 0, \quad 0 \leqslant t \leqslant 1.$$

Заметим, что порядок определения фрагментов границы не имеет значения, т. е. строчки с операторами `border` в приведенном выше фрагменте программы могут быть переставлены. Однако важно, чтобы при возрастании параметра t обход области D совершался в направлении ***против часовой стрелки***.

3.1.5 Полный код программы на языке FreeFem++

Зададим для определенности следующие функции и параметры для исходной задачи (3.1)–(3.4)

$$\overline{D} = \big\{(x, y)\colon [0, a] \times [0, b]\big\}, \quad a = 1, \quad b = 1,$$

$$f(x, y) = \sin 2\pi x \, \sin 2\pi y, \quad g_1 = 0, \quad g_2 = 1, \quad g_3 = 1, \quad \beta = 1. \tag{3.14}$$

Приведем полный код программы для построения численного решения задачи

```
1  real a=1.0;  // ширина области D
2  real b=1.0;  // высота области D
3  int n=4;     // вспомогательный параметр для триангуляции области D
4  // задание границ области D (прямоугольник [0,a]x[0,b])
5  // сохраняем ориентацию контура -- против часовой стрелки
6  border Gamma2(t=0,1){ x=a*t; y=0; };          // bottom
7  border Gamma11(t=0,1){ x=a; y=b*t; };         // right
8  border Gamma3(t=0,1){ x=a*(1-t); y=b; };      // top
9  border Gamma12(t=0,1){ x=0; y=b*(1-t); };     // left
10 // построение сетки Th, каждый фрагмент границы разбит на 5*n отрезков
11 mesh Th = buildmesh(Gamma2(5*n)+Gamma11(5*n)+Gamma3(5*n)+Gamma12(5*n));
12 // plot(Th,wait=1);  // для визуализации построенной сетки
13 // задание пространства конечных элементов
14 fespace Vh(Th,P2);
15 // задание на пространстве Vh искомой функции u и пробной функции v
16 Vh u,v;
17 // определение функций и параметров исходной задачи
18 func f = sin(2*pi*x)*sin(2*pi*y);
19 func g1 = 0;
20 func g2 = 1;
21 func g3 = 1;
22 real beta=1;
23 // запись слабой (вариационной) формулировки задачи и ее решение
24 solve Poisson(u,v) = int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v) )
25                      - int1d(Th,Gamma2)( g2 * v )
26                      + int1d(Th,Gamma3)( beta * u*v )
27                      - int1d(Th,Gamma3)( g3 * v )
28                      - int2d(Th)( f * v )
```

```
29                          + on( Gamma11, u = g1 )
30                          + on( Gamma12, u = g1 ) ;
31  plot(u);  // графическое представление решения
```

Результат решения задачи (3.1)–(3.4) представлен на рис. 3.2, на котором изображены линии уровня (изолинии или изотермы, если u температура) функции $u(x, y)$. Заметим, что на экране дисплея изолинии будут цветными. Записав вместо 31-ой строки `plot(u, value=1)`, можно получить таблицу соответствия цвета изолинии и значения функции $u(x, y)$ на этой изолинии. Для сохранения результатов расчетов в файл следует использовать строку `plot(u, value=1, ps="FileName")`. При этом рисунок будет сохранен в postcript-формате (.ps).

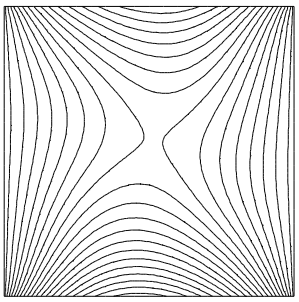

Рис. 3.2. Изолинии функции $u(x, y)$ (температуры) — решение задачи (3.1)–(3.4)

Сделаем некоторые пояснения:

1) `Th` — идентификатор генерируемой сетки. Можно задавать и другое имя, например, `Th2`, однако, в этом случае в `int2d(Th)`, `int1d(Th,GammaI)` и т. п. следует заменить `Th` на `Th2`.

2) `Vh(Th,P2)` — идентификатор пространства конечных элементов. Имя `Vh` может быть любым, `Th` — должен совпадать с именем генерируемой сетки, `P2` — наименование типа конечных элементов (зарезервированное слово в FreeFem++, подробнее см. гл. 18).

3) `solve Poisson(u,v)` — имя `Poisson` задается пользователем и может быть любым. Ключевое слово `solve` означает, что задача формулируется и одновременно решается (см. также о ключевом слове `problem` в п. 19.1).

На этом этапе изучения языка FreeFem++ можно не задумываться над строками 10–14 программы. Интуитивно понятно, что при помощи этих строк задается триангуляция области D и выбирается способ аппроксимации решения некоторыми конечноэлементными функциями. Заметим только, что использованный тип конечных элементов (`P2`) соответствует аппроксимации кусочно-непрерывными квадратичными функциями (подробнее см. гл. 18).

3.2 Решение задачи о распределении температуры в областях сложной формы

Одной из важных особенностей метода конечных элементов и, в частности, языка FreeFem++ является возможность решать задачу в области

сравнительно произвольной формы с достаточно гладкой границей. В языке FreeFem++ реализован эффективный алгоритм триангуляции области, требующий для генерации треугольной сетки задания лишь количества вершин треугольников на границах. Замечательно то, что основная часть кода программы для решения задачи в прямоугольнике остается прежней. Переход к другой области осуществляется заменой только строк 6–9, определяющих границу.

3.2.1 Решение задачи для круга

Пусть область $\overline{D}$ представляет собой круг $x^2 + y^2 \leqslant 1$ единичного радиуса. Границу области $\overline{D}$ можно задать параметрически при помощи соотношений $x = \cos t$, $y = \sin t$, $t \in [0, 2\pi]$. Предположим, что граница области разбита на четыре фрагмента

$$\begin{array}{lllll}
\Gamma_{11}: & x = \cos t, & y = \sin t, & t \in [0,\ 0{,}33\pi]; \\
\Gamma_{2}\ : & x = \cos t, & y = \sin t, & t \in [0{,}33\pi,\ \pi]; \\
\Gamma_{3}\ : & x = \cos t, & y = \sin t, & t \in [\pi,\ 1{,}41\pi]; \\
\Gamma_{12}: & x = \cos t, & y = \sin t, & t \in [1{,}41\pi,\ 2\pi].
\end{array}$$

Операторы, задающие границу, в этом случае будут следующими:

```
border Gamma11(t=0,0.33*pi)     { x=cos(t); y=sin(t); };
border Gamma2 (t=0.33*pi,pi)    { x=cos(t); y=sin(t); };
border Gamma3 (t=pi,1.41*pi)    { x=cos(t); y=sin(t); };
border Gamma12(t=1.41*pi,2.0*pi){ x=cos(t); y=sin(t); };
```

Решение задачи (изолинии температуры) (3.1)–(3.4) для параметров, определенных соотношениями (3.14), приведено на рис. 3.3.

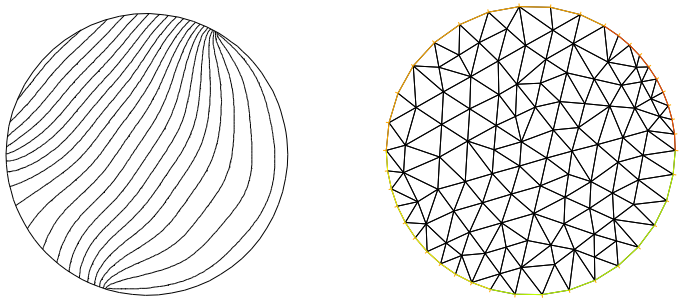

Рис. 3.3. Изолинии температуры для задачи в круге и триангуляция

3.2.2 Распределение температуры в четырехугольнике

Пусть D представляет собой четырехугольник с координатами вершин (x_i, y_i) $(i = 0, \ldots, 3)$. В этом случае код, задающий границу, может быть записан в виде

```
int m=4;
real[int] xx(m), yy(m);
xx[0]=0;    yy[0]=0.2;    xx[1]=0.5;  yy[1]=0;
xx[2]=0.8;  yy[2]=0.8;    xx[3]=0.5;  yy[3]=1;
border Gamma2(t=0,1) { x=xx[0]*(1-t)+xx[1]*t; y=yy[0]*(1-t)+yy[1]*t; };
border Gamma11(t=0,1){ x=xx[1]*(1-t)+xx[2]*t; y=yy[1]*(1-t)+yy[2]*t; };
border Gamma3(t=0,1) { x=xx[2]*(1-t)+xx[3]*t; y=yy[2]*(1-t)+yy[3]*t; };
border Gamma12(t=0,1){ x=xx[3]*(1-t)+xx[0]*t; y=yy[3]*(1-t)+yy[0]*t; };
```

Решение задачи (изолинии температуры) (3.1)–(3.4) для параметров, определенных соотношениями (3.14), приведено на рис. 3.4.

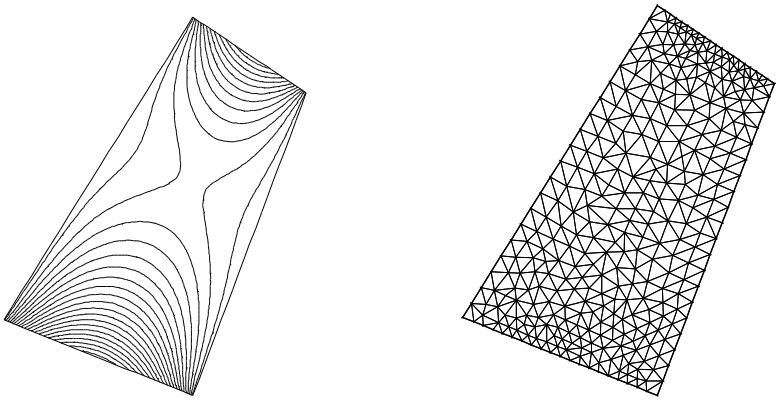

Рис. 3.4. Изолинии температуры для задачи в четырехугольнике и триангуляция

Для области, представляющей собой полигон, координаты вершин которого (x_i, y_i) $(i = 0, \ldots, m)$ известны, граница может быть задана аналогичным образом. Однако, при этом потребуются и другие изменения в программе. Необходимо будет переписать строку с оператором генерации сетки `mesh Th = buildmesh(...)` и строки, задающие краевые условия на соответствующих участках границы полигона.

3.2.3 Решение задачи в криволинейной области

Две точки на плоскости (x_1, y_1), (x_2, y_2) можно соединить кривой линией, используя любое параметрическое представление кривой, например,

$$x = x_1 \cos t + x_2 \sin t, \qquad y = y_1 \cos t + y_2 \sin t, \quad t \in [0, \pi/2]$$

или

$$x = x_1(1 - t^2) + x_2 t^2, \qquad y = y_1(1 - t^2) + y_2 t^2, \quad t \in [0, 1].$$

Приведем код для задания криволинейной области (см. рис. 3.5)

```
int m=4;
real[int] xx(m), yy(m);
xx[0]=0;     yy[0]=0.4;   xx[1]=0.5;    yy[1]=0;
xx[2]=1;     yy[2]=1;     xx[3]=0.15;   yy[3]=0.35;
```

```
border Gamma2(t=0,1) { x=xx[0]*(1-t)+xx[1]*t;
                                    y=yy[0]*(1-t)+yy[1]*t; };
border Gamma11(t=0,1){ x=xx[1]*(1-t)+xx[2]*t;
                                    y=yy[1]*(1-t)+yy[2]*t; };
border Gamma3(t=0,1) { x=xx[2]*(1-t^2)+xx[3]*t^2;
                                    y=yy[2]*(1-t^2)+yy[3]*t^2; };
border Gamma12(t=0,0.5*pi){ x=xx[3]*cos(t)+xx[0]*sin(t);
                                    y=yy[3]*cos(t)+yy[0]*sin(t); };
```

Решение задачи (изолинии температуры) (3.1)–(3.4) для параметров, определенных соотношениями (3.14), приведено на рис. 3.5.

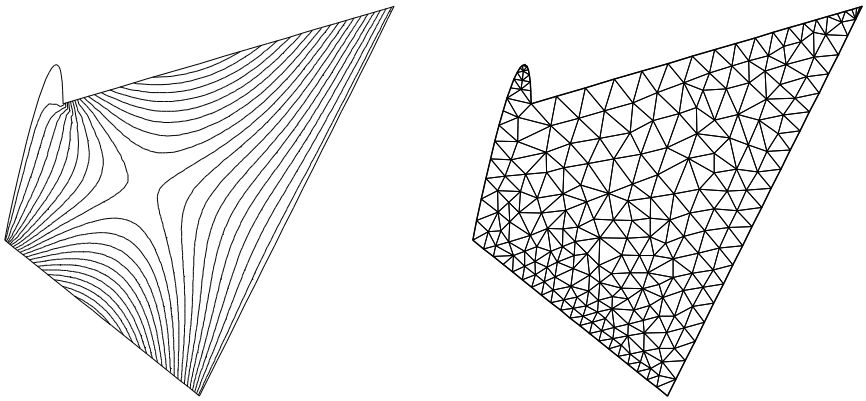

Рис. 3.5. Изолинии температуры для задачи в криволинейной области и триангуляция

3.2.4 Решение задачи в области с отверстием

Приведем фрагмент кода, позволяющий рассмотреть задачу, например, в области D с круглым отверстием (см. рис. 3.6).

```
border Gamma11(t=0,0.33*pi) { x=cos(t); y=sin(t); };
border Gamma2 (t=0.33*pi,pi){ x=cos(t); y=sin(t); };
border Gamma3 (t=pi,2*pi)   { x=cos(t); y=sin(t); };
border Gamma12(t=0,2*pi)    { x=0.5*cos(t); y=0.5*sin(t); };
```

Решение задачи (изолинии температуры) (3.1)–(3.4) для параметров, определенных соотношениями (3.14), приведено на рис. 3.6.

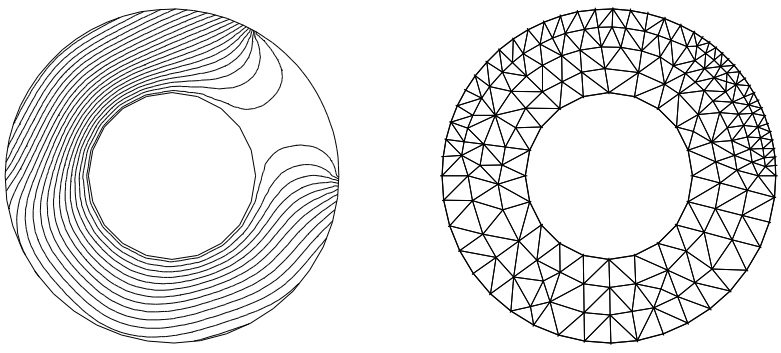

Рис. 3.6. Изолинии температуры для задачи в кольце и триангуляция

Обратим внимание, что для сохранения ориентации контура в операторе `mesh` следует записывать внутренний контур в следующем формате `Gamma12(-5*n)` (подробнее см. гл. 17). Соответствующая строка кода, генерирующая сетку, имеет вид

```
mesh Th = buildmesh(Gamma2(5*n)+Gamma11(5*n)+Gamma3(5*n)+Gamma12(-5*n));
```

Еще один пример сложной области с отверстиями, для которой можно решать задачи (в частности, уравнение теплопроводности с заданными краевыми условиями), показан на рис. 3.7.

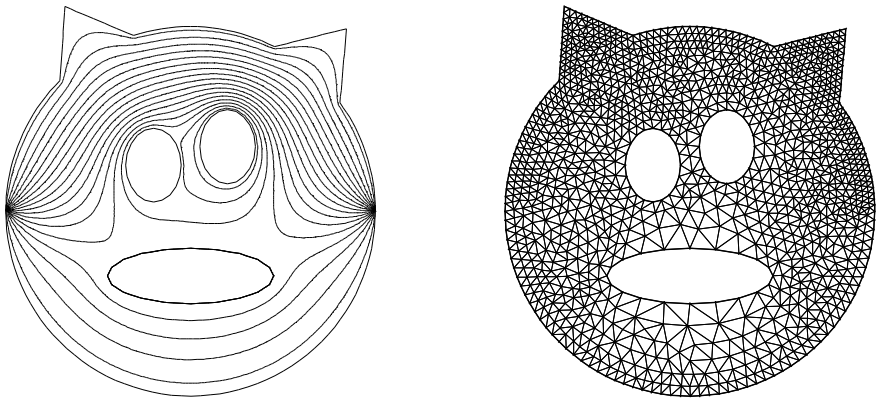

Рис. 3.7. Изолинии температуры в области сложной формы и триангуляция

3.3 Физические задачи, приводящие к уравнению Лапласа

Этот раздел носит справочный характер и предназначен в первую очередь для тех, кто, написав первую программу на языке FreeFem++, хочет расширить свой физический кругозор и исследовать некоторые модели реальных прикладных задач. Более полные сведения о постановке задач, конечно же, можно найти практически в любом курсе уравнений математической физики. Однако, на наш взгляд, полезно иметь возможность почерпнуть некоторую информацию непосредственно в книге о языке FreeFem++. Это, в частности, позволит сразу же провести вычислительные эксперименты и получить наглядное представление о многих физических процессах.

3.3.1 Теплопроводность

Стационарное распределение температуры в некоторой области D описывается уравнением (уравнение баланса энергии)

$$\operatorname{div} \boldsymbol{q} = f, \tag{3.15}$$

где $\boldsymbol{q}$ — плотность потока тепла, f — плотность внутренних источников.

Плотность потока тепла $\boldsymbol{q}$ связана с температурой θ законом теплопроводности Фурье

$$\boldsymbol{q} = -\varkappa \nabla \theta, \tag{3.16}$$

где $\varkappa$ — коэффициент теплопроводности ($\varkappa \geqslant 0$).

Умножая (3.15) на v и интегрируя по области D, получим

$$-\iint\limits_D \boldsymbol{q} \cdot \nabla v \, dx \, dy + \int\limits_{\partial D} v\boldsymbol{q} \cdot \boldsymbol{n} \, ds - \iint\limits_D vf \, dx \, dy = 0 \tag{3.17}$$

или с учетом (3.16)

$$\iint\limits_D \varkappa \nabla\theta \cdot \nabla v \, dx \, dy + \int\limits_{\partial D} v\boldsymbol{q} \cdot \boldsymbol{n} \, ds - \iint\limits_D vf \, dx \, dy = 0. \tag{3.18}$$

Далее считаем, что граница области D состоит из трех, возможно несвязных, фрагментов (некоторые Γ_i могут быть пустыми множествами)

$$\partial D = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3.$$

Краевые условия первого рода, соответствующие заданию температуры на границе области, имеют вид

$$\theta\big|_{\Gamma_1} = \theta_0(x, y)\big|_{\Gamma_1}, \tag{3.19}$$

где $\theta_0(x, y)$ — заданное распределение температуры на границе области.

С физической точки зрения такое условие соответствует случаю, когда область D «окружена» некоторой «внешней» областью («внешняя» область может быть отверстием в области D, см., например, рис. 3.6, 3.7), в которой известно распределение температуры $\theta_0(x, y)$. Кроме этого предполагается, что распределение температуры θ в области D не влияет на распределение температуры $\theta_0(x, y)$. Такое возможно, по крайней мере, по двум причинам — размеры внешней области много больше размеров области D, теплоемкость (способность тела сохранять тепло) внешней области много больше теплоемкости области D.

Краевые условия второго рода, соответствующие заданию плотности потока тепла через границу области, имеют вид

$$(\boldsymbol{q} \cdot \boldsymbol{n})\big|_{\Gamma_2} = q_n^0(x, y)\big|_{\Gamma_2}, \tag{3.20}$$

где $q_n^0(x, y)$ — заданная плотность потока тепла через границу (точнее, нормальная к границе компонента потока тепла).

Краевые условия третьего рода имеют вид

$$(\boldsymbol{q} \cdot \boldsymbol{n})\big|_{\Gamma_3} = \varkappa_{\text{out}}(x, y)(\theta - \theta_{\text{out}}(x, y))\big|_{\Gamma_3}, \tag{3.21}$$

где $\varkappa_{\text{out}}$ — коэффициент внешней теплопроводности ($\varkappa_{\text{out}} \geqslant 0$), $\theta_{\text{out}}(x, y)$ — температура на внешней границе области.

Случаи, когда $q_n^0 = 0$ или $\varkappa_{\text{out}} = 0$, соответствуют ***теплоизолированным*** участкам границы.

С физической точки зрения, краевые условия третьего рода (3.21) отвечают равенству нормальных компонент плотности внутреннего ($\boldsymbol{q}$) и внешнего ($\boldsymbol{q}^{\text{out}}$) потоков тепла на границе

$$(\boldsymbol{q} \cdot \boldsymbol{n})\big|_{\Gamma_3} = (\boldsymbol{q}^{\text{out}} \cdot \boldsymbol{n})\big|_{\Gamma_3}. \tag{3.22}$$

При теплообмене по закону Ньютона считается, что нормальная компонента плотности внешнего потока тепла определяется разностью температур на внутренней (θ) и внешней (θ_{out}) границах области при помощи соотношения

$$\boldsymbol{q}^{\text{out}} = -\varkappa^{\text{out}}(\theta_{\text{out}} - \theta)\boldsymbol{n}. \tag{3.23}$$

Легко проверить, что это соответствует потоку тепла от горячего тела к холодному ($\boldsymbol{n}$ — внешняя нормаль).

Краевое условие (3.21), соответствующее теплообмену тела с внешней средой по закону Ньютона получится при помощи (3.23) в (3.22).

Коэффициенты теплопроводности могут быть функциями, зависящими от координат. Случай $\varkappa = \varkappa(x, y)$ соответствует неоднородной области, например, состоящей из различных материалов (медь, сталь). Случай $\varkappa_{\text{out}} = \varkappa_{\text{out}}(x, y)$ моделирует неоднородности теплопроводящей границы.

Сильной формулировке задачи соответствуют уравнения (3.15), (3.16) с граничными условиями (3.19)–(3.21). Задача в слабой формулировке получится, если в (3.18) исключить при помощи условий (3.20) и (3.21) величину $(\boldsymbol{q} \cdot \boldsymbol{n})$, т. е. заменить $(\boldsymbol{q} \cdot \boldsymbol{n})$ на правые части условий (3.20) и (3.21). Для выполнения краевых условий (3.19) следует требовать выполнения на границе Γ_1 дополнительного ограничения на тестовые функции v — выполнения *однородных* условий $v|_{\Gamma_1} = 0$.

Одним из достоинств метода конечных элементов и языка FreeFem++ является возможность несложного модифицирования кода программы для решения задачи с коэффициентами, зависящими от координат.

Пусть, например, $\varkappa = \varkappa(x, y)$. Оператор, задающий функцию $\varkappa$, имеет вид (для определенности, $\varkappa(x, y) = xy$)

```
Vh kappa = x * y;
```

Эту строку следует вставить, например, между строками 22, 23 (см. с. 43).

Для решения с помощью языка FreeFem++ достаточно вместо строки 24 (см. с. 43)

```
solve Poisson(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )
```

записать строку

```
solve Poisson(u,v) =
          int2d(Th)( kappa*(dx(u)*dx(v) + dy(u)*dy(v)) )
```

Особое внимание следует уделять решению второй краевой задачи Неймана, когда ***на всей границе*** поставлено краевое условие второго рода

$$(\boldsymbol{q} \cdot \boldsymbol{n})\big|_{\partial D} = q_n^0(x, y)\big|_{\partial D}. \tag{3.24}$$

В этом случае должно выполняться так называемое условие разрешимости, которое получается интегрированием по области D уравнения (3.15) с последующей заменой $(\boldsymbol{q} \cdot \boldsymbol{n})$ при помощи (3.24)

$$\iint\limits_D \operatorname{div} \boldsymbol{q}\, dx\, dy = \iint\limits_D f\, dx\, dy \quad \Rightarrow \quad \int\limits_{\partial D} (\boldsymbol{q} \cdot \boldsymbol{n})\, ds = \iint\limits_D f\, dx\, dy$$

и окончательно

$$\int\limits_{\partial D} q_n^0\, ds = \iint\limits_D f\, dx\, dy. \tag{3.25}$$

Полученное соотношение означает, что при постановке задачи Неймана нельзя произвольно задавать плотность распределения источников тепла $f(x, y)$ и плотность потока тепла на границе $q_n^0|_{\partial D}$ — они должны удовлетворять соотношению (3.25).

Кроме этого, непосредственной подстановкой в (3.15), (3.16), (3.24) можно проверить, что если θ является решением задачи, то и $\theta + \mathrm{const}$ также будет решением задачи. Это означает, что задача Неймана имеет не единственное решение (определяется с точностью до константы).

При численном решении задачи Неймана возникают дополнительные осложнения, связанные с тем, что, по причине наличия вычислительной погрешности, точно удовлетворить условиям разрешимости (3.25) невозможно. В дальнейшем (см. гл. 8 и формулы (8.25)–(8.29)) будут показаны некоторые варианты численного решения задачи Неймана.

В заключение, укажем способ, который позволяет визуализировать векторное поле, соответствующее потоку тепла. Для этого в код программы на с. 43 следует добавить строки

```
Vh qx=-dx(u), qy=-dy(u);
plot(u, [qx,qy], coef=0.05, value=1);
```

Параметр `coef` регулирует длину стрелок векторов векторного поля при визуализации.

3.3.2 Диффузия

Предположим, что некоторая область D заполнена раствором, состоящим из растворителя и примеси. Стационарное распределение концентрации примеси описывается уравнением (уравнение баланса массы)

$$\operatorname{div} \boldsymbol{i} = r, \tag{3.26}$$

где $\boldsymbol{i}$ — плотность потока концентрации примеси, r — плотность внутренних источников концентрации, возникающих, например, в результате химических реакций.

Плотность потока концентрации $\boldsymbol{i}$ связана с концентрацией $c(x, y)$ законом Фика

$$\boldsymbol{i} = -D_c \nabla c, \tag{3.27}$$

где D_c — коэффициент диффузии ($D_c \geqslant 0$).

Далее считаем, что граница области D состоит из трех, возможно несвязных, фрагментов (некоторые Γ_i могут быть пустыми множествами)

$$\partial D = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3.$$

Краевые условия первого рода, соответствующие заданию концентрации на границе области, имеют вид

$$c\big|_{\Gamma_1} = c_0(x, y)\big|_{\Gamma_1}, \tag{3.28}$$

где $c_0(x, y)$ — заданное распределение концентрации на границе области.

С физической точки зрения такое условие соответствует случаю, когда область D «окружена» некоторой «внешней» областью, в которой известна концентрация примеси $c_0(x, y)$, и предполагается, что распределение концентрации c в области D не влияет на распределение $c_0(x, y)$.

Краевые условия второго рода, соответствующие заданию плотности потока примеси через границу области, имеют вид

$$(\boldsymbol{i} \cdot \boldsymbol{n})\big|_{\Gamma_2} = i_n^0(x, y)\big|_{\Gamma_2}, \tag{3.29}$$

где $i_n^0(x, y)$ — заданная плотность потока концентрации через границу (точнее, её нормальная к границе компонента).

Краевые условия третьего рода имеют вид (ср. с (3.21)-(3.23))

$$(\boldsymbol{i} \cdot \boldsymbol{n})\big|_{\Gamma_3} = D_{\text{out}}(x, y)(c - c_{\text{out}}(x, y))\big|_{\Gamma_3}, \tag{3.30}$$

где D_{out} — коэффициент внешней диффузии ($D_{\text{out}} \geqslant 0$), $c_{\text{out}}(x, y)$ — концентрация на внешней границе области.

Случаи, когда $i_n^0 = 0$ или $D_{\text{out}} = 0$, соответствуют участкам границы, ***непроницаемым*** для концентрации примеси.

Совершенно очевидно, что с математической точки зрения процессы теплопроводности и диффузии описываются одними и теми же уравнениями. Соотношения п. 3.3.1 и п. 3.3.2 становятся одинаковыми после формальных замен

$$\theta \leftrightarrows c, \quad \boldsymbol{q} \leftrightarrows \boldsymbol{i}, \quad \varkappa \leftrightarrows D_c, \quad \varkappa_{\text{out}} \leftrightarrows D_{\text{out}},$$
$$\theta_{\text{out}} \leftrightarrows c_{\text{out}}, \quad q_n^0 \leftrightarrows i_n^0, \quad \theta_0 \leftrightarrows c_0, \quad f \leftrightarrows r.$$

3.3.3 Электрический потенциал (электростатика)

Предположим, что в диэлектрике, занимающем некоторую область D, известна плотность распределения электрического заряда $\rho_e(x, y)$. Напряженность электрического поля $\boldsymbol{E}$ связана с $\rho_e(x, y)$ одним из уравнений Максвелла

$$\operatorname{div}(\varepsilon \boldsymbol{E}) = \rho_e, \tag{3.31}$$

где ε — диэлектрическая проницаемость области D ($\varepsilon > 0$).

Электрическое поле называется потенциальным, если напряженность электрического поля может быть представлена в виде

$$\boldsymbol{E} = -\nabla\varphi, \tag{3.32}$$

где φ — потенциал электрического поля.

Для уравнений (3.31), (3.32) на фрагментах границы Γ_1 можно задавать краевые условия Дирихле

$$\varphi\big|_{\Gamma_1} = \varphi_0(x, y)\big|_{\Gamma_1}, \tag{3.33}$$

где $\varphi_0(x, y)$ — заданное распределение электрического потенциала на Γ_1.

Для остальной части границы $\Gamma_2 = \partial D \setminus \Gamma_1$ можно задавать краевые условия Неймана

$$(\boldsymbol{n} \cdot \nabla\varphi)\big|_{\Gamma_2} = \frac{\partial\varphi}{\partial n}\Big|_{\Gamma_2} = \sigma_e(x,y)\big|_{\Gamma_2}, \tag{3.34}$$

где $\sigma_e(x,y)$ — заданное распределение поверхностного заряда на Γ_2 (если диэлектрик граничит с проводником, то $\sigma_e(x,y) = 0$).

Формально можно рассматривать и третью краевую задачу, считая, что плотность поверхностного заряда линейно зависит от потенциала на границе. Однако физическая трактовка такого условия весьма затруднительна.

3.3.4 Электрический потенциал (проводимость)

Пусть $\boldsymbol{j}$ — плотность электрического тока, протекающего через некоторую область D. Уравнение неразрывности электрического тока (закон сохранения заряда) имеет вид

$$\operatorname{div} \boldsymbol{j} = 0. \tag{3.35}$$

Плотность электрического тока связана с напряженностью электрического поля $\boldsymbol{E}$ законом Ома

$$\boldsymbol{j} = \sigma \boldsymbol{E}, \tag{3.36}$$

где σ — электрическая проводимость области D ($\sigma \geqslant 0$).

В случае потенциального электрического поля имеем

$$\boldsymbol{j} = -\sigma\nabla\varphi, \quad \boldsymbol{E} = -\nabla\varphi. \tag{3.37}$$

Для уравнений (3.35)–(3.37) с физической точки зрения разумно ставить лишь первые и вторые краевые условия

Далее считаем, что граница области D состоит из двух, возможно несвязных, фрагментов (некоторые Γ_i могут быть пустыми множествами)

$$\partial D = \Gamma_1 \cup \Gamma_2.$$

Краевые условия первого рода, соответствующие заданию потенциала на границе области, имеют вид

$$\varphi\big|_{\Gamma_1} = \varphi_0(x,y)\big|_{\Gamma_1}, \tag{3.38}$$

где $\varphi_0(x,y)$ — заданное распределение электрического потенциала на Γ_1.

Заметим, что физически реализовать распределение потенциала, зависящее от координат, на каком-либо участке границы проводящей области довольно трудно. Чаще всего предполагается, что граница Γ_1 состоит из нескольких несвязных фрагментов

$$\Gamma_1 = \Gamma_{11} \cup \Gamma_{12} \ldots \Gamma_{1m}, \quad \Gamma_{1i} \cap \Gamma_{1j} = \varnothing, \quad i \neq j.$$

На каждом фрагменте Γ_{1k} задается ***постоянное*** значение потенциала

$$\varphi\big|_{\Gamma_{1k}} = \varphi_k = \text{const}, \quad k = 1, 2, \ldots, m. \tag{3.39}$$

Краевые условия второго рода, соответствующие заданию плотности электрического тока, протекающего через границу области, имеют вид

$$(\boldsymbol{j}\cdot\boldsymbol{n})\big|_{\Gamma_2}= j_n^0(x,y)\big|_{\Gamma_2}, \tag{3.40}$$

где $j_n^0(x,y)$ — заданная плотность электрического тока через границу.

Случай $j_n^0 = 0$ соответствует электрически ***изолированным*** участкам границы. Также как и для краевых условий первого рода, чаще всего считается, что граница Γ_2 состоит из нескольких несвязных фрагментов.

3.3.5 Потенциальное течение несжимаемой жидкости

Течение жидкости называется потенциальным, если скорость течения $\boldsymbol{v}$ можно представить в виде

$$\boldsymbol{v} = \nabla\varphi. \tag{3.41}$$

Уравнение неразрывности для несжимаемой жидкости имеет вид

$$\operatorname{div}\boldsymbol{v} = 0. \tag{3.42}$$

Подставляя (3.41) в (3.42), получим

$$\Delta\varphi = 0. \tag{3.43}$$

Уравнение Бернулли, связывающее давление p со скоростью течения $\boldsymbol{v}$, записывается в форме (см. [16])

$$p + \frac{1}{2}\rho\boldsymbol{v}^2 = \text{const}, \tag{3.44}$$

где ρ — плотность жидкости ($\rho = \text{const}$).

Это соотношение, справедливое для стационарного течения невязкой жидкости, легко выводится из стационарных уравнений Эйлера

$$\rho\boldsymbol{v}\cdot\nabla\boldsymbol{v} = -\nabla p \tag{3.45}$$

или в декартовых координатах (подразумевается суммирование по повторяющимся индексам)

$$\rho v_k\frac{\partial v_i}{\partial x_k} = -\frac{\partial p}{\partial x_i}. \tag{3.46}$$

Подставляя (3.41) в (3.46), получим

$$\rho\frac{\partial\varphi}{\partial x_k}\frac{\partial}{\partial x_k}\frac{\partial\varphi}{\partial x_i} = -\frac{\partial p}{\partial x_i}, \quad \rho\frac{\partial\varphi}{\partial x_k}\frac{\partial}{\partial x_i}\frac{\partial\varphi}{\partial x_k} = -\frac{\partial p}{\partial x_i}, \quad \frac{1}{2}\rho\frac{\partial}{\partial x_i}\left(\frac{\partial\varphi}{\partial x_k}\right)^2 = -\frac{\partial p}{\partial x_i}.$$

Интегрируя, выводим (3.44)

$$\frac{1}{2}\rho\frac{\partial}{\partial x_i}\left(\frac{\partial\varphi}{\partial x_k}\right)^2 = -\frac{\partial p}{\partial x_i}, \quad \frac{1}{2}\rho\left(\frac{\partial\varphi}{\partial x_k}\right)^2 + p = \text{const}, \quad \frac{1}{2}\rho\boldsymbol{v}^2 + p = \text{const}.$$

В двумерном случае уравнению неразрывности можно удовлетворить, вводя функцию тока $\psi(x, y)$

$$u = \psi_y, \quad w = -\psi_x, \quad \boldsymbol{v} = (u, w), \tag{3.47}$$

где u, w — соответственно, компоненты скорости вдоль осей x и y.

Действительно, при дополнительных требованиях на гладкость (для возможности изменения порядка дифференцирования) имеем

$$\operatorname{div} \boldsymbol{v} = u_x + w_y = \psi_{yx} - \psi_{xy} \equiv 0.$$

Для функции тока ψ в случае потенциального течения справедливо уравнение Лапласа

$$\Delta\psi = 0, \quad (u_y - w_x = \psi_{yy} + \psi_{xx}, \quad u_y - w_x = \varphi_{xy} - \varphi_{yx} = 0). \tag{3.48}$$

Для задач гидродинамики довольно затруднительно перечислить типичные виды краевых условий. Эти условия могут быть и первого, и второго, и третьего рода. Ограничимся лишь рассмотрением одной из задач, возникающих в области аэродинамики, напомнив, что рассматриваемые уравнения справедливы не только для жидкостей, но и для не очень разреженных газов, таких, например, как воздух.

3.3.5.1 Стационарное обтекание крыла

Предположим, что в неограниченной области имеется некоторый контур W (wing), моделирующий крыло (см. рис. 3.8), и обтекаемый потоком газа (воздуха). Считаем, что контур непроницаем для газа и вдали от контура (на бесконечности) скорость течения постоянна и направлена вдоль некоторого вектора

$$\boldsymbol{v}\big|_\infty = \boldsymbol{v}_\infty, \quad \boldsymbol{v}_\infty = (u_\infty, w_\infty), \quad \operatorname{tg}\alpha = \frac{w_\infty}{u_\infty}, \tag{3.49}$$

где u_∞, w_∞ — заданы и α — угол наклона направления течения к оси x.

Воспользовавшись определением (3.47), легко убедиться, что функция тока $\psi(x, y)$ на бесконечности задается соотношением (с точностью до несущественной константы)

$$\psi\big|_\infty = \psi_\infty = u_\infty y - w_\infty x. \tag{3.50}$$

Для определения ψ имеем следующую задачу

$$\Delta\psi = 0, \quad \psi\big|_W = 0, \quad \psi\big|_\infty = \psi_\infty. \tag{3.51}$$

Краевое условие $\psi|_W = 0$ на контуре W отвечает случаю контура, непроницаемого для газа.

Решив задачу (3.51) и используя (3.47), можно найти поле скоростей и при помощи уравнения Бернулли (3.44) определить давление p (естественно, с точностью до константы)

$$\boldsymbol{v} = (u, w) = (\psi_y, -\psi_x), \quad p = -\frac{1}{2}\boldsymbol{v}^2 = -\frac{1}{2}\left(\psi_x^2 + \psi_y^2\right). \tag{3.52}$$

Ясно, что невозможно численно решать задачу (3.51) в бесконечной области. Поэтому условие $\psi|_\infty = \psi_\infty$ заменим условием на некоторой окружности достаточно большого радиуса R

$$\psi\big|_{(x^2+y^2)=R^2} = \psi_\infty.$$

Приведем код программы на языке FreeFem++, задавая, для определенности, в качестве контура W так называемый профиль крыла NACA2412 (способ конструирования различных профилей приведен на с. 57) и выбирая следующие параметры

$$R = 5, \quad u_\infty = 1{,}0, \quad w_\infty = 0{,}5.$$

```
int Sp=97, Sm=98; // метки контура
int n=1;
real R=5, R0=0.8, shift=0.5; // параметры
real p0, m0, t0; // параметры контура крыла
// NACA2412
m0 = 0.01*2;      p0 = 0.1*4;     t0 = 0.01*12;
// функция для задания толщины крыла
func real Yt(real t)
{ return  5*t0*(0.296900*sqrt(t)- 0.126556*t - 0.356307*t^2
          + 0.290672*t^3 - 0.104709*t^4);                    }
// функция для задания средней линии крыла
func real Yc(real t)
{ return (0<=t)*(t<=p0)*(m0*(1/(p0^2))*(2*p0*t -t^2))+
         (p0<=t)*(t<=1.0)*(m0*((1/(1-p0)^2)*((1-2*p0)+2*p0*t-t^2))); }
//  theta=arctg(dYc/dt)
func real theta(real t)
{ return   atan((0<=t)*(t<=p0)*(m0*(1/(p0^2))*(2*p0 -2*t))+
           (p0<=t)*(t<=1.0)*(m0*((1/(1-p0)^2)*(2*p0-2*t))));    }
// задание профиля крыла
border C(t=0,2*pi) { x=5*cos(t); y=5*sin(t);}
border WingUp(t=0,1){x = t-Yt(t)*sin(theta(t));
                     y = Yc(t)+ Yt(t)*cos(theta(t)); label=Sp; }
border WingDn(t=1,0){x = t+Yt(t)*sin(theta(t));
                     y = Yc(t)- Yt(t)*cos(theta(t)); label=Sm; }
mesh Th= buildmesh(C(50*n)+WingUp(70*n)+WingDn(70*n));
// задание  пространства конечных элементов
fespace Vh(Th,P2); Vh psi,v;
// вспомогательный контур для визуализации в окрестности крыла
border c0(t=0,2*pi) { x=R0*cos(t)+shift; y=R0*sin(t); }
mesh Zoom = buildmesh(c0(30*n)+WingUp(35*n)+WingDn(35*n));
plot(c0(50*n)+WingUp(70*n)+WingDn(70*n),wait=1);
fespace ZVh(Zoom,P2);
ZVh Zpsi,Zcp,Zu,Zw;
// направление потока воздуха
real Uinfty = 1.0,  Winfty = 0.5;
solve potential(psi,v) = int2d(Th)(dx(psi)*dx(v)+dy(psi)*dy(v))
                       + on(C, psi = Uinfty*y-Winfty*x)
                       + on(Sp, Sm, psi=0);
```

```
39 // вычисление давления и компонент скорости
40 Vh u = dy(psi), w = -dx(psi), cp = -u^2-w^2;
41 // вспомогательные операторы для визуализации
42 Zpsi = psi;   Zu = u;   Zw = w;   Zcp = cp;
43 plot(Zpsi, wait=1, bw=1, nbiso=50);
44 plot(Zcp, wait=1, bw=1, nbiso=40);
45 plot([Zu,Zw], wait=1, bw=1, coef=0.02);
```

Результаты расчетов представлены на рис. 3.8.

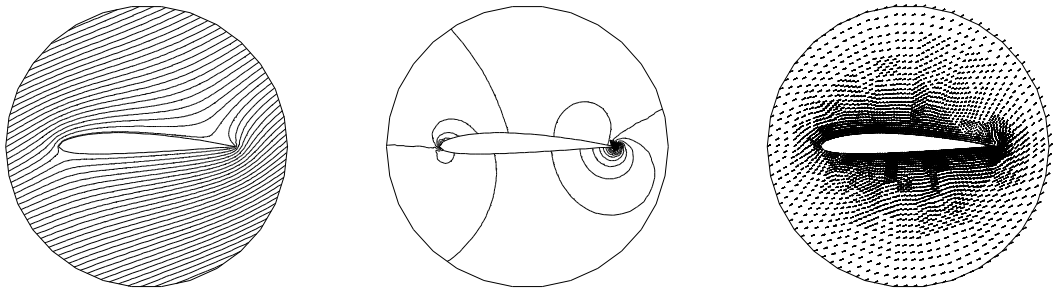

Рис. 3.8. Изолинии функции тока, давления и поле скоростей при обтекании крыла

Коротко опишем один из способов конструирования профилей крыла, известных как семейство профилей `NACA 4 digits`[1].

Задается стандартная функция $y_t(t)$, называемая толщиной крыла, и функция $y_c(t)$, называемая средней линией контура крыла (chamber line or mean line)

$$y_t(t) = 5T(0{,}296900\sqrt{t} - 0{,}126556\,t - 0{,}356307\,t^2 + 0{,}290672\,t^3 - 0{,}104709\,t^4),$$

$$y_c(t) = \begin{cases} \dfrac{m}{p^2}(2pt - t^2), & 0 \leqslant t \leqslant p, \\ \dfrac{m}{(1-p^2)}(1 - 2p + 2pt - t^2), & p \leqslant t \leqslant 1. \end{cases}$$

Вводится вспомогательная функция $\theta(t)$

$$\theta(t) = \operatorname{arctg}\left(\frac{dy_c(t)}{dt}\right), \quad \frac{dy_c(t)}{dt} = \begin{cases} \dfrac{2m}{p^2}(p - t), & 0 \leqslant t \leqslant p, \\ \dfrac{2m}{(1-p^2)}(p - t), & p \leqslant t \leqslant 1. \end{cases}$$

Координаты верхней (upper) и нижней (lower) частей контура крыла определяются параметрическими уравнениями

$$x_{\text{upper}} = t - y_t(t)\sin(\theta(t)), \quad y_{\text{upper}} = y_c(t) + y_t(t)\cos(\theta(t)), \quad 0 \leqslant t \leqslant 1,$$
$$x_{\text{lower}} = t + y_t(t)\sin(\theta(t)), \quad y_{\text{lower}} = y_c(t) - y_t(t)\cos(\theta(t)), \quad 0 \leqslant t \leqslant 1.$$

[1] См., например, `http://www-berkeley.ansys.com/cfd/naca.html`

Параметры T, m, p, входящие в указанные соотношения, вычисляются по цифровому коду профиля. Например, для профиля крыла `NACA2412`, используя цифры кода в качестве числителей дробей, получим (см. рис. 3.9)

$$m = \frac{2}{100}, \quad p = \frac{4}{10}, \quad T = \frac{12}{100}.$$

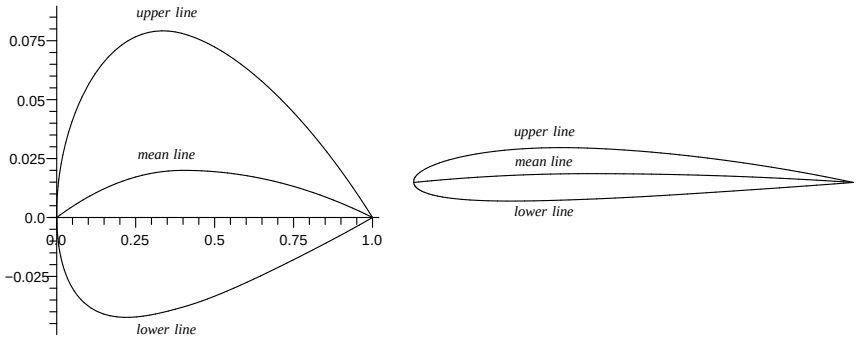

Рис. 3.9. Профиль крыла NACA2412 (справа в масштабе 1:1)

На рис. 3.10 для сравнения показаны некоторые профили крыла `NACA`.

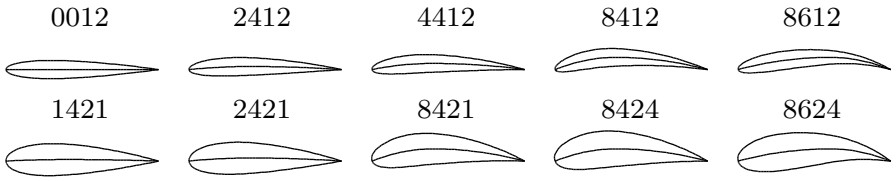

Рис. 3.10. Различные профили крыла NACA

Глава 4

Решение нестационарной задачи для уравнения Лапласа

FreeFem++ не дает возможности непосредственно решать нестационарные уравнения, т. е. в языке не предусмотрены специальные операторы для этой цели. Для того, чтобы решать нестационарную задачу при помощи FreeFem++, необходимо составить некоторую программу, позволяющую пошагово находить решение в различные моменты времени t.

Цель настоящей главы — демонстрация использования FreeFem++ для решения 2D нестационарной краевой задачи на примере уравнения Лапласа. Также как и в стационарном случае, задача для уравнения Лапласа позволяет описывать множество различных физических процессов, таких как нестационарное распределение температуры, нестационарное распределение концентрации и т. п. Заметим, что краевые условия для нестационарных задач задаются также как и для стационарных (см. пп. 3.3.1, 3.3.2). Коды программ для решения задач при помощи FreeFem++ мало отличаются от кодов для стационарных задач — новыми будут лишь фрагменты кодов, при помощи которых осуществляется аппроксимация производных по времени и организация вычислений в различные моменты времени.

Далее, для определенности, рассматриваем задачу о нестационарном распределении температуры, хотя точно также будет решаться и задача о нестационарном распределении концентрации.

4.1 Постановка задачи

Нестационарное уравнение теплопроводности, описывающее изменение температуры $u(x, y, t)$ (или концентрации) с течением времени, имеет вид

$$\frac{\partial u}{\partial t} - \mu \Delta u = f(x, y, t), \quad (x, y) \in D, \quad t > 0, \qquad \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \quad (4.1)$$

где $\mu > 0$ — коэффициент температуропроводности, $f(x, y, t)$ — плотность распределения внутренних источников тепла, D — область, в которой исследуется распределение температуры.

Для простоты рассматриваем случай, когда на границе $\Gamma = \partial D$ области D поставлены лишь условия Дирихле, которые соответствуют заданию

температуры $g(x, y, t)$ на границе (см. пп. 3.3.1, 3.3.2)

$$u\big|_\Gamma = g(x, y, t)\big|_\Gamma. \tag{4.2}$$

Кроме этого, задаем начальное распределение температуры $h(x, y)$ в области D

$$u\big|_{t=0} = h(x, y), \quad (x, y) \in D. \tag{4.3}$$

Заметим, что при аналитическом решении задачи (4.1)–(4.3) *не требуется* выполнения так называемого *условия согласования* начального и краевого условий. Однако, при численном решении задачи желательно, чтобы условия согласования были выполнены — начальное распределение температуры должно удовлетворять краевым условиям в начальный момент времени

$$g(x, y, t)\big|_{t=0} = h(x, y), \quad (x, y) \in \Gamma.$$

4.2 Способ построения решения

Для решения задачи (4.1)–(4.3) при помощи FreeFem++ необходимо заменить производную по времени ее конечно-разностной аппроксимацией, записать слабую формулировку задачи и реализовать алгоритм последовательного решения задачи для различных моментов времени.

4.2.1 Аппроксимация производной по времени

Ищем решение задачи (4.1)–(4.3) на интервале времени $[0, T]$. Зададим на этом интервале набор значений $t_m = m\tau$, где τ — шаг по времени. Для функции u в момент времени t_m введем обозначения:

$$u^m(x, y) = u(x, y, t_m), \quad t_m = m\tau. \tag{4.4}$$

Аппроксимируем производную по времени конечной разностью

$$\left.\frac{\partial u(x, y, t)}{\partial t}\right|_{t=t_m} \approx \frac{u^{m+1}(x, y) - u^m(x, y)}{\tau}. \tag{4.5}$$

Перепишем задачу (4.1)–(4.3) в виде (аргументы x, y у функции u далее опущены)

$$\frac{u^{m+1} - u^m}{\tau} - \mu\Delta u^{m+1} = f^{m+1}, \qquad f^{m+1} = f(x, y, t_{m+1}), \quad (x, y) \in D, \tag{4.6}$$

$$u^{m+1}\big|_\Gamma = g(x, y, t_{m+1})\big|_\Gamma, \quad m = 0, 1, \ldots \tag{4.7}$$

$$u^0 = h(x, y). \tag{4.8}$$

Таким образом, если $u^m(x, y) = u(x, y, t_m)$ известно, то для определения функции $u^{m+1}(x, y) = u(x, y, t_{m+1})$ имеем «стационарную» краевую задачу

(4.6), (4.7). Используя (4.8) и решая (4.6), (4.7) при $m = 0, 1, \ldots$, получим приближенное решение нестационарной задачи.

Напомним, что аппроксимация вида (4.6)–(4.8) называется неявной — все члены уравнения, не содержащие производной по времени, выбираются в точке $t = t^{m+1}$, а аппроксимация производной по времени осуществляется в точке $t = t^m$.

4.2.2 Слабая формулировка задачи

Получим слабую формулировку задачи (4.6), (4.7). Умножая (4.6) на тестовую функцию $v(x, y)$ и интегрируя по области D с использованием формулы Грина, выводим

$$\iint\limits_D u^{m+1} v \, dx \, dy - \iint\limits_D u^m v \, dx \, dy + \mu\tau \iint\limits_D \nabla u^{m+1} \cdot \nabla v \, dx \, dy -$$

$$- \tau\mu \int\limits_\Gamma v \frac{\partial u^{m+1}}{\partial n} \, ds - \tau \iint\limits_D f^{m+1} v \, dx \, dy = 0, \quad \forall\, v.$$

Для удовлетворения краевому условию (4.7), потребуем выполнения аналогичного однородного условия для тестовой функции: $v|_\Gamma = 0$. Тогда исходная задача в слабой формулировке примет вид

$$\iint\limits_D \left(u^{m+1} v + \mu\tau \nabla u^{m+1} \cdot \nabla v\right) dx \, dy - \iint\limits_D \left(u^m + \tau f^{m+1}\right) v \, dx \, dy = 0, \quad (4.9)$$

$$u^{m+1}\big|_\Gamma = g(x, y, t_{m+1})\big|_\Gamma. \quad (4.10)$$

Подчеркнем, что для каждого t_{m+1} это обычная стационарная задача, которая может решаться стандартными методами языка FreeFem++. Дополнительно требуется лишь организовать последовательное решение задач для u^1, u^2, u^3,...

4.3 Реализация алгоритма на языке FreeFem++

Приведем код программы на языке FreeFem++, считая область D прямоугольником $\overline{D} = [0, a] \times [0, b]$.

Для простоты полагаем, что внутренние источники тепла отсутствуют

$$f = 0.$$

Начальное распределение температуры является периодическим

$$h(x, y) = \sin 2\pi x \sin 2\pi y.$$

Краевые условия выберем независящими от времени

$$g(x, y) = \begin{cases} 1, & 0 \leqslant x \leqslant a, & y = 0, & (\texttt{Bottom}), \\ 0, & 0 \leqslant x \leqslant a, & y = b, & (\texttt{Top}), \\ 0, & x = 0, & 0 \leqslant y \leqslant b, & (\texttt{Left}), \\ 0, & x = a, & 0 \leqslant y \leqslant b, & (\texttt{Right}). \end{cases}$$

Кроме этого, зададим параметры

$$\mu = 1, \quad \tau = 0{,}001.$$

Обратим внимание на то, что краевые и начальные условия не согласованы. В момент $t = 0$ начальное распределение температуры на фрагменте границы $0 \leqslant x \leqslant a$, $y = 0$ взято равным $u|_{t=0} = h(0 \leqslant x \leqslant a, y = 0) = 0$, хотя на этом фрагменте поддерживается температура $g(x, y) = 1$. Как будет видно из дальнейшего вычислительного эксперимента (см. п. 4.4), разрыв в краевых условиях и начальных данных, имеющийся при $t = 0$, будет сглажен уже в начальные моменты времени на нескольких первых шагах расчета по времени. В случае аналитического решения такое сглаживание происходит при $t \to +0$.

Код программы, соответствующий указанным значениям, имеет вид

```
real a = 1.0, b = 1.0;
int  n = 4;
real t, dt;
real mu = 1;
int  k = 0;
// задаем границы области (прямоугольник [0,a]x[0,b])
// следует сохранять ориентацию контура -- против часовой стрелки
border GammaB(t=0,1){ x=a*t;     y=0; };        // GammaB -- Bottom
border GammaR(t=0,1){ x=a;       y=b*t; };      // GammaR -- Right
border GammaT(t=0,1){ x=a*(1-t); y=b; };        // GammaT -- Top
border GammaL(t=0,1){ x=0;       y=b*(1-t); };  // GammaL -- Left
// строим сетку, на каждой границе по 5*n-узлов
mesh Th = buildmesh(GammaB(5*n)+GammaR(5*n)+GammaT(5*n)+GammaL(5*n));
fespace Vh(Th, P2);  // задаем пространство конечных элементов Vh
// на пространстве Vh задаем искомую функцию u, тестовую функцию v и
// вспомогательную функцию uOld
// используем обозначения: u=u(x,y,(m+1)*dt),  uOld=u(x,y,m*dt)
Vh u, v, uOld;
// определяем функцию -- начальное распределение
func h = sin(2*pi*x)*sin(2*pi*y);
// определяем функцию -- правую часть уравнения
func f = 0;
// определяем функции для задания граничных условий
func gB = 1;  func gR = 0;
func gT = 0;  func gL = 0;
// записываем слабую (вариационную) формулировку задачи
problem Heat(u,v) =
          int2d(Th)( u * v + mu*dt*(dx(u)*dx(v) + dy(u)*dy(v)) )
        - int2d(Th)( (uOld + dt * f) * v )
        + on( GammaB, u = gB ) + on( GammaR, u = gR )
        + on( GammaT, u = gT ) + on( GammaL, u = gL ) ;
t = 0;   dt = 0.001;
uOld = h;
// организуем пошаговое решение задачи
for (int m=0; m<=120; m++)
  {  t = t + dt;
```

```
37        k = k+1;
38        Heat; // вызов процедуры
39        uOld = u;
40        // организация вывода данных в файлы через 10 шагов
41        if (k==10)
42        { k = 0;
43          plot(u, bw=true, cmm=" t="+t, ps="Heat"+m+"");
44        }
45      }
```

Полезно сравнить приведенную программу с аналогичной программой решения стационарной задачи для уравнения Лапласа. Существенное отличие заключается лишь во фрагменте со строками 36–39, при помощи которых организовано последовательное решение задачи для различных моментов времени.

4.4 Вычислительный эксперимент

На рис. 4.1 приведены результаты расчетов распределения температуры в различные моменты времени в случае, когда температура «нижней»[1] границы области поддерживается постоянной и равна $u = 1$, а на всех остальных частях границы $u = 0$. Хорошо видно, что с течением времени начальное периодическое распределение температуры исчезает и уже при $t \approx 0{,}12$ (для выбранных параметров задачи) устанавливается стационарное распределение температуры.

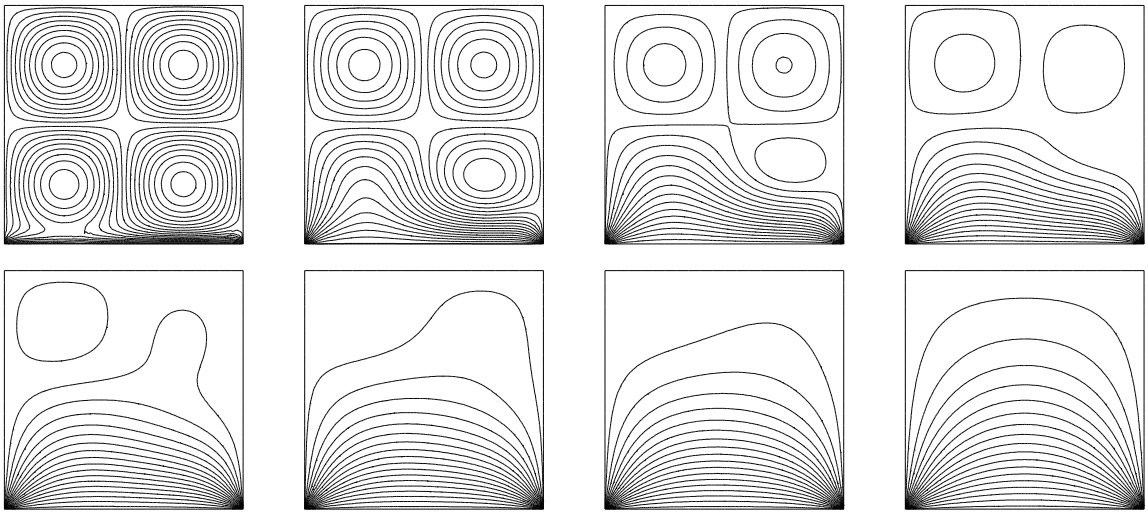

Рис. 4.1. Изолинии функции $u(x, y, t)$ при $t = 0$; 0,01; 0,02; 0,03; 0,04; 0,05; 0,06; 0,12

Аналогичная картина установления стационарного распределения температуры приведена на рис. 4.2 в случае, когда значение температуры на

[1] Напомним, что понятия «низ» и «верх» имеют смысл лишь при наличии силы тяжести.

верхней и нижней границах $u = 1$, на боковых границах — $u = 0$, а функции f, h и параметры μ, τ остаются прежними. Стационарное распределение наблюдается в данном случае также при $t \approx 0{,}12$. В коде программы для проведения расчетов достаточно изменить лишь строки 24, 25.

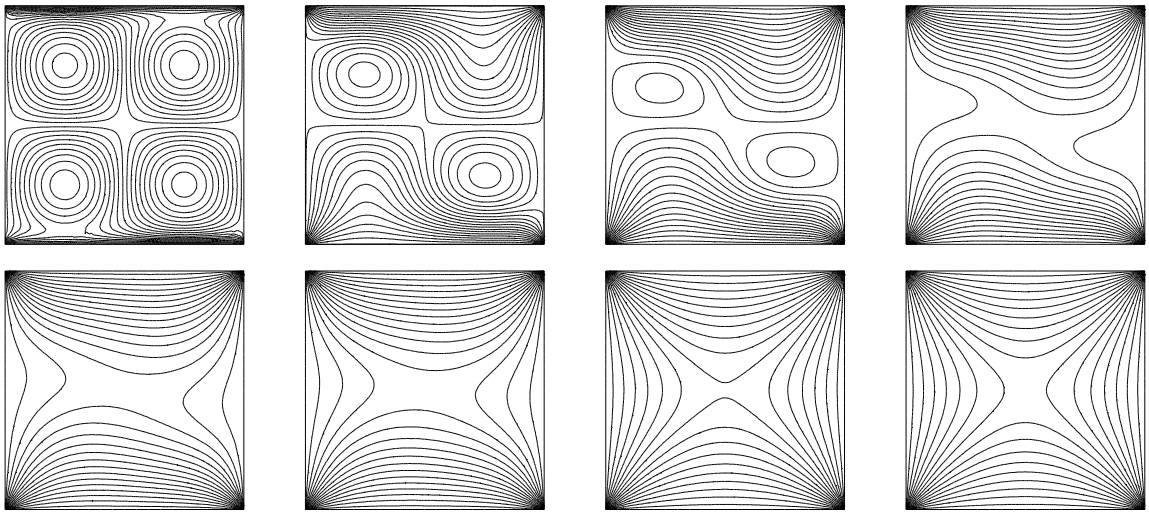

Рис. 4.2. Изолинии функции $u(x, y, t)$ при $t = 0$; 0,01; 0,02; 0,03; 0,04; 0,05; 0,11; 0,12

4.5 Общая схема аппроксимации производной по времени

Использованная в п. 4.2.1 для решения задачи неявная схема (4.6)–(4.8), конечно же, не является единственно возможной. Покажем некоторый, достаточно универсальный, подход к построению конечно-разностных одношаговых по времени аппроксимаций эволюционных задач (см. также [1]).

Рассмотрим следующую задачу

$$\frac{\partial u(t)}{\partial t} + Lu(t) = f(t), \quad u(0) = U, \tag{4.11}$$

где L — линейный (дифференциальный) оператор ($L\colon \mathbb{H} \to \mathbb{H}$), $\mathbb{H}$ — некоторое гильбертово пространство, выбранное таким образом, чтобы краевые условия для функции $u(t)$ были выполнены, U — известная функция.

Слабую (вариационнную) формулировку этой задачи получим, вычисляя скалярное произведение уравнения на тестовую функцию v

$$\frac{d}{dt}(u(t), v) + (Lu(t), v) = (f(t), v), \quad v \in \mathbb{H}. \tag{4.12}$$

Введем параметр $0 \leqslant \theta \leqslant 1$ и для дискретизации задачи (4.12) используем следующую формальную схему

$$\frac{1}{\tau}(u^{m+1} - u^m, v) + (Lu^{m+\theta}, v) = (f^{m+\theta}, v), \tag{4.13}$$

где

$$u^{m+\theta} = u(t_{m+\theta}), \quad f^{m+\theta} = f(t_{m+\theta}), \quad t_{m+\theta} = (m + \theta)\tau. \tag{4.14}$$

Иными словами, производная по времени аппроксимируется обычной конечной разностью первого порядка, а остальные члены уравнения берутся в момент времени $t_{m+\theta} \in [t_m, t_{m+1}]$.

Для аппроксимации $u^{m+\theta}$ и $f^{m+\theta}$ используем линейную интерполяцию

$$u^{m+\theta} = \theta u^{m+1} + (1-\theta)u^m, \quad f^{m+\theta} = \theta f^{m+1} + (1-\theta)f^m. \tag{4.15}$$

При $\theta = 0$ аппроксимация (4.13) называется явной схемой Эйлера, при $\theta = 1$ получим неявную схему Эйлера, а при $\theta = 1/2$ — схему Кранка–Никольсона.

Для рассматриваемой ранее задачи (4.1)–(4.3) вместо аппроксимации (4.6)–(4.8) можно использовать аппроксимацию вида (4.13)

$$\frac{u^{m+\theta} - u^m}{\tau} - \mu\Delta u^{m+\theta} = f^{m+\theta}, \tag{4.16}$$

$$u^{m+\theta}\big|_\Gamma = g(x, y, t_m + \theta\tau)\big|_\Gamma, \quad m = 0, 1, \ldots \tag{4.17}$$

$$u^0 = h(x, y). \tag{4.18}$$

Приведем фрагмент кода для решения задачи с использованием схемы Кранка–Никольсона. Этим фрагментом следует заменить строки 27–31 программы на с. 62.

```
real theta = 0.5; // схема Кранка-Никольсона
                  // theta = 1.0 -- неявная схема
problem Heat(u,v) =
   int2d(Th)( u * v + theta * mu * dt * (dx(u)*dx(v) + dy(u)*dy(v)) )
 + int2d(Th)( (1-theta) * mu * dt * (dx(uOld)*dx(v) + dy(uOld)*dy(v)) )
 - int2d(Th)( (uOld + theta * dt * f) * v )
 - int2d(Th)( ((1-theta) * dt * fm) * v )
 + on( GammaB, u = gB ) + on( GammaR, u = gR )
 + on( GammaT, u = gT ) + on( GammaL, u = gL ) ;
```

4.6 Контроль погрешности

Простейший способ контроля погрешности заключается в сравнении результатов вычислений для различных (в смысле размеров треугольников) триангуляций области D. Для примера, описанного в п. 4.3, приведем код программы, позволяющей сравнивать нормы решений, полученных на двух сетках. Будем использовать норму решения $u(x, y, t)$ в момент времени $t = t^m$ в пространстве L_2

$$\|u^m\|_{L_2}^2 = \iint\limits_D |u^m(x, y)|^2 \, dx \, dy. \tag{4.19}$$

Строки 9, 10 кода отвечают за создание сеток `Th1`, `Th2`, для которых характерные размеры треугольников отличаются друг от друга. На каждой из этих сеток задаются пространства конечных элементов `Vh1`, `Vh2` (строки

12, 13) и на этих пространствах — наборы функций u1, v1, uOld1 и u2, v2, uOld2, используемых при расчетах. Для каждой сетки записываются слабые формулировки исходной задачи Heat1(u1,v1) и Heat2(u2,v2) (строки 22–26, 27–31). При пошаговом решении задачи вызываются процедуры Heat1 и Heat2 (строки 38, 40). В строках 43, 44 вычисляются квадраты норм (см. (4.19)) в каждый момент времени $t = t^m$. Результаты сохраняются в файл (строка 45), описанный в строке 34. Кроме этого, оператор plot в строке 42 позволяет визуализировать на экране дисплея решения, построенные на двух разных сетках.

```
1  real a = 1.0, b = 1.0, mu = 1, t, dt;
2  int  n = 4;
3  real NormL21,  NormL22;
4  border GammaB(t=0,1){ x=a*t;     y=0; };        // GammaB -- Bottom
5  border GammaR(t=0,1){ x=a;       y=b*t; };      // GammaR -- Right
6  border GammaT(t=0,1){ x=a*(1-t); y=b; };        // GammaT -- Top
7  border GammaL(t=0,1){ x=0;       y=b*(1-t); };  // GammaL -- Left
8  // строим две различных сетки
9  mesh Th1 = buildmesh(GammaB(4*n)+GammaR(4*n)+GammaT(4*n)+GammaL(4*n));
10 mesh Th2 = buildmesh(GammaB(2*n)+GammaR(2*n)+GammaT(2*n)+GammaL(2*n));
11 // для каждой сетки задаем пространство конечных элементов
12 fespace Vh1(Th1, P2);
13 fespace Vh2(Th2, P2);
14 // на каждом пространстве задаем набор функций u, v и  uOld
15 Vh1 u1, v1, uOld1;
16 Vh2 u2, v2, uOld2;
17 // определяем функцию -- начальное распределение
18 func h = sin(2*pi*x)*sin(2*pi*y);
19 // определяем функцию -- правую часть уравнения
20 func f = 0;
21 // записываем задачи для различных сеток
22 problem Heat1(u1,v1) =
23         int2d(Th1)( u1 * v1 + mu*dt*(dx(u1)*dx(v1) + dy(u1)*dy(v1)) )
24       - int2d(Th1)( (uOld1 + dt * f) * v1 )
25       + on( GammaB, u1 = 1 )
26       + on( GammaR, GammaT, GammaL, u1 = 0 );
27 problem Heat2(u2,v2) =
28         int2d(Th2)( u2 * v2 + mu*dt*(dx(u2)*dx(v2) + dy(u2)*dy(v2)) )
29       - int2d(Th2)( (uOld2 + dt * f) * v2 )
30       + on( GammaB, u2 = 1 )
31       + on( GammaR, GammaT, GammaL, u2 = 0 );
32 t = 0;          dt = 0.001;
33 uOld1 = h;     uOld2 = h;
34 ofstream MyFile("DFile.txt"); // открываем файл для записи
35 // организуем пошаговое решение задачи
36 for (int m=0; m<=120; m++)
37   { t = t + dt;
38     Heat1; // вызов процедуры для сетки Th1
39     uOld1 = u1;
40     Heat2;  // вызов процедуры для сетки Th2
41     uOld2 = u2;
```

```
42      plot(u1, u2, bw=true, cmm=" t="+t);
43      NormL21 = int2d(Th1)(u1*u1);  // норма в L2
44      NormL22 = int2d(Th2)(u2*u2);
45      MyFile << t << "," << NormL21 << "," << (NormL22-NormL21) << "\n";
46    }
```

На рис. 4.3 показаны результаты вычислений нормы в L_2 (рис. A) и разности норм (рис. B, линия 1)

$$\delta = \|\overline{u}^m\|^2_{L_2} - \|u^m\|^2_{L_2}, \tag{4.20}$$

где $\overline{u}^m$ и u^m — соответственно, решения на сетках, показанных на рис. 4.4.

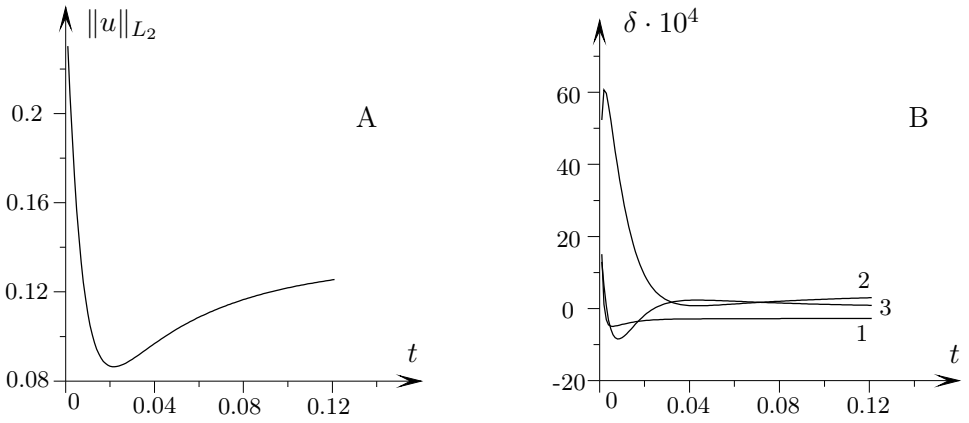

Рис. 4.3. Контроль погрешности

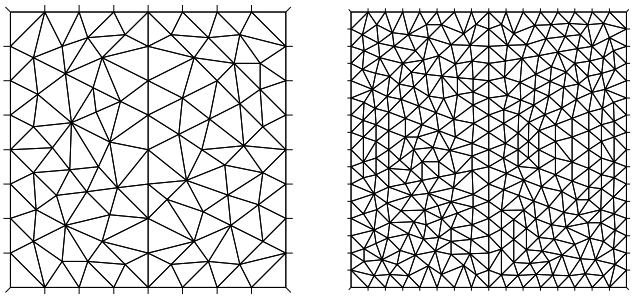

Рис. 4.4. Триангуляции `Th2` и `Th1`

Аналогичное сравнение решений можно провести на одной и той же сетке, выбирая различные пространства конечных элементов (см. рис. 4.3 B, линия 2), заменив строки 9–13 следующими

```
mesh Th1 = buildmesh(GammaB(4*n)+GammaR(4*n)+GammaT(4*n)+GammaL(4*n));
mesh Th2 = buildmesh(GammaB(4*n)+GammaR(4*n)+GammaT(4*n)+GammaL(4*n));
fespace Vh1(Th1, P1);
fespace Vh2(Th2, P2);
```

Кроме этого, несложно написать код программы, позволяющей проводить вычисления для различных шагов по времени. На рис. 4.3В (линия 3) показаны результаты такого сравнения для $\tau = 0{,}001$ и $\tau = 0{,}0005$.

Укажем и другие приемы контроля погрешности. В частности, при вычислении интегралов можно использовать разнообразные квадратурные формулы (см. п. 18.5). Можно также вычислять и другие нормы, например, в пространстве H_1

$$\|u^m\|^2_{H_1} = \iint\limits_D \nabla u^m \cdot \nabla u^m \, dx \, dy,$$

записав код

```
NormaH1 = int2d(Th)( dx(u)*dx(u) + dy(u)*dy(u) );
```

Конечно, более корректно оценивать совпадение решений не по разности норм (4.20), а по норме разности

$$\delta_0 = \|\overline{u}^m - u^m\|^2_{L_2}.$$

FreeFem++ позволяет вычислять интегралы от функций, заданных на неодинаковых сетках

```
NormaL22 = int2d(Th2)( (u2-u1) * (u2-u1) );
```

Здесь `u1`, `u2` заданы на сетках `Th1`, `Th2`, соответственно. Однако следует иметь ввиду, что при вычислении функций в случае несовпадающих сеток используется алгоритм интерполяции (см. гл. 18 и [1]) и вычисление разности функций может осуществляться с большой погрешностью.

Все перечисленные способы контроля погрешности, по существу, позволяют контролировать лишь вычислительную погрешность, а не погрешность алгоритма. Надежнее было бы сравнивать результаты решения задач, полученные с помощью разнообразных алгоритмов, например, используя при аппроксимации явную и неявную схемы.

Наконец, заметим, что вычисляя норму решения (см. рис. 4.3А), можно получить дополнительную информацию о задаче. Так, для нестационарной задачи из п. 4.3 поведение нормы косвенно указывает, что решение $u(x, y, t)$ с течением времени становится стационарным.

Глава 5

Уравнение переноса

В задачах математической физики при описании движения или переноса в сплошной среде часто встречается так называемая материальная (или субстанциональная) производная, которая имеет вид

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla.$$

Например, для описания переноса примеси с концентрацией $u(\boldsymbol{x}, t)$ в заданном поле скорости $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}, t)$ при отсутствии процессов диффузии используется дифференциальное уравнение в частных производных первого порядка

$$u_t + \boldsymbol{v} \cdot \nabla u = f,$$

где $f = f(\boldsymbol{x}, t)$ — источник примеси.

Другой, более сложный пример — уравнение Эйлера для описания движения идеальной жидкости

$$\frac{d\boldsymbol{v}}{dt} = -\nabla p + f,$$

где p — давление, f — сила, действующая на жидкость.

FreeFem++ предоставляет пользователю конструкцию `convect[...]`, позволяющую эффективно вычислять материальную производную. Математический аппарат, использованный для создания алгоритма вычислений, тесно связан с методом характеристик для уравнений гиперболического типа в частных производных первого порядка.

5.1 Постановка задачи Коши для гиперболических уравнений

Пусть дано гиперболическое уравнение в частных производных первого порядка

$$u_t + \boldsymbol{v} \cdot \nabla u = f. \tag{5.1}$$

Ограничимся рассмотрением пространственно одномерного случая и для определенности считаем, что исследуется процесс переноса примеси с концентрацией $u(\boldsymbol{x}, t)$ в поле скорости $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}, t)$. Источник (или сток)

примеси задан известной функцией $f(\boldsymbol{x}, t)$. Заметим, что в общем случае скорость переноса может зависеть и от концентрации u, т. е. $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}, t, u)$.

Для однородного уравнения (5.1) поставим задачу Коши, задав начальное условие (конечно, это можно сделать и для неоднородного уравнения)

$$u_t + v(x,t)u_x = 0, \quad -\infty < x < +\infty, \quad t > 0, \tag{5.2}$$

$$u(x,0) = g(x), \quad -\infty < x < +\infty. \tag{5.3}$$

Теория решения такой задачи, даже в случае $v = v(x,t,u)$, хорошо развита (см., например, [17, 18]).

В случае, например, области $0 \leqslant x < +\infty$ можно поставить и краевую задачу, задав в точке $x = 0$ условие $u(0,t) = h(t)$. Такая задача, на самом деле, также является задачей Коши. Дело в том, что переменные x, t в уравнении (5.2) лишь условно считаются координатой и временем. Производные u_t и u_x входят в уравнения достаточно равноправным образом и с таким же успехом можно считать, что t это координата, а x — время.

5.2 Метод характеристик

Коротко изложим суть метода характеристик, позволяющего получить решение задачи (5.2), (5.3), по крайней мере, локально по времени.

Пусть на плоскости (x,t) задана достаточно гладкая линия Γ. Предположим, что известны параметрические уравнения, задающие эту линию (см. рис. 5.1)

$$\Gamma\colon\ x = X(\tau), \quad t = t(\tau). \tag{5.4}$$

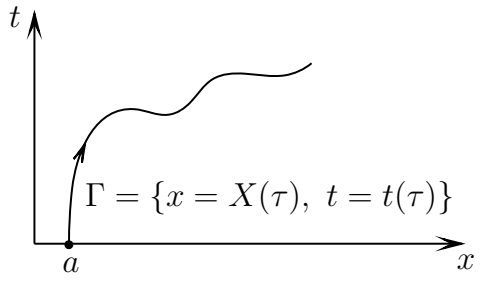

Рис. 5.1. Характеристика на плоскости (x,t)

Рассмотрим функцию $u(x,t)$ на линии Γ

$$u\big|_{\Gamma} = u(X(\tau), t(\tau)) = U(\tau). \tag{5.5}$$

Предполагая, что u — достаточно гладкая функция двух переменных, найдем производную функции u вдоль линии Γ

$$U_\tau(\tau) = u_t t_\tau + u_x X_\tau. \tag{5.6}$$

Потребуем выполнения следующих соотношений

$$t_\tau = 1, \quad X_\tau = v(X(\tau), t(\tau)). \tag{5.7}$$

Тогда формула (5.6) совпадает с левой частью уравнения (5.2) и в силу этого уравнения производная $U_\tau(\tau) = 0$

$$U_\tau(\tau) = u_t + vu_x = 0. \tag{5.8}$$

Это означает, что вдоль линии Γ значения функции $u(x,t)$ не изменяются. Такая линия называется ***характеристикой***. Уравнения (5.7) называются уравнениями характеристики и задают семейство линий Γ на плоскости (x,t).

Из уравнения $t_\tau = 1$ (см. (5.7)) следует, что $\tau = t + \text{const}$. Без потери общности константу можно считать равной нулю, т. к. это будет означать лишь замену переменной t (сдвиг по времени). Далее полагаем, что

$$\tau = t. \tag{5.9}$$

Действуя формально, можно ввести оператор дифференцирования

$$\frac{d}{dt} = \frac{\partial}{\partial t} + v\frac{\partial}{\partial x}. \tag{5.10}$$

Применяя этот оператор к функции u, получим

$$\frac{du}{dt} = u_t + vu_x = 0. \tag{5.11}$$

Аналогично, действуя этим оператором на x и учитывая, что x и t — независимые переменные, получим уравнение характеристики

$$\frac{dX(t)}{dt} = \frac{dx}{dt} = x_t + vx_x = v(X(t),t). \tag{5.12}$$

Таким образом, уравнение в частных производных первого порядка (5.2) сводится к системе двух *обыкновенных* дифференциальных уравнений (5.11) и (5.12)

$$\frac{dX(t)}{dt} = v(X(t),t), \quad \frac{du(X(t),t)}{dt} = 0. \tag{5.13}$$

Для того, чтобы сформулировать задачу Коши (5.2), (5.3) в терминах обыкновенных дифференциальных уравнений, рассмотрим линию Γ (характеристику), проходящую через точку $x = a$ на оси абсцисс (см. рис. 5.1). Тогда начальным условием для уравнения этой характеристики будет условие $X(0) = a$. Начальное условие для функции u получим, используя (5.3). При $t = 0$ имеем $x = X(0) = a$ и $u(x,0) = u(X(0),0) = g(x) = g(X(0)) = g(a)$.

Окончательно запишем задачу Коши для системы обыкновенных дифференциальных уравнений

$$\frac{dX(t)}{dt} = v(X(t),t), \quad X(0) = a, \tag{5.14}$$

$$\frac{du}{dt} = 0, \quad u(0) = g(a). \tag{5.15}$$

Данная задача в рассматриваемом случае легко решается. В частности, из уравнения и начального условия (5.15) сразу же следует, что $u = g(a)$ при всех t. Это означает, что вдоль каждой характеристики функция u не изменяется и сохраняет свое первоначальное значение.

Обозначим $X(t) = F(t; a)$ решение задачи (5.14). Тогда уравнение характеристики, проходящей через точку $(a, 0)$ на плоскости (x, t), задается соотношением

$$x = F(t; a) \quad (a = F(0; a)). \tag{5.16}$$

Считаем, что это уравнение разрешимо относительно a, т. е. $a = a(x, t)$. Тогда решением исходной задачи (5.2), (5.3) будет

$$u(x, t) = g(a(x, t)). \tag{5.17}$$

Читатель, знакомый с основами механики сплошной среды, легко узнает в описанном способе решения переход от эйлеровых переменных x к лагранжевым переменным a. Исходная задача (5.2), (5.3) записана в эйлеровых переменных — слежение за изменением функции $u(x, t)$ осуществляется в каждый момент времени t в точке x. Задача (5.14), (5.15) — это исходная задача, записанная в лагранжевых переменных. В этом случае осуществляется слежение за изменением вдоль характеристики функции $u(a(x, t))$, которая в начальный момент времени находилась в точке $x = a(x, 0) = a$.

Пример 5.1. Пусть поле скорости постоянно, т. е. $v = \text{const}$. Тогда решением задачи (5.14) будет $X(t) = F(t; a) \equiv vt + a$. Уравнение характеристики на плоскости (x, t) имеет вид $x = a + vt$ — это прямая, проходящая через точку $(a, 0)$. При изменении параметра a получим семейство параллельных прямых на плоскости (x, t). Решая уравнение характеристик относительно a, имеем $a = x - vt$. Окончательно для функции u имеем $u(x, t) = g(x - vt)$. Это означает, что с течением времени начальное распределение $g(x)$ переносится вдоль оси x со скоростью v (бегущая волна). На рис. 5.2 показано семейство характеристик и движение начального профиля функции $u(x, t)$.

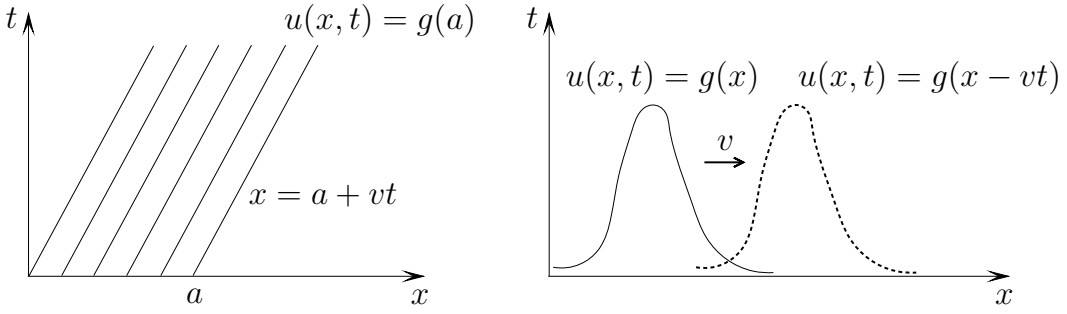

Рис. 5.2. Характеристики на плоскости (x, t) и движение профиля функции $u(x, t)$

5.3 Метод характеристик (двумерный случай)

Аналогичным образом можно получить решение задачи и в двумерном случае. Ограничимся лишь записью соответствующих соотношений для

уравнений (5.1). Введем дифференциальный оператор

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla. \tag{5.18}$$

Система обыкновенных дифференциальных уравнений аналогичная (5.13) запишется в виде

$$\frac{d\boldsymbol{X}(t)}{dt} = \boldsymbol{v}(\boldsymbol{X}(t), t), \quad \frac{du(\boldsymbol{X}(t), t)}{dt} = f(\boldsymbol{X}(t), t). \tag{5.19}$$

Пример 5.2. Пусть для уравнений (5.19) поставлена задача Коши, т. е. заданы начальные условия

$$\boldsymbol{X}(0) = \boldsymbol{a}, \quad u\big|_{t=0} = g(a_1, a_2), \quad \boldsymbol{a} = (a_1, a_2), \quad \boldsymbol{X} = (X_1, X_2), \quad X_1 = x, \quad X_2 = y.$$

Рассмотрим случай, когда поле скоростей задано в виде (твердотельное вращение по часовой стрелке при $\omega > 0$)

$$\boldsymbol{v} = (\omega y, -\omega x),$$

где ω — известная угловая скорость вращения.

Тогда, подставляя $\boldsymbol{v}$ в (5.19), получим

$$\frac{dX_1}{dt} = \omega X_2, \quad \frac{dX_2}{dt} = -\omega X_1.$$

Легко проверить, что с учетом начальных условий решение записывается в виде

$$X_1 = a_1 \cos \omega t + a_2 \sin \omega t, \quad X_2 = -a_1 \sin \omega t + a_2 \cos \omega t.$$

Эти соотношения определяют характеристики, проходящие через точку (a_1, a_2). В данном случае это окружность, показанная на рис. 5.3.

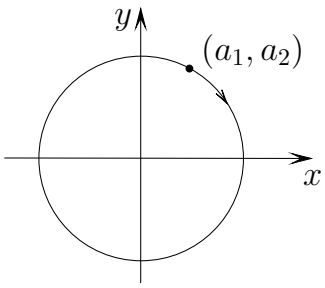

Рис. 5.3. Характеристика на плоскости (x, y)

Введем матрицу поворота (или вращения) $\boldsymbol{\Omega}$

$$\boldsymbol{\Omega} = \begin{pmatrix} \cos \omega t & \sin \omega t \\ -\sin \omega t & \cos \omega t \end{pmatrix}, \quad \boldsymbol{\Omega}\boldsymbol{\Omega}^T = \boldsymbol{I}, \quad \boldsymbol{\Omega}^{-1} = \boldsymbol{\Omega}^T.$$

Тогда связь между векторами $\boldsymbol{X}$ и $\boldsymbol{a}$ задается соотношениями

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} \cos \omega t & \sin \omega t \\ -\sin \omega t & \cos \omega t \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \quad \boldsymbol{X} = \boldsymbol{\Omega}\boldsymbol{a}, \quad \boldsymbol{a} = \boldsymbol{\Omega}^T \boldsymbol{X}.$$

Величины a_1, a_2 выражаются через X_1, X_2 при помощи формул

$$a_1 = X_1 \cos \omega t - X_2 \sin \omega t, \quad a_2 = X_1 \sin \omega t + X_2 \cos \omega t.$$

Функция $u(\boldsymbol{x}, t)$ сохраняется вдоль каждой характеристики и имеет вид

$$u(\boldsymbol{x}, t) = u(x, y, t) = g(a_1, a_2) = g(x \cos \omega t - y \sin \omega t, x \sin \omega t + y \cos \omega t).$$

5.4 Аппроксимация уравнения переноса

Приведем некоторые соотношения, которые потребуются в дальнейшем. Рассмотрим задачу Коши

$$\frac{d\boldsymbol{X}(t)}{dt} = \boldsymbol{v}(\boldsymbol{X}(t), t), \quad \boldsymbol{X}(t_{m+1}) = \boldsymbol{x}, \quad t_{m+1} = (m+1)\tau. \tag{5.20}$$

Если τ достаточно мало, то решение этой задачи в точке $t = t_m = m\tau$, используя разложение в ряд Тейлора, запишем в виде

$$\boldsymbol{X}(t_m) = \boldsymbol{X}(t_{m+1}) - \tau\frac{d\boldsymbol{X}(t_{m+1})}{dt} + \mathcal{O}(\tau^2) = \boldsymbol{x} - \tau\boldsymbol{v}^m(\boldsymbol{x}) + \mathcal{O}(\tau^2), \tag{5.21}$$

$$\boldsymbol{v}^m(\boldsymbol{x}) = \boldsymbol{v}(\boldsymbol{X}(t_{m+1}), t_m) = \boldsymbol{v}(\boldsymbol{x}, t_m).$$

Как и прежде (см. п. 4.2.1), введем обозначения

$$u^m(\widetilde{\boldsymbol{x}}) = u(\widetilde{\boldsymbol{x}}, t_m), \quad \boldsymbol{X}^m(\boldsymbol{x}) = \boldsymbol{X}(t_m). \tag{5.22}$$

Значения функции $u^m(\boldsymbol{X}^m(\boldsymbol{x}))$ найдем, вновь используя разложение в ряд

$$\begin{aligned} u^m(\boldsymbol{X}^m(\boldsymbol{x})) &= u^m(\boldsymbol{x} - \tau\boldsymbol{v}^m(\boldsymbol{x})) + \mathcal{O}(\tau^2) = \\ &= u^m(\boldsymbol{x}) - \tau\boldsymbol{v}^m(\boldsymbol{x}) \cdot \nabla u^m(\boldsymbol{x}) + \mathcal{O}(\tau^2). \end{aligned} \tag{5.23}$$

В языке FreeFem++ для вычисления величины $u^m(\boldsymbol{X}^m(\boldsymbol{x}))$ имеется специальная конструкция `convect`

$$u^m(\boldsymbol{X}^m(\boldsymbol{x})) = \texttt{convect}([v_1^m(\boldsymbol{x}), v_2^m(\boldsymbol{x})], -\tau, u^m(\boldsymbol{x})), \quad \boldsymbol{v} = (v_1, v_2). \tag{5.24}$$

Обратим внимание, что для определения величины $u^m(\boldsymbol{X}^m(\boldsymbol{x}))$ фактически используется решение задачи Коши (5.20) назад по времени. Иными словами, по точке на характеристике $\boldsymbol{x} = \boldsymbol{X}(t_{m+1})$ в момент времени t_{m+1} определяется точка на характеристике в момент t_m, т. е. точка $\boldsymbol{X}(t_m)$.

Запишем для уравнения (5.1) аппроксимацию по времени, используя явную схему (см. формулы (4.13)–(4.15) при $\theta = 0$)

$$\frac{u^{m+1}(\boldsymbol{x}) - u^m(\boldsymbol{x})}{\tau} + \boldsymbol{v}^m(\boldsymbol{x}) \cdot \nabla u^m(\boldsymbol{x}) = f^m(\boldsymbol{x}). \tag{5.25}$$

Легко проверить, что с учетом (5.23) это выражение записывается в форме

$$\frac{u^{m+1}(\boldsymbol{x}) - u^m(\boldsymbol{X}^m(\boldsymbol{x}))}{\tau} = f^m(\boldsymbol{x}). \tag{5.26}$$

Таким образом, для расчета функции u в точке $t = t_{m+1}$ достаточно воспользоваться функцией `convect` (см. (5.24))

$$u^{m+1}(\boldsymbol{x}) = \texttt{convect}([v_1^m(\boldsymbol{x}), v_2^m(\boldsymbol{x})], -\tau, u^m(\boldsymbol{x})) + \tau f^m(\boldsymbol{x}). \tag{5.27}$$

5.5 Реализация алгоритма на языке FreeFem++

Пример 5.3 (**Перенос примеси в заданном поле скоростей**). Пусть дана задача Коши в круге $\overline{D}$: $x^2+y^2 \leqslant 1$, описывающая перенос примеси с концентрацией $u(x,y,t)$

$$u_t + \boldsymbol{v}\cdot\nabla u = 0, \quad u\big|_{t=0} = g(x,y). \tag{5.28}$$

Функцию $g(x,y)$ и поле скоростей $\boldsymbol{v}$ зададим соотношениями

$$g(x,y) = 0{,}5\Big(1+\operatorname{th}\big(-\beta((x-x_0)^2+(y-y_0)^2-r_0^2)\big)\Big),$$

$$\boldsymbol{v} = (v_1, v_2), \quad v_1 = \omega y, \quad v_2 = -\omega x.$$

Это означает, что начальное распределение концентрации примеси $g(x,y)$ задано в виде кругового «пятна» радиуса r_0 с центром (x_0,y_0). В начальный момент времени концентрация примеси в пятне будет $g \approx 1$ и вне пятна стремится к нулю (скорость стремления к нулю определяется параметром β).

Для определенности зададим следующие параметры

$$r_0 = 0{,}1, \quad x_0 = 0{,}25, \quad y_0 = 0{,}50, \quad \beta = 10.$$

Код на языке FreeFem++ имеет вид

```
int n=4;
real t, dt;
real x0, y0, r0;
// задаем границы области (окружность единичного радиуса)
border C(t=0,2*pi){ x=cos(t); y=sin(t); };
// строим сетку, на границе 30*n-узлов
mesh Th = buildmesh(C(30*n));
 // задаем пространство конечных элементов
fespace Vh(Th,P2);
// на Vh задаем искомую функцию u, пробную функцию v и вспом. uOld
// используем обозначения: u=u(x,y,(m+1)*dt),  uOld=u(x,y,m*dt)
Vh u, v1, v2, uOld;
// определяем функцию -- начальное распределение примеси
// (x0,y0) -- координаты центра пятна примеси; r0 -- радиус пятна
func g = 0.5*(1+tanh( -10*((x-x0)^2 + (y-y0)^2 - r0^2)));
// задаем поле скоростей
real omega=2*pi;
v1 = omega*y;       v2 = -omega*x;
x0 = 0.25;    y0 = 0.5;    r0 = 0.1;
t = 0;        dt = 0.01;
uOld = g;
// для изображения линий уровня 0.05,0.1,0.2,0.3,....
real [int] viso=[0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1];
// организуем пошаговое решение задачи
for (int m=0; m<=1000; m++)
  {  t = t + dt;
     u = convect([v1,v2],-dt,uOld);
     uOld = u;
     real massa=int2d(Th)(u); // расчет массы
     cout << massa << endl;
     plot(u,  cmm=" t=" + t + "  massa=  " + massa, viso=viso);
  }
```

Заметим, что данную задачу можно также решать, используя слабую формулировку. Это потребуется, в частности, в п. 5.7 для учета эффектов диффузии. Опуская стандартные выкладки, укажем лишь изменения, которые следует сделать в коде программы.

Необходимо описать тестовую функцию `vv`

```
Vh vv;
```

записать коды, соответствующие слабой формулировке задачи (напомним, что нельзя объединять интегралы `int2d`, т. к. как один из них содержит билинейную форму, а другой — линейную)

```
problem Transport(u,vv) =
            int2d(Th)(vv*u) - int2d(Th)(vv * convect([v1,v2],-dt,uOld))
          + on(GammaB, GammaL, GammaR, GammaT, u=0) ;
```

и заменить строку

```
u = convect([v1,v2],-dt,uOld);
```

строкой

```
Transport;
```

5.6 Вычислительный эксперимент

Результаты расчетов по приведенной программе даны на рис. 5.4, на котором изображены линии уровня концентрации $u(x, y, t)$ в моменты времени $t = 0$; 0,1; 0,2; 3,0.

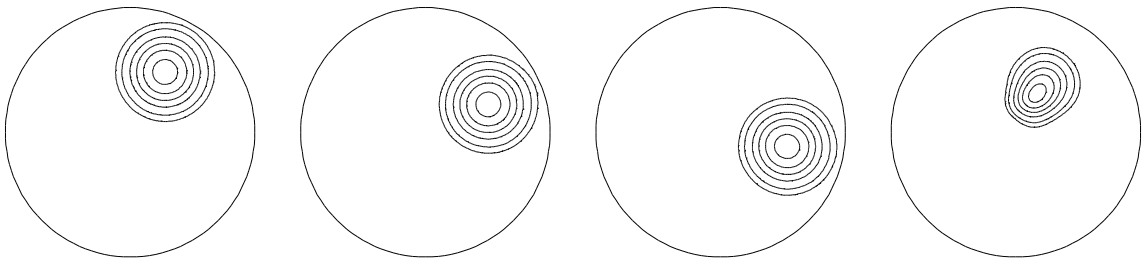

Рис. 5.4. Линии уровня концентрации в моменты времени $t = 0$; 0,1; 0,2; 3,0

Дополнительную информацию о процессе переноса можно получить, реализовав вычисление массы примеси в области D (см. строку 29)

$$M(t) = \iint\limits_D u(x, y, t)\, dx\, dy.$$

В соответствии с примером 5.2, для поля скоростей $\boldsymbol{v} = (\omega y, -\omega x)$ характеристики представляют собой окружности и на каждой характеристике сохраняется начальное значение концентрации. Это, в частности, означает, что «пятно» примеси должно вращаться по часовой стрелке с угловой скоростью ω ***без искажения формы пятна***.

Расчеты показали, что примесь движется без искажения формы лишь на достаточно малом интервале времени (примерно, $t < 0{,}5$). На рис. 5.4 при $t = 3{,}0$ хорошо видно, что форма «пятна» сильно отличается от первоначальной. Наблюдается также уменьшение массы примеси

$$M(0) = 0{,}124489, \quad M(0{,}2) = 0{,}121216, \quad M(3{,}0) = 0{,}074882.$$

На самом деле масса должна сохраняться, так как поле скоростей направлено по касательной к границе и перенос примеси через границу отсутствует. Дефекты расчета (искажение формы и исчезновение массы), конечно же, связаны с вычислительной погрешностью и, в частности, с выбором большого шага движения по времени `dt=0.01` (см. строку 20).

5.7 Уравнение диффузии–переноса

Аналогичным образом, с использованием конструкции `convect`, можно решать, например, задачу о переносе примеси в известном поле скоростей $\boldsymbol{v}(x, y)$ с учетом процессов диффузии.

В случае области D, на границе которой концентрация примеси $u(x, y, t)$ задана равной нулю, указанная задача имеет вид

$$u_t + \boldsymbol{v} \cdot \nabla u - \mu \Delta u = f, \quad u\big|_\Gamma = 0, \quad u\big|_{t=0} = h(x, y), \tag{5.29}$$

где μ — коэффициент диффузии, $h(x, y)$ — начальное распределение концентрации, $f(x, y, t)$ — плотность внутренних источников примеси, возникающих, например, в результате химических реакций.

Укажем, как можно изменить код программы для решения нестационарного уравнения Лапласа (см. с. 62), включив в рассмотрение процесс переноса примеси в заданном поле скорости.

Аппроксимация по времени уравнения (5.29) неявной схемой имеет вид (ср. с формулами (4.6), (4.7))

$$\frac{u^{m+1}(\boldsymbol{x}) - u^m(\boldsymbol{X}^m(\boldsymbol{x}))}{\tau} - \mu \Delta u^{m+1}(\boldsymbol{x}) = f^{m+1}(\boldsymbol{x}), \quad u^{m+1}\big|_\Gamma = 0. \tag{5.30}$$

Соответствующий код слабой формулировки задачи будет

```
problem DifConv(u,vv) =
              int2d(Th)( u*vv + mu*dt*(dx(u)*dx(vv)+dy(u)*dy(vv)) )
            - int2d(Th)( vv*convect([v1,v2],-dt,uOld) )
            + on(GammaB, GammaL, GammaR, GammaT, u=0);
```

Этим фрагментом кода следует заменить строки 27–31 кода на с. 62 (дополнительно можно удалить неиспользуемые строки 24–25, т. к. на границе задано условие $u|_\Gamma = 0$) и определить поле скорости

```
Vh v1, v2;
real omega = 2*pi;
v1 = omega*y;    v2 = -omega*x;
```

После проведенных изменений код на с. 62 можно использовать для решения задачи (5.29).

Глава 6

Уравнения реакция–диффузия. Окраска шкур животных

Все рассмотренные выше примеры относились к случаю, когда изучаемый процесс описывается при помощи одной функции. На практике часто встречаются задачи, в которых для описания процесса требуется несколько неизвестных функций, например, многокомпонентные смеси (неизвестные функции — концентрация $c_k(x, y, t)$, $k = 1, 2, \dots$); уравнения Навье–Стокса (неизвестные функции — компоненты скорости и давление); уравнения тепловой гравитационной конвекции (неизвестные функции — компоненты скорости, давление и температура) и т. п.

Схема построения алгоритма на языке FreeFem++ для решения задач с многими неизвестными функциями мало отличается от случая одной неизвестной функции. Основная особенность, на которую следует обратить внимание, хотя она и достаточно очевидна — это запись слабой (вариационной) формулировки задачи в виде *одного* соотношения для всех неизвестных функций, а не для каждой функции в отдельности.

Проще всего продемонстрировать алгоритм построения решения на примере нестационарных уравнений реакции–диффузии.

6.1 Постановка задачи

Пусть дана система уравнений для определения двух неизвестных функций $s(x, y, t)$ и $a(x, y, t)$

$$\begin{aligned} & s_t - \mu_s \Delta s = g(s, a, x, y, t), \\ & a_t - \mu_a \Delta a = f(s, a, x, y, t), \quad (x, y) \in D, \end{aligned} \tag{6.1}$$

где μ_s, μ_a — параметры; $g(s, a, x, y, t)$, $f(s, a, x, y, t)$ — заданные функции.

Такую систему можно использовать, например, для описания поведения двух веществ (примесей) в растворе с учетом эффектов диффузии и при наличии химических реакций между примесями. В этом случае систему (6.1) принято называть ***уравнениями реакции-диффузии***. Величины, входящие в (6.1), имеют следующий физический смысл: $s(x, y, t)$ и $a(x, y, t)$ — концентрации примесей, μ_s, μ_a — коэффициенты диффузии, $g(s, a, x, y, t)$, $f(s, a, x, y, t)$ — плотности внутренних источников примесей.

Систему уравнений (6.1) дополним краевыми условиями, для простоты выбрав условия Неймана на границе области D

$$\left.\frac{\partial s}{\partial n}\right|_{\Gamma} = 0; \quad \left.\frac{\partial a}{\partial n}\right|_{\Gamma} = 0. \tag{6.2}$$

Эти краевые условия соответствуют случаю непроницаемой для примесей границы Γ — нормальные компоненты плотностей потоков концентрации $\boldsymbol{i}_a \cdot \boldsymbol{n} = -\mu_a \boldsymbol{n} \cdot \nabla a$, $\boldsymbol{i}_s \cdot \boldsymbol{n} = -\mu_s \boldsymbol{n} \cdot \nabla s$ равны нулю.

Кроме этого, зададим начальное распределение концентраций примесей

$$s\big|_{t=0} = H_s(x,y), \qquad a\big|_{t=0} = H_a(x,y), \quad (x,y) \in D, \tag{6.3}$$

где $H_s(x,y)$, $H_a(x,y)$ — известные функции.

Для определенности используем следующие выражения для плотностей источников концентраций

$$g(s,a) = \gamma\left\{s_0 - s - \rho F(s,a)\right\}, \quad f(s,a) = \gamma\left\{\alpha(a_0 - a) - \rho F(s,a)\right\},$$
$$F(s,a) = \frac{sa}{1 + s + Ks^2}. \tag{6.4}$$

Все величины, входящие в эти соотношения, за исключением, естественно, s и a, считаются постоянными параметрами. В данном примере функции g и f не зависят от x, y, t и эти аргументы опущены.

Формулы (6.4) соответствуют задаче об окраске шкур животных, подробно описанной, например, в книге [19]. В п. 6.4 эта задача будет детально исследована средствами FreeFem++.

6.2 Слабая формулировка задачи

Для аппроксимации задачи (6.1)–(6.4) по времени используем так называемую явно-неявную схему (ср. с (4.6), (4.7))

$$\frac{s^{m+1} - s^m}{\tau} - \mu_s \Delta s^{m+1} = \gamma\left\{s_0 - s^{m+1} - \rho F(s^m, a^m)\right\}, \tag{6.5}$$

$$\frac{a^{m+1} - a^m}{\tau} - \mu_a \Delta a^{m+1} = \gamma\left\{\alpha(a_0 - a^{m+1}) - \rho F(s^m, a^m)\right\}, \tag{6.6}$$

$$\left.\frac{\partial s^{m+1}}{\partial n}\right|_{\Gamma} = 0, \quad \left.\frac{\partial a^{m+1}}{\partial n}\right|_{\Gamma} = 0, \quad m = 0, 1, \ldots, \tag{6.7}$$

где

$$s^m = s(x,y,t_m), \quad a^m = a(x,y,t_m), \quad t_m = m\tau. \tag{6.8}$$

Обратим внимание, что исходная задача (6.1)–(6.4) является *нелинейной* задачей относительно неизвестных s, a. Полученная же задача (6.5)–(6.8) относительно неизвестных s^{m+1}, a^{m+1} является *линейной*. Этого удалось достичь за счет использованного явно-неявного способа дискретизации. Производные по времени аппроксимируются обычными конечными

разностями, все остальные *линейные члены* уравнений и краевые условия выбираются в моменты времени $t = t_{m+1}$ (неявная схема), а *нелинейные члены* $F(s,a)$ выбираются в момент времени $t = t_m$ (явная схема). Впрочем, имеются и иные варианты аппроксимации для функции $F(s,a)$. Например, числитель этой функции, т. е. sa, можно представлять в виде $s^{m+1}a^m$ для уравнения (6.5) и в виде $s^m a^{m+1}$ для уравнения (6.6).

Слабую формулировку задачи (6.5)–(6.8) получим, умножая уравнения (6.5), (6.6) на *независимые* тестовые функции $v_s(x,y)$, $v_a(x,y)$, а затем складывая уравнения и интегрируя по частям при помощи формулы Грина (с учетом краевых условий (6.7))

$$\begin{aligned} &\iint\limits_D \left(\frac{s^{m+1}-s^m}{\tau} v_s + \mu_s \nabla s \cdot \nabla v_s \right) dx\,dy - \\ - &\iint\limits_D \gamma(s_0 - s^{m+1} - \rho F(s^m, a^m)) v_s \, dx\,dy + \\ + &\iint\limits_D \left(\frac{a^{m+1}-a^m}{\tau} v_a + \mu_a \nabla a \cdot \nabla v_a \right) dx\,dy - \\ - &\iint\limits_D \gamma(\alpha(a_0 - a^{m+1}) - \rho F(s^m, a^m)) v_a \, dx\,dy = 0. \end{aligned} \tag{6.9}$$

6.3 Фрагменты кодов на языке FreeFem++

Полный код на языке FreeFem++ для решения задачи (6.5)–(6.8) дан в п. 6.4.2. Приведем фрагменты программы, позволяющие использовать результаты, полученные ранее при решении нестационарной задачи для уравнения Лапласа (см. с. 62).

Здесь и далее используем следующие идентификаторы

s^{m+1}	→	`s`	$g(s,a)$	→	`gsa`	α	→	`alpha`
a^{m+1}	→	`a`	$f(s,a)$	→	`fsa`	ρ	→	`rho`
s^m	→	`spr`	μ_s	→	`mus`	β	→	`beta`
a^m	→	`apr`	μ_a	→	`mua`	τ	→	`dt`
v_s	→	`vs`	K	→	`K`	t	→	`t`
v_a	→	`va`	γ	→	`gamma`			

При составлении алгоритма следует учесть, что необходимо ввести *две* неизвестные функции `s` и `a` и для *каждой из них* свои тестовые функции, например, `vs`, `va`

```
Vh s, a, vs, va;
```

Слабая формулировка задачи (6.9) записывается в виде

```
problem Diffusion(s, a, vs, va, solver=UMFPACK) =
             int2d(Th)( s*vs + dt*mus*(dx(s)*dx(vs) + dy(s)*dy(vs)) )
            -int2d(Th)( ( spr + dt*gsa )*vs )
```

```
            +int2d(Th)( a*va + dt*mua*(dx(a)*dx(va) + dy(a)*dy(va)) )
            -int2d(Th)( ( apr + dt*fsa )*va ) ;
```

Сравнивая `problem Diffusion(s, a, vs, va)` с `problem Heat(u, v)` на с. 62, видим, что в данном случае в качестве аргументов `Diffusion` следует использовать `s`, `a`, `vs`, `va`, т. е. перечислять все неизвестные и тестовые функции. Алгоритм решения нестационарной задачи имеет вид (ср. с аналогичным для `problem Heat` на с. 62)

```
for (int m=0; m<=1000; m++)
  {  gsa = gamma*(s0-spr - rho*spr*apr/(1+spr+K*spr^2));
     fsa = gamma*(alpha*(a0-apr) - rho*spr*apr/(1+spr+K*spr^2));
     t = t + dt;
     Diffusion;
     spr = s;    apr = a;
     plot(s);
   }
```

Обратим внимание, что функции `gsa`, `fsa` вычисляются внутри цикла на каждом шаге по времени и должны быть предварительно описаны до их появления в строках кода, т. е. до записи `problem Diffusion`

```
Vh gsa, fsa;
```

Наконец, укажем как на языке FreeFem++ в случае задачи с несколькими неизвестными функциями следует задавать краевые условия первого рода. Например, для задания однородных условий Дирихле на каком-либо участке границы с именем `Bn` в `problem Diffusion(...) =` необходимо добавить строку

```
+ on(Bn, s=0, a=0);
```

$$(s|_{\mathrm{Bn}} = 0, \quad a|_{\mathrm{Bn}} = 0).$$

Иными словами, краевые условия должны быть указаны для каждой неизвестной функции.

Учет краевых условий третьего рода и неоднородных условий второго рода, как обычно, осуществляется при записи слабой (вариационной) формулировки задачи, т. е. при помощи интегралов по контуру `int1d` и интегралов по области `int2d` (для краевых условий третьего рода).

6.4 Окраска шкур животных

Как уже говорилось при задании функций (6.4), задача (6.1)–(6.4) может моделировать процесс формирования окраски шкур животных [19]. Эта прекрасная задача как нельзя лучше позволяет продемонстрировать широкие возможности метода конечных элементов для построения решения в областях сложной формы.

Приведем минимально необходимые сведения, требующиеся для понимания результатов дальнейших расчетов. Предположим, что имеется шкура зародыша животного (двумерная область D сложной формы), на которой в процессе развития эмбриона формируется окраска, например, в виде

чередующихся полос, как у зебры, или пятен, как у леопарда. В простейшей модели считается, что на шкуре животного имеется начальное распределение двух веществ, одно из которых субстрат (красящий пигмент), а другое — косубстрат. В результате химических взаимодействий в присутствии фермента (катализатора) и процессов диффузии происходит перераспределение концентрации красящего вещества.

Система уравнений (6.1), (6.4) как раз и описывает двухкомпонентную смесь, в которой происходит превращение тирозина в меланин при наличии ферментов в эпидерме и (или) шерсти. Вещество с концентрацией s является субстратом (меланин), а вещество с концентрацией a — косубстратом (тирозин). Предполагается, что на шкуре имеется равномерное распределение субстрата s_0 и косубстрата a_0. Члены $\gamma(s_0 - s)$ и $\gamma\alpha(a_0 - a)$ в плотностях источников концентраций (функции g и f) моделируют приток веществ к поверхности шкуры из тела эмбриона.

Функция $F(s, a)$ (см. (6.4)) моделирует реакцию, которую называют реакцией с ингибированием субстратом (K — коэффициент ингибирования). Термин «ингибирование субстратом» означает, что с ростом концентрации s скорость химической реакции $F(s, a)$ стремится к нулю — большое количество субстрата подавляет (ингибирует) процесс протекания реакций.

Процесс возникновения сложной окраски качественно можно объяснить следующим образом. Если в результате химических реакций между компонентами s и a в окрестности какой-либо точки (x, y) области D количество субстрата s становится достаточно большим, то протекание химической реакции полностью подавляется. В такой ситуации решающую роль в распределении концентрации играют диффузионные процессы. В результате диффузии концентрация субстрата s в точке (x, y) начинает уменьшаться и вновь «включаются» химические процессы, приводящие к увеличению концентрации субстрата. При некоторых значениях параметров могут возникнуть сложные пространственно-временные структуры, соответствующие полосам, пятнам и другим рисункам окраски шкуры животного.

Описаный механизм возможного образования структур называется диффузионной неустойчивостью и особенно ярко проявляется в окрестности точки равновесия $(\tilde{s}, \tilde{a})$, которая является решением системы уравнений $f(s, a) = 0$, $g(s, a) = 0$ (и всей задачи в целом).

6.4.1 Вычислительный эксперимент

На рис. 6.1 показано образование пространственной структуры распределения субстрата в области D, которая имитирует шкуру домашней длинношвостой кошки[1]. Для проведения расчетов использован один из наборов параметров, предложенный в [19] (к сожалению, размеры области D там не указаны)

$$K = 0{,}125; \quad \rho = 13; \quad s_0 = 103; \quad a_0 = 77;$$

$$\alpha = 1{,}5; \quad \mu_a = 1; \quad \mu_s = \beta = 7; \quad \gamma = 510.$$

[1] Прототипом области D послужила шкура кошки учебной компьютерной лаборатории кафедры вычислительной математики и математической физики `htpp://vmmf.math.rsu.ru`

Равновесному решению соответствуют следующие значения

$$s = \tilde{s} = 24{,}95939605, \quad a = \tilde{a} = 24{,}97293070$$

(заметим, что в [19] приведены ошибочные значения $\tilde{s} = 24$ и $\tilde{a} = 23$).

В качестве начального распределения (6.3) взято малое возмущение стационарного решения (имитация случайного возмущения)

$$s = \tilde{s} + 10^{-2}\cos(12{,}765\pi x)\sin(12\pi y),$$

$$a = \tilde{a} + 10^{-5}\cos(10\pi x)\sin(16{,}786\pi y).$$

На рис. 6.1 хорошо видно, как с течением времени формируется пространственно-периодическая структура изолиний распределения концентрации s. На самом деле, на рис. 6.1 для лучшей визуализации приведена разность $s(x, y, t) - \tilde{s}$. Дело в том, что в задаче очень быстро устанавливается решение, близкое к стационарному, т. е. диффузионная неустойчивость возникает в окрестности точки $(\tilde{s}, \tilde{a})$.

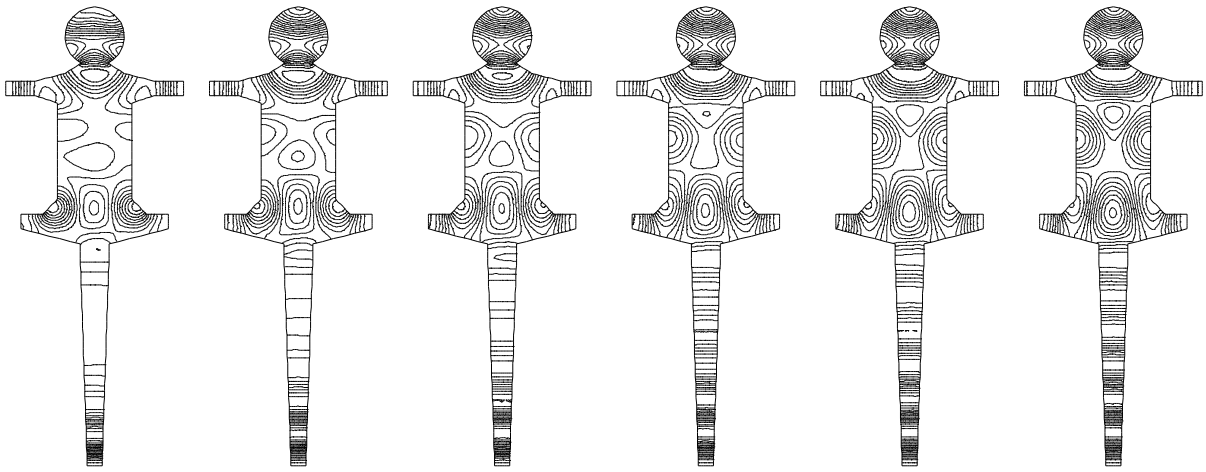

Рис. 6.1. Изолинии функции $s(x, y, t) - \tilde{s}$ при $t = 0{,}025m$, $m = 1, \ldots, 6$

Рис. 6.1 показывает, что с течением времени формируется ярко выраженная полосатость протяженных областей (хвост, лапы). В областях с соизмеримыми размерами (туловище, голова) полосатость не так заметна.

Подчеркнем, что модель (6.1)–(6.4) реально описывает лишь двухцветный механизм формирования полос или пятен. Значениям $(s - \tilde{s}) > 0$ соответствует черный цвет, а значениям $(s - \tilde{s}) < 0$ — белый. На рис. 6.2 (второй слева) показано распределение черно-белых полос на шкуре. Для реализации такой визуализации был использован следующий код (конечно, необходимо описать массив `rr` — `Vh rr`)

```
for (int j=0; j<sdif[].n; j++)
  {
   if (!(sdif[][j]>0))  { rr[][j]=1; }
   if (!(sdif[][j]<0))  { rr[][j]=0; }
  }
plot(rr, bw=1);
```

6.4.1.1 Деформация области

В языке FreeFem++ имеется ключевое слово `movemesh` для деформации исходной области. Пример использования `movemesh`

```
func uu = sin(y*pi)/10,    vv = cos(x*pi)/10;
// коэффициент деформации сетки
real coef = 1.5;
// деформация сетки
Th = movemesh(Th0,[x+coef*uu,y+coef*vv]);
```

На рис. 6.2 показана триангуляция сложной области, использованной для расчетов, показанных на рис. 6.1, и триангуляция области, полученной в результате деформации.

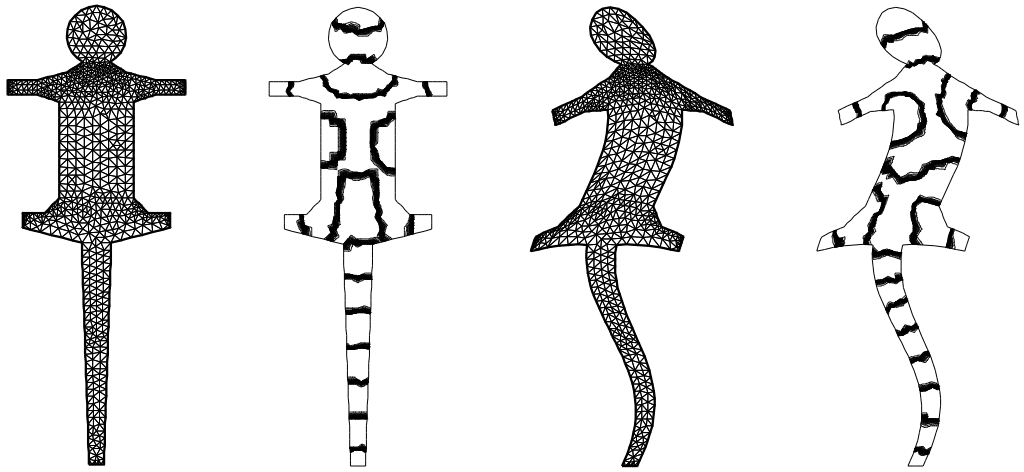

Рис. 6.2. Полосы окраски в момент времени $t = 0{,}1$

6.4.1.2 Различные модификации кода

Подчеркнем, что во всех приводимых программах контроль точности вычислений не производится. Результаты расчетов могут довольно сильно зависеть от количества точек, выбираемых для триангуляции области

```
mesh Th0 = buildmesh(...);
```

от выбора конечных элементов (`P1`, `P2`,...) в операторе

```
fespace Vh(Th, P2);
```

от способа вычисления интегралов `int2d`, `int1d`, от алгоритма решения алгебраических уравнений (`solver`)

```
problem Diffusion(s,a,vs,va, solver=UMFPACK) = ...;
```

Обо всем этом подробно рассказано в гл. 19. Приведем здесь для примера лишь результаты расчета с конечными элементами `P1` и `P2`.

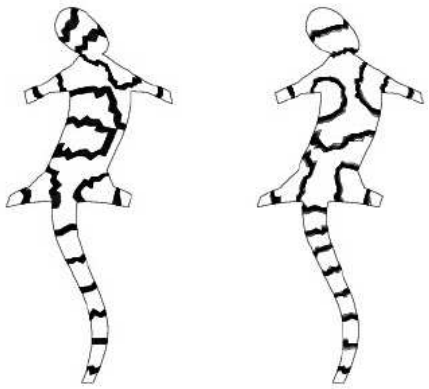

Рис. 6.3. Полосы окраски при $t = 0{,}1$ для конечных элементов `P1` (слева) и `P2`

Как указано в [19], структура полос сильно зависит от параметра γ. При малых γ в области будет формироваться лишь один черный цвет, при увеличении γ возникнет двухцветный окрас (черный верх, белый низ). При дальнейшем увеличении γ будет возникать более частое чередование черного и белого цветов и, наконец, при очень больших γ чередование будет столь велико, что практически останется лишь один черный цвет (кошка повышенной черной пятнистости).

Заметим, что все параметры, входящие в задачу (K, β, α, s_0, a_0,...), могут быть функциями координат и времени. Очевидно, это будет имитировать ситуацию пространственно-неоднородной шкуры и изменения параметров, например, константы ингибирования K с течением времени.

6.4.2 Реализация алгоритма на языке FreeFem++

Приведем полный код программы на языке FreeFem++, которая использовалась для расчета окраски шкуры кошки. Заметим, что большую часть кода (строки 5–65) занимает задание области D (имитирующей форму шкуры), в которой решается задача.

```
1  // описание параметров задачи
2  int n=2;
3  real alpha, beta, gamma, rho, a0, s0, K;
4  real t, dt;
5  // параметры для задания сложной области -- шкуры животного
6  real xx, yy;
7  real w1,w2,w3,w4,w5,w6,w7,w8,w9;
8  real h1,h2,h3,h4,h5,h6,h7,h8,h9,h10;
9  real x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14;
10 real y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14;
11 real x15,x16,x17,x18,x19,x20,x21,x22,x23,x24,x25,x26,x1;
12 real y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y1;
13 real delta,R,y00;
14 // задание координат для построения границ области
15 // голова
16 delta = 1.1;    R = 0.2;    y00 = 1.0;
17 w1=R*cos(-delta);   h1=y00+R*sin(-delta);
```

```
// параметры для координат точек контура границы
//w1=0.15;
w2=0.20; w3=0.40; w4=0.60; w5=0.25; w6=0.35; w7=0.50; w8=0.10; w9=0.05;
//h1=1.0;
h2=0.75;  h3=0.7;   h4=0.6;   h5=0.55;  h6=-0.1;
h7=-0.2;  h8=-0.3;  h9=-0.4;  h10=-1.9;
// координаты точек левого контура
x2=-w1;  y2=h1;   x3=-w2;  y3=h2;   x4=-w3;  y4=h3;   x5=-w4;  y5=h3;
x6=-w4;  y6=h4;   x7=-w3;  y7=h4;   x8=-w5;  y8=h5;   x9=-w5;  y9=h6;
x10=-w6; y10=h7;  x11=-w7; y11=h7;  x12=-w7; y12=h8;  x13=-w8; y13=h9;
x14=-w9; y14=h10;
// координаты точек правого контура
x1=w1;   y1=h1;   x26=w2;  y26=h2;  x25=w3;  y25=h3;  x24=w4;  y24=h3;
x23=w4;  y23=h4;  x22=w3;  y22=h4;  x21=w5;  y21=h5;  x20=w5;  y20=h6;
x19=w6;  y19=h7;  x18=w7;  y18=h7;  x17=w7;  y17=h8;  x16=w8;  y16=h9;
x15=w9;  y15=h10;
// задание линий контура
// окружность для головы
border L1(t=-delta,pi+delta){ x=R*cos(t);         y=y00+R*sin(t); };
// отрезки на границе для левого контура
//border L1(t=0,1){ x=(1-t)*x1+t*x2;      y=(1-t)*y1+t*y2; };
border L2(t=0,1){ x=(1-t)*x2+t*x3;        y=(1-t)*y2+t*y3; };
border L3(t=0,1){ x=(1-t)*x3+t*x4;        y=(1-t)*y3+t*y4; };
border L4(t=0,1){ x=(1-t)*x4+t*x5;        y=(1-t)*y4+t*y5; };
border L5(t=0,1){ x=(1-t)*x5+t*x6;        y=(1-t)*y5+t*y6; };
border L6(t=0,1){ x=(1-t)*x6+t*x7;        y=(1-t)*y6+t*y7; };
border L7(t=0,1){ x=(1-t)*x7+t*x8;        y=(1-t)*y7+t*y8; };
border L8(t=0,1){ x=(1-t)*x8+t*x9;        y=(1-t)*y8+t*y9; };
border L9(t=0,1){ x=(1-t)*x9+t*x10;       y=(1-t)*y9+t*y10; };
border L10(t=0,1){ x=(1-t)*x10+t*x11;     y=(1-t)*y10+t*y11; };
border L11(t=0,1){ x=(1-t)*x11+t*x12;     y=(1-t)*y11+t*y12; };
border L12(t=0,1){ x=(1-t)*x12+t*x13;     y=(1-t)*y12+t*y13; };
border L13(t=0,1){ x=(1-t)*x13+t*x14;     y=(1-t)*y13+t*y14; };
// нижний горизонтальный отрезок
border L14(t=0,1){ x=(1-t)*x14+t*x15;     y=(1-t)*y14+t*y15; };
// отрезки на границе для правого контура
border L15(t=0,1){ x=(1-t)*x15+t*x16;     y=(1-t)*y15+t*y16; };
border L16(t=0,1){ x=(1-t)*x16+t*x17;     y=(1-t)*y16+t*y17; };
border L17(t=0,1){ x=(1-t)*x17+t*x18;     y=(1-t)*y17+t*y18; };
border L18(t=0,1){ x=(1-t)*x18+t*x19;     y=(1-t)*y18+t*y19; };
border L19(t=0,1){ x=(1-t)*x19+t*x20;     y=(1-t)*y19+t*y20; };
border L20(t=0,1){ x=(1-t)*x20+t*x21;     y=(1-t)*y20+t*y21; };
border L21(t=0,1){ x=(1-t)*x21+t*x22;     y=(1-t)*y21+t*y22; };
border L22(t=0,1){ x=(1-t)*x22+t*x23;     y=(1-t)*y22+t*y23; };
border L23(t=0,1){ x=(1-t)*x23+t*x24;     y=(1-t)*y23+t*y24; };
border L24(t=0,1){ x=(1-t)*x24+t*x25;     y=(1-t)*y24+t*y25; };
border L25(t=0,1){ x=(1-t)*x25+t*x26;     y=(1-t)*y25+t*y26; };
border L26(t=0,1){ x=(1-t)*x26+t*x1;      y=(1-t)*y26+t*y1; };
// формирование сетки - на различных участках границы, в зависимости от
// их длины, выбирается различное количество отрезков разбиения
mesh Th0 =
```

```
69      buildmesh(L1(10*n)+L2(3*n)+L3(3*n)+L4(3*n)+L5(3*n)+L6(3*n)+L7(3*n)
70          +L8(6*n)+L9(3*n)+L10(3*n)+L11(3*n)+L12(3*n)+L13(16*n)+L14(3*n)
71          +L15(16*n)+L16(3*n)+L17(3*n)+L18(3*n)+L19(3*n)+L20(6*n)
72          +L21(3*n)+L22(3*n)+L23(3*n)+L24(3*n)+L25(3*n)+L26(3*n));
73  // для деформации
74  mesh Th=Th0;
75  // функции для деформации сетки
76  //func uu = sin(2*y*pi)/10;
77  //func vv = cos(2*x*pi)/10;
78  func uu = sin(y*pi)/10;
79  func vv = cos(x*pi)/10;
80  // коэффициент деформации сетки
81  real coef=1.5;
82  // деформация сетки (убрать комментарии, если понадобится деформация)
83  //Th = movemesh(Th0,[x+coef*uu,y+coef*vv]);
84  // рисование сетки
85  plot(Th, wait=1);
86  // задание пространства конечных элементов
87  fespace Vh(Th, P2);
88  // на  Vh определяем искомые функции s, a и тестовые функции vs, va
89  Vh s, vs, a, va;
90  // на пространстве Vh определяем вспомогательные функции
91  Vh gsa, fsa;
92  Vh spr, apr;
93  // запись слабой (вариационной) формулировки задачи
94  problem Diffusion(s,a,vs,va, solver=UMFPACK) =
95      int2d(Th)(s*vs + dt*(dx(s)*dx(vs) + dy(s)*dy(vs)) )
96     -int2d(Th)((spr + dt*gsa)*vs)
97     +int2d(Th)(a*va + beta*dt*(dx(a)*dx(va) + dy(a)*dy(va)) )
98     -int2d(Th)((apr + dt*fsa)*va)  ;
99  // задание параметров для исходной задачи
100 alpha=1.5;  K=0.125;  rho=13;  s0=103;  a0=77;  beta=7;  gamma=510;
101 // описание переменных для стационарных состояний
102 real sinit, ainit;
103 // задание вспомогательной функции для вывода информации
104 Vh sdif;
105 // стационарное решение -- корни системы уравнений gsa=0, fsa=0
106 sinit = 24.95939605;  ainit = 24.97293070;
107 // возмущения стационарного решения
108 spr = sinit + 0.01*cos(12.765*pi*x)*sin(12*pi*y);
109 apr = ainit + 0.00001*cos(10*pi*x)*sin(16.786*pi*y);
110 // организуем пошаговое решение задачи
111 t = 0;   dt = 0.001;
112 for (int m=0; m<=1000; m++)
113   { // определение функций gsa, fsa
114     gsa = gamma*(s0-spr - rho*spr*apr/(1+spr+K*spr^2));
115     fsa = gamma*(alpha*(a0-apr) - rho*spr*apr/(1+spr+K*spr^2));
116     t = t + dt;
117     // вызов problem Diffusion
118     Diffusion;
119     // перенесение решения с одного временного слоя на другой
```

```
120       spr = s;
121       apr = a;
122       // формирование разности между решением и стационарным состоянием
123       sdif = s - sinit;
124       // вывод результатов
125       plot(sdif, fill=!(m % 5), //  заливка через пять шагов
126            wait=!(m % 25),      // ожидание "щелчка мыши" для
127                                 // продолжения расчета через 25 шагов
128            value=!(m % 25),     // вывод легенды через 25 шагов
129            cmm=m);
130       cout << 10000*s[].min << " " << 10000*s[].max << endl;
131     }
```

✓. *В языке FreeFem++ имеются встроенные генераторы случайных чисел (см. [1]), которые, в принципе, можно использовать для задания возмущения стационарного решения, изменив соответствующим образом строки 108, 109.*

Глава 7

Перенос-диффузия вихря

Уравнения движения вязкой несжимаемой жидкости в двумерном случае, записанные в так называемых переменных вихрь-функция тока, формально совпадают с уравнениями переноса примеси при наличии диффузии. Это, в частности, позволяет использовать результаты гл. 5 для конструирования на языке FreeFem++ достаточно простого алгоритма решения задачи о движении жидкости.

7.1 Переменные вихрь-функция тока

Уравнения Навье–Стокса для описания поведения вязкой несжимаемой жидкости имеют вид (см., например, [16])

$$\begin{aligned} &\boldsymbol{v}_t + \boldsymbol{v}\cdot\nabla\boldsymbol{v} = -\nabla p + \mu\Delta\boldsymbol{v}, \\ &\operatorname{div}\boldsymbol{v} = 0. \end{aligned} \tag{7.1}$$

Здесь $\boldsymbol{v}$ — скорость, p — давление, μ — коэффициент кинематической вязкости жидкости.

В двумерном случае ($\boldsymbol{v} = (u, w)$) покоординатная запись уравнений (7.1) будет следующей

$$\begin{aligned} &u_t + uu_x + wu_y = -p_x + \mu(u_{xx} + u_{yy}), \\ &w_t + uw_x + ww_y = -p_y + \mu(w_{xx} + w_{yy}), \\ &u_y + w_x = 0. \end{aligned} \tag{7.2}$$

Введем функцию тока ψ при помощи соотношений (при этом уравнение $\operatorname{div}\boldsymbol{v} = 0$ удовлетворяется автоматически)

$$u = \psi_y, \quad w = -\psi_x. \tag{7.3}$$

Введем обозначение для вихря скорости ω

$$\omega = w_x - u_y. \tag{7.4}$$

На самом деле, ω — это z-компонента вектора $\boldsymbol{\omega} = \operatorname{rot}\boldsymbol{v}$, т. е. $\omega = \boldsymbol{k}\cdot\boldsymbol{\omega}$, где $\boldsymbol{k}$ — единичный вектор, перпендикулярный плоскости (x, y).

Дифференцируя первое уравнение (7.2) по y, второе — по x и вычитая друг из друга, с учетом обозначений (7.4) выводим (эти действия позволяют исключить давление p из уравнений (7.2))

$$\omega_t + u\omega_x + w\omega_y = \mu(\omega_{xx} + \omega_{yy}). \tag{7.5}$$

При помощи (7.3) с учетом (7.4) нетрудно получить уравнения, связывающие вихрь и функцию тока

$$\psi_{xx} + \psi_{yy} = -\omega. \tag{7.6}$$

Окончательно, для определения ω, ψ имеем так называемые уравнения в переменных вихрь–функция тока

$$\omega_t + u\omega_x + w\omega_y = \mu(\omega_{xx} + \omega_{yy}), \tag{7.7}$$

$$\psi_{xx} + \psi_{yy} = -\omega, \tag{7.8}$$

$$u = \psi_y, \quad w = -\psi_x. \tag{7.9}$$

Векторная форма записи уравнений (7.7)–(7.9) имеет вид

$$\omega_t + \boldsymbol{v} \cdot \nabla\omega = \mu\Delta\omega, \tag{7.10}$$

$$\Delta\psi = -\omega, \tag{7.11}$$

$$\boldsymbol{v} = (\psi_y, -\psi_x). \tag{7.12}$$

Конечно, можно подставить (7.12) в (7.10) и, используя (7.11), записать уравнения лишь для функции тока

$$(\Delta\psi)_t + \psi_y(\Delta\psi)_x - \psi_x(\Delta\psi)_y = \mu\Delta^2\psi.$$

Для дальнейших целей удобна именно система (7.7)–(7.9) или (7.10)–(7.12) в переменных вихрь-функция тока.

Формально, уравнение (7.10) совпадает с уравнением (5.29), описывающим процесс переноса примеси в поле скорости $\boldsymbol{v}$ при наличии диффузии. Роль примеси в данном случае играет вихрь ω. Отметим, что поле скорости в гл. 5 считалось известной функцией $\boldsymbol{v} = \boldsymbol{v}(x, y, t)$, а в случае системы (7.10)–(7.12) поле скорости $\boldsymbol{v} = (\psi_y, -\psi_x)$ зависит от ω и определяется уравнением Пуассона (7.11). Иными словами, вихрь в процессе эволюции «перестраивает» поле скорости, в котором осуществляется его перенос.

7.2 Постановка задачи

Рассмотрим задачу о поведении вязкой несжимаемой жидкости в ограниченной двумерной области D с непроницаемыми границами. Предполагаем, что в начальный момент времени в области D задано распределение вихря скорости $\omega_0(x, y)$

$$\omega_t + \boldsymbol{v} \cdot \nabla\omega = \mu\Delta\omega, \quad \boldsymbol{v} = (\psi_y, -\psi_x), \quad (x, y) \in D, \tag{7.13}$$

$$\omega\big|_\Gamma = 0, \quad \Gamma = \partial D, \tag{7.14}$$

$$\omega\big|_{t=0} = \omega_0(x,y), \quad (x,y) \in D. \tag{7.15}$$

$$\Delta\psi = -\omega, \quad (x,y) \in D, \tag{7.16}$$

$$\psi\big|_\Gamma = 0, \quad \Gamma = \partial D. \tag{7.17}$$

Объясним, почему краевое условие для функции тока ψ имеет вид (7.17). В случае вязкой жидкости на непроницаемой границе для скорости $\boldsymbol{v}$ имеем

$$u\big|_\Gamma = 0, \qquad w\big|_\Gamma = 0$$

или

$$\psi_y\big|_\Gamma = 0, \qquad \psi_x\big|_\Gamma = 0.$$

Пусть Γ достаточно гладкая граница: $\Gamma = \{(x,y) : x = x(s), y = y(s)\}$, где s — параметр. Вычисляя производную функции ψ вдоль границы, получим

$$\psi_s\big|_\Gamma = (\psi_x x'(s) + \psi_y y'(s))\big|_\Gamma = \psi_x\big|_\Gamma x'(s) + \psi_y\big|_\Gamma y'(s) = 0.$$

Это означает, что $\psi\big|_\Gamma = \text{const}$. С учетом того, что функция ψ определена с точностью до постоянной, выберем $\psi\big|_\Gamma = 0$.

7.3 Алгоритм решения задачи

Для решения задачи (7.13)–(7.17) можно применять результаты гл. 5, т. к. левая часть уравнения (5.1) идентична левой части уравнения (7.13).

Будем использовать следующий алгоритм дискретизации задачи по времени. Обозначим (см. (5.22))

$$\omega^m(\boldsymbol{x}) = \omega(\boldsymbol{x}, t_m). \tag{7.18}$$

Для вычисления величины $\omega^m(\boldsymbol{X}^m(\boldsymbol{x}))$ возьмем формулы, аналогичные (5.23), (5.24)

$$\omega^m(\boldsymbol{X}^m(\boldsymbol{x})) = \omega^m(\boldsymbol{x}) - \tau \boldsymbol{v}^m(\boldsymbol{x}) \cdot \nabla\omega^m(\boldsymbol{x}). \tag{7.19}$$

$$\omega^m(\boldsymbol{X}^m(\boldsymbol{x})) = \texttt{convect}([u^m(\boldsymbol{x}), w^m(\boldsymbol{x})], -\tau, \omega^m(\boldsymbol{x})), \quad \boldsymbol{v} = (u,w). \tag{7.20}$$

Уравнение (7.10) аппроксимируем выражением (см. (5.25), (5.26), (5.30))

$$\frac{\omega^{m+1}(\boldsymbol{x}) - \omega^m(\boldsymbol{x})}{\tau} + \boldsymbol{v}^m(\boldsymbol{x}) \cdot \nabla\omega^m(\boldsymbol{x}) - \mu\Delta\omega^{m+1}(\boldsymbol{x}) = 0. \tag{7.21}$$

Заметим, что, как и в гл. 6 (см. (6.5)–(6.8)), для аппроксимации использована явно-неявная схема — линейные члены, т. е. $\Delta\omega$, выбираются в момент времени $t = t_{m+1}$, а нелинейные, т. е. $\boldsymbol{v} \cdot \nabla\omega$, — в момент времени $t = t_m$.

При помощи (7.19) получим

$$\frac{\omega^{m+1}(\boldsymbol{x}) - \omega^m(\boldsymbol{X}^m(\boldsymbol{x}))}{\tau} - \mu\Delta\omega^{m+1}(\boldsymbol{x}) = 0. \tag{7.22}$$

С учетом обозначения (7.20) запишем (7.22) в виде

$$\omega^{m+1}(\boldsymbol{x}) - \texttt{convect}([u^m(\boldsymbol{x}), w^m(\boldsymbol{x})], -\tau, \omega^m(\boldsymbol{x})) - \tau\mu\Delta\omega^{m+1}(\boldsymbol{x}) = 0. \quad (7.23)$$

Таким образом, величина ω^{m+1} рассчитывается по значениям величин ω^m, u^m, w^m. Очевидно, что если величина ω^m известна, то u^m, w^m определяются решением стационарной задачи (7.16), (7.17)

$$\Delta\psi^m = -\omega^m, \quad \psi^m\big|_\Gamma = 0. \quad (7.24)$$

По известной ψ^m с учетом формулы (7.12) вычисляем u^m, w^m. В языке FreeFem++ для вычисления производных по x и y используются ключевые слова dx, dy

$$u^m = \texttt{dy}(\psi^m); \quad w^m = -\texttt{dx}(\psi^m). \quad (7.25)$$

✓. *Обратим внимание, что использование явно-неявной дискретизации по времени позволило «расщепить» исходную задачу (7.13)–(7.17) на две задачи — задачу (7.16), (7.17) для определения скорости $\boldsymbol{v}^m$ и задачу (7.13)–(7.15) для определения ω^{m+1}.*

7.4 Реализация алгоритма на языке FreeFem++

Приведем полный текст программы на языке FreeFem++ для решения задачи (7.13)–(7.17) в случае прямоугольной области $\overline{D} = [0, a] \times [0, b]$. Для определенности выбираем начальное распределение вихря скорости $\omega_0(x, y)$ в виде

$$\omega_0(x, y) = \sum_{k=1}^{n} A_k \Big\{ 1 + \operatorname{th}\big(-\beta((x - x_k)^2 + (y - y_k)^2 - r_k^2)\big) \Big\}. \quad (7.26)$$

При больших значениях параметра $\beta > 0$ это соответствует вихревым «пятнам» $D_k = \big\{(x, y)\colon (x - x_k)^2 + (y - y_k)^2 \leqslant r_k^2\big\}$, которые сосредоточены в кругах с радиусами r_k и центрами в точках с координатами (x_k, y_k). Для каждого «пятна» при $\beta \gg 1$ имеем

$$\Delta\psi \approx -2A_k, \quad (x, y) \in D_k, \quad \psi\big|_{D \setminus \bigcup\limits_k D_k} \approx 0,$$

$$\psi \approx -A_k\big((x - x_k)^2 + (y - y_k)^2 - r_k^2\big),$$

$$u = \psi_y \approx -2A_k(y - y_k), \quad w = -\psi_x \approx 2A_k(x - x_k).$$

Это означает, что при $A_k > 0$ жидкость в «пятне» D_k вращается как твердое тело вокруг точки (x_k, y_k) против часовой стрелки (при $A_k < 0$ — по часовой стрелке) с угловой скоростью $2A_k$.

```
1 real a=1.0, b=1.0;
2 int n=6;
3 real t, dt, mu;
```

```
// задаем границы области (прямоугольник [0,a]x[0,b])
border GammaB(t=0,1){ x=a*t; y=0; };
border GammaR(t=0,1){ x=a; y=b*t; };
border GammaT(t=0,1){ x=a*(1-t); y=b; };
border GammaL(t=0,1){ x=0; y=b*(1-t); };
mesh Th = buildmesh(GammaB(5*n)+GammaR(5*n)+GammaT(5*n)+GammaL(5*n));
fespace Vh(Th,P2);  // задаем пространство конечных элементов
// на Vh задаем искомые функции omega, psi, u, w,
// тестовые функции v, vpsi  и вспомогательную omegaOld
Vh omega, psi, u, w, v, vpsi, omegaOld;
// определяем функцию -- начальное распределение вихря
real Ampl1=24, Ampl2=-24, Ampl3=24, Ampl4=-24;
real x1=0.2, x2=0.8, y1=0.25, y2=0.25, x3=0.2, x4=0.8, y3=0.75, y4=0.75;
real beta=50, r1=0.05, r2=0.05, r3=0.05, r4=0.05;
func omega0 = Ampl1*(1+tanh( -beta*((x-x1)^2 + (y-y1)^2 - r1^2) ))
            + Ampl2*(1+tanh( -beta*((x-x2)^2 + (y-y2)^2 - r2^2) ))
            + Ampl3*(1+tanh( -beta*((x-x3)^2 + (y-y3)^2 - r3^2) ))
            + Ampl4*(1+tanh( -beta*((x-x4)^2 + (y-y4)^2 - r4^2) ));
mu = 0.001;
t = 0;   dt = 0.01;
omegaOld = omega0;
problem Curl(omega, v) = // задача (7.13)-(7.14)
     int2d(Th)(omega*v)
   - int2d(Th)(v * convect([u,w],-dt,omegaOld))
   + int2d(Th)( dt * mu * (dx(omega) * dx(v) + dy(omega) * dy(v)))
   + on(GammaB,GammaL,GammaR,GammaT, omega=0) ;
problem Poisson(psi,vpsi) =  //  задача (7.16)-(7.17)
     int2d(Th)( dx(psi)*dx(vpsi)+dy(psi)*dy(vpsi) )
   - int2d(Th)( omegaOld * vpsi )
   + on(GammaB, GammaL, GammaR, GammaT, psi=0);
// организуем пошаговое решение задачи
for (int m=0; m<=1000; m++)
  {  t = t + dt;
     Poisson;
     u = dy(psi);     w = -dx(psi);
     omega = dx(w) - dy(u);
     Curl;
     omegaOld = omega;
     plot(omega, cmm=" t="+t);
   }
```

7.5 Вычислительный эксперимент

7.5.1 Движение вихрей в квадратной области

На рис. 7.1 приведены результаты расчета при $\mu = 0{,}001$ с шагом по времени $\Delta t = 0{,}01$ для $t = 0{,}25m$, $m = 0, 1, \ldots, 11$. Начальное распределение вихря скорости $\omega_0(x, y)$ в области $\overline{D} = [0, 1] \times [0, 1]$ задавалось формулой (7.26) с параметрами $\beta = 50$; $A_1 = A_3 = 24$; $A_2 = A_4 = -24$;

$r_1 = r_2 = r_3 = r_4 = 0{,}05$; $x_1 = 0{,}2$; $x_2 = 0{,}8$; $y_1 = 0{,}25$; $y_2 = 0{,}25$; $x_3 = 0{,}2$; $x_4 = 0{,}8$; $y_3 = 0{,}75$; $y_4 = 0{,}75$; $n = 4$.

В начальный момент времени первое и третье «пятно» вращаются против часовой стрелки, а второе и четвертое — по часовой стрелке.

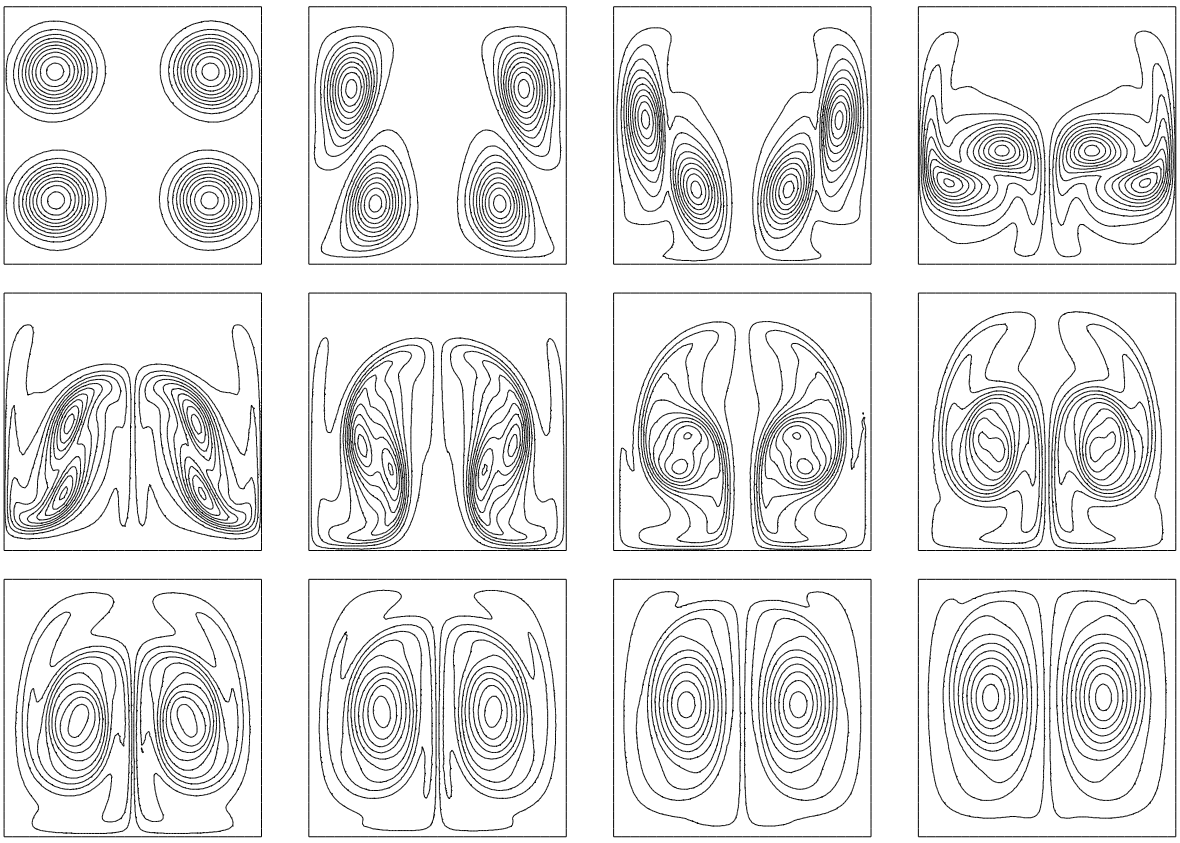

Рис. 7.1. Изолинии вихря скорости ω при $t = 0{,}25m$, $m = 0, 1, \ldots, 11$

7.5.2 Движение вихрей в круге

Одним из преимуществ FreeFem++ является тот факт, что для изменения формы области и начальных данных требуются минимальные изменения кода программы. Например, для исследования движения вихрей в круге достаточно задать границу в виде

```
border Gamma(t=0,2*pi){ x=cos(t); y=sin(t); };
```

заменить строки

```
problem Curl(omega, v) = ...
                  + on(GammaB,GammaL,GammaR,GammaT, omega=0);
problem Poisson(psi,vpsi) =  ...
                  + on(GammaB, GammaL, GammaR, GammaT, psi=0);
```

на

```
problem Curl(omega, v) = ... + on(Gamma, omega=0);
problem Poisson(psi,vpsi) = ... +  on(Gamma, psi=0);
```

При задании начальных условий (7.26) используем параметры, которые соответствуют распределению семи вихрей в круге: $\beta = 50$; $r_k = 0{,}005$, $k = 1, \ldots, 7$; $A_k = 24$, $k = 1, \ldots, 6$; $A_7 = -24$, $x_7 = 0$; $y_7 = 0$; $r_0 = 0{,}65$;

$$x_k = r_0 \cos \frac{k\pi}{3}, \quad y_k = r_0 \sin \frac{k\pi}{3}, \quad k = 1, \ldots, 6.$$

Результаты вычислений для $\mu = 0{,}001$ в круге единичного радиуса в различные моменты времени приведены на рис. 7.2. Расчет проводился с шагом $\Delta t = 0{,}0025$. Движение жидкости в целом происходит против часовой стрелки.

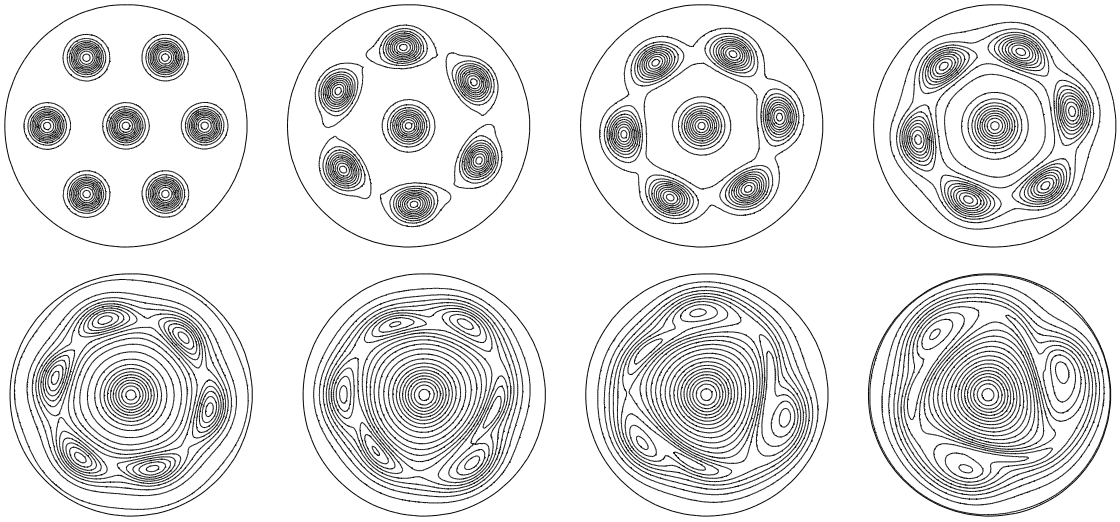

Рис. 7.2. Семь вихрей $A_k = 24$, $k = 1, \ldots, 6$, $A_7 = -24$ при $t = 0$, 1, 2, 4, 7, 9, 10, 11

Заметим, что на эволюцию вихрей достаточно сильно влияет их начальное распределение. Результаты вычислений для $A_k = 48$, $k = 1, \ldots, 6$, $A_7 = -48$, $\mu = 0{,}001$ в круге единичного радиуса приведены на рис. 7.3. Расчет проводился с шагом $\Delta t = 0{,}0025$.

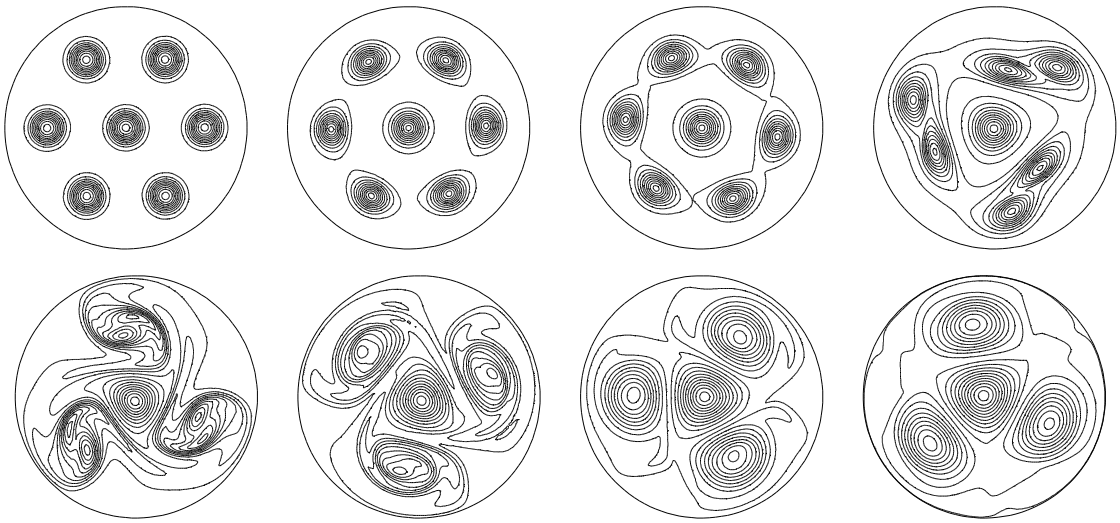

Рис. 7.3. Семь вихрей $A_k = 48$, $k = 1, \ldots, 6$, $A_7 = -48$ при $t = 0$, 1, 2, 4, 5, 6, 7, 8

На рис. 7.4 для случая $A_k = 48$, $k = 1, \ldots, 6$, $A_7 = -48$ показаны линии уровня функции тока ψ.

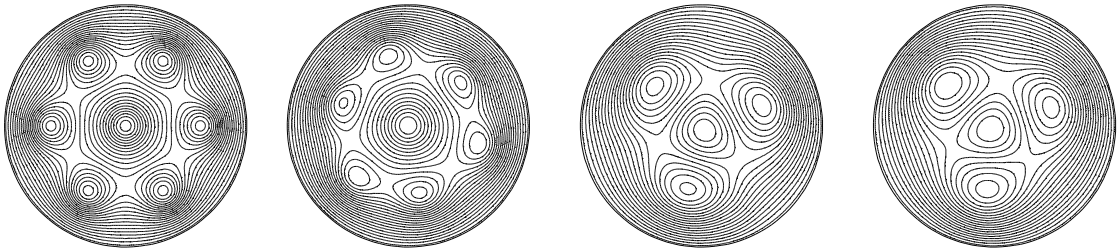

Рис. 7.4. Семь вихрей $A_k = 48$, $k = 1, \ldots, 6$, $A_7 = -48$; изолинии функция тока $\psi(x, y, t)$ при $t = 0, 3, 6, 9$

7.5.3 Периодические краевые условия. Вихри на торе

При помощи FreeFem++ можно решать задачи с периодическими краевыми условиями. Для определенности рассмотрим задачу о движении вихрей на торе, заменив в задаче (7.13)–(7.17) краевые условия (7.14), (7.17) условиями периодичности

$$\omega_t + \boldsymbol{v} \cdot \nabla\omega = \mu\Delta\omega, \quad \Delta\psi = -\omega, \quad \boldsymbol{v} = (\psi_y, -\psi_x), \quad (x, y) \in D, \tag{7.27}$$

$$\omega(0, y, t) = \omega(a, y, t), \quad \psi(0, y, t) = \psi(a, y, t), \quad 0 \leqslant y \leqslant b, \tag{7.28}$$

$$\omega(x, 0, t) = \omega(x, b, t), \quad \psi(x, 0, t) = \psi(x, b, t), \quad 0 \leqslant x \leqslant a, \tag{7.29}$$

$$\omega\big|_{t=0} = \omega_0(x, y), \quad (x, y) \in D. \tag{7.30}$$

Для реализации условия периодичности (7.28) в коде программы п. 7.4 следует записать строку 10 в виде

```
fespace Vh(Th, P2, periodic=[ [GammaL,y], [GammaR,y] ]);
```

Для реализации условия периодичности (7.29) в коде программы п. 7.4 следует записать строку 10 в виде

```
fespace Vh(Th, P2, periodic=[ [GammaB,x], [GammaT,x] ]);
```

Чтобы ***одновременно*** реализовать условия (7.28) и (7.29), в коде программы п. 7.4 следует записать строку 10 в виде

```
fespace Vh(Th, P2, periodic=[ [GammaB,x], [GammaT,x],
                              [GammaL,y], [GammaR,y] ]);
```

Формат записи условий периодичности достаточно прозрачен. Используется ключевое слово `periodic`, указываются идентификаторы границ (например, для (7.28) это `GammaL`, `GammaR`) и имя переменной, которая изменяется на данных границах (для (7.28) это `y`). Конечно, при задании условий (7.28), (7.29) из кода программы п. 7.4 следует удалить строки 29 и 33, которые реализовали задание краевых условий первого рода (7.14),

(7.17) на границах прямоугольника. Более подробно способ использования ключевого слова `periodic` описано в [1].

Топологически область D представляет собой двумерный тор — прямоугольник со «склеенными» противоположными сторонами. Во избежание недоразумений, заметим, что D не является «физическим» тором, т. е. тороидальной поверхностью $S^1 \times S^1$, где S^1 — окружности. Задача (7.27)–(7.30) для исследования поведения начального распределения вихря $\omega_0(x, y)$ решается в декартовых координатах (плоской геометрии) и кривизна поверхности не учитывается.

Результаты вычислений с шагом $\Delta t = 0{,}01$ для начальных данных, таких же как в п. 7.4, в различные моменты времени приведены на рис. 7.5.

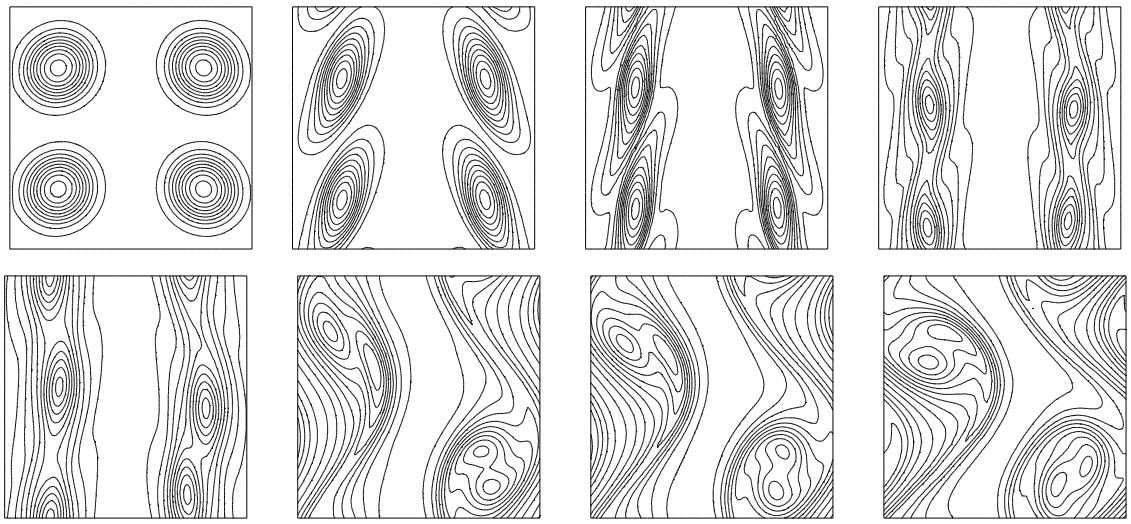

Рис. 7.5. Изолинии вихря скорости ω на торе при $t = 0{,}0$; 0,2; 0,4; 0,8; 1,2; 1,8; 1,9; 2,1

Глава 8

Тепловая конвекция

При исследовании течений жидкости численными методами уравнения Навье–Стокса для несжимаемой жидкости необязательно требуется преобразовывать к переменным вихрь–функция тока, как это было сделано в гл. 7. Для решения можно применять, например, проекционный алгоритм, предложенный в [20] (см. также [21, гл. 17]). В этой главе проекционный алгоритм использован для решения задачи о тепловой гравитационной конвекции в вязкой несжимаемой теплопроводной жидкости.

В неравномерно нагретой жидкости, по причине зависимости плотности жидкости от температуры, возникает неравномерное распределение плотности. В поле тяжести легкие (тяжелые) участки жидкости всплывают (тонут), что при определенных значениях параметров (вязкости, теплопроводности, величины характерного градиента температуры) может привести к перемешиванию жидкости и возникновению пространственно-временных структур. Такой процесс называется тепловой гравитационной конвекцией. Конечно, неравномерное распределение плотности жидкости необязательно связано с нагревом жидкости. При наличии в жидкости растворенной примеси плотность может зависеть от концентрации примеси и также возможно возникновение перемешивания раствора. В этом случае процесс называется концентрационной гравитационной конвекцией.

8.1 Постановка задачи

Пусть область D заполнена вязкой теплопроводной несжимаемой жидкостью. Система уравнений, описывающая процесс тепловой гравитационной конвекции в приближении Обербека–Буссинеска, имеет вид (см., например, [22])

$$\operatorname{div} \boldsymbol{v} = 0, \tag{8.1}$$

$$\boldsymbol{v}_t + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\nabla p + \mu \Delta \boldsymbol{v} + \boldsymbol{k}\theta, \tag{8.2}$$

$$\theta_t + \boldsymbol{v} \cdot \nabla \theta = \delta \Delta \theta, \tag{8.3}$$

здесь $\boldsymbol{v}$ — скорость, p — давление, θ — температура (точнее, отклонение температуры от некоторого фиксированного значения), μ — коэффициент кинематической вязкости, δ — коэффициент температуропроводности,

$\boldsymbol{k}$ — единичный вектор в направлении, противоположном действию силы тяжести.

Предположим, что граница области D является твердой стенкой, и зададим граничные условия, соответствующие прилипанию вязкой жидкости на границе (условия непроницаемости границы в этом случае автоматически выполнены)

$$\boldsymbol{v}\big|_\Gamma = 0. \tag{8.4}$$

Считаем также, что на границе области задана температура

$$\theta\big|_\Gamma = g(x,y)\big|_\Gamma, \tag{8.5}$$

где $g(x,y)$ — известная функция.

Кроме этого, возьмем начальные условия, отвечающие неподвижной жидкости с известным начальным распределением температуры

$$\boldsymbol{v}\big|_{t=0} = 0, \quad \theta\big|_{t=0} = \theta_0(x,y), \tag{8.6}$$

где $\theta_0(x,y)$ — известная функция.

Напомним, что член $\boldsymbol{k}\theta$ в уравнении (8.2) — это архимедова сила, приводящая к всплыванию легких «частиц» жидкости.

Для системы (8.1)–(8.3) можно ставить и иные краевые и начальные условия. В частности, можно считать границы (или части границы) области свободными недеформируемыми границами. На таких границах вместо условий прилипания (8.4) следует задавать условие непроницаемости границ для жидкости и условие отсутствия касательных напряжений. Можно также вместо (8.5) задавать поток тепла на границе или условие теплообмена. Некоторые из указанных условий будут использованы при проведении вычислений (см. п. 8.4).

Задача в покоординатной записи в случае, когда сила тяжести действует против направления оси y, т. е. $\boldsymbol{k} = (0,1)$, имеет вид

$$u_x + w_y = 0, \quad \boldsymbol{v} = (u,w), \tag{8.7}$$

$$u_t + uu_x + wu_y = -p_x + \mu(u_{xx} + u_{yy}), \tag{8.8}$$

$$w_t + uw_x + ww_y = -p_y + \mu(w_{xx} + w_{yy}) + \theta, \tag{8.9}$$

$$\theta_t + u\theta_x + w\theta_y = \delta(\theta_{xx} + \theta_{yy}), \tag{8.10}$$

$$u\big|_\Gamma = 0, \quad w\big|_\Gamma = 0, \qquad \theta\big|_\Gamma = g(x,y)\big|_\Gamma, \tag{8.11}$$

$$u\big|_{t=0} = 0, \quad w\big|_{t=0} = 0, \qquad \theta\big|_{t=0} = \theta_0(x,y). \tag{8.12}$$

8.2 Алгоритм решения

Для решения задачи (8.7)–(8.12) применим так называемый проекционный алгоритм, предложенный первоначально в [20] и исправленный в [23]. Краткое изложение алгоритма имеется, например, в [21, гл. 17, с. 400, 421].

Проекционный алгоритм используем для решения «гидродинамической части» задачи, т. е. для уравнений (8.7)–(8.9), а задачу для уравнения теплопроводности (8.10) решаем обычным образом, как это сделано, например, в пп. 5.7, 7.3.

8.2.1 Аппроксимация по времени

Для дискретизации по времени уравнений (8.8)–(8.10) обозначим

$$u(\boldsymbol{x},t_m)=u^m(\boldsymbol{x}),\quad w(\boldsymbol{x},t_m)=w^m(\boldsymbol{x}),\quad \theta(\boldsymbol{x},t_m)=\theta^m(\boldsymbol{x}),$$
$$p(\boldsymbol{x},t_m)=p^m(\boldsymbol{x}),\quad \boldsymbol{v}^m=(u^m,w^m),\quad \boldsymbol{x}=(x,y).$$

Используем для аппроксимации явно-неявную схему — линейные члены задаем в моменты $t=t_{m+1}$, а нелинейные — в моменты $t=t_m$

$$\frac{u^{m+1}(\boldsymbol{x})-u^m(\boldsymbol{x})}{\tau}+\boldsymbol{v}^m(\boldsymbol{x})\nabla u^m(\boldsymbol{x})=$$
$$=-p_x^m(\boldsymbol{x})-q_x^{m+1}(\boldsymbol{x})+\mu\Delta u^{m+1}(\boldsymbol{x}),\tag{8.13}$$
$$\frac{w^{m+1}(\boldsymbol{x})-w^m(\boldsymbol{x})}{\tau}+\boldsymbol{v}^m(\boldsymbol{x})\nabla w^m(\boldsymbol{x})=$$
$$=-p_y^m(\boldsymbol{x})-q_y^{m+1}(\boldsymbol{x})+\mu\Delta w^{m+1}(\boldsymbol{x})+\theta^{m+1}(\boldsymbol{x}),\tag{8.14}$$
$$\frac{\theta^{m+1}(\boldsymbol{x})-\theta^m(\boldsymbol{x})}{\tau}+\boldsymbol{v}^m(\boldsymbol{x})\nabla\theta^m(\boldsymbol{x})=\delta\Delta\theta^{m+1}(\boldsymbol{x}).\tag{8.15}$$

Обратим внимание, что при аппроксимации давления p в уравнениях (8.13), (8.14), на самом деле, давление задано в момент $t=t_{m+1}$

$$p^{m+1}(\boldsymbol{x})=p^m(\boldsymbol{x})+q^{m+1}(\boldsymbol{x}).\tag{8.16}$$

Величина $q^{m+1}(\boldsymbol{x})$ представляет собой добавку к давлению $p^m(\boldsymbol{x})$.

Как обычно, используя оператор `convect` (см. (5.23), (5.24)), преобразуем уравнения (8.13)–(8.15). Для функции u имеем (аналогичные формулы будут для w и θ)

$$u^m(\boldsymbol{X}^m(\boldsymbol{x}))=u^m(\boldsymbol{x})-\boldsymbol{v}^m(\boldsymbol{x})\cdot\nabla u^m(\boldsymbol{x})\tau,\quad \boldsymbol{v}=(u,w),\tag{8.17}$$
$$u^m(\boldsymbol{X}^m(\boldsymbol{x}))=\texttt{convect}([u^m(\boldsymbol{x}),w^m(\boldsymbol{x})],-\tau,u^m(\boldsymbol{x})).\tag{8.18}$$

Система (8.13)–(8.15) примет вид (далее аргумент $\boldsymbol{x}$ опускаем)

$$\frac{u^{m+1}-u^m(\boldsymbol{X}^m)}{\tau}=-p_x^m-q_x^{m+1}+\mu\Delta u^{m+1},\tag{8.19}$$
$$\frac{w^{m+1}-w^m(\boldsymbol{X}^m)}{\tau}=-p_y^m-q_y^{m+1}+\mu\Delta w^{m+1}+\theta^{m+1},\tag{8.20}$$
$$\frac{\theta^{m+1}-\theta^m(\boldsymbol{X}^m)}{\tau}=\delta\Delta\theta^{m+1}.\tag{8.21}$$

8.2.2 Метод проекций

Для решения «гидродинамической части» задачи, т. е. для (8.19), (8.20), используем метод проекций. Обратим внимание, что величину $\theta^{m+1}(\boldsymbol{x})$ в уравнении (8.20) можно считаться известной, т. к. она определяется из (8.21) обычными способами по известным величинам $\theta^m(\boldsymbol{x})$, $u^m(\boldsymbol{x})$, $w^m(\boldsymbol{x})$ (см. пп. 5.7, 7.3). В частности, это означает, что при вычислениях уравнение (8.21) должно быть решено прежде, чем будут решаться уравнения (8.19), (8.20).

Необходимость использования специального метода для решения уравнений (8.19), (8.20) связана с тем, что ***два*** уравнения (8.19), (8.20) содержат ***три*** неизвестных функции $u^{m+1}(\boldsymbol{x})$, $w^{m+1}(\boldsymbol{x})$, $q^{m+1}(\boldsymbol{x})$ и до сих пор не было использовано уравнение неразрывности (8.7). В принципе, для определения давления p можно получить уравнение, применяя операцию div к уравнению (8.2) (или вычисляя производные по x, y от (8.19), (8.20)) и учитывая (8.7)

$$(uu_x + wu_y)_x + (uw_x + ww_y)_y = -(p_{xx} + p_{yy}) + \theta_y$$

или

$$-\Delta p = -(p_{xx} + p_{yy}) = u_x u_x + 2u_y w_x + w_y w_y - \theta_y.$$

Однако, практика показала, что непосредственное использование такого уравнения для вычисления давления приводит к значительным погрешностям и малоэффективно (см., в частности, [21]).

Идея метода проекций заключается в следующем. Вместо уравнений (8.19), (8.20) рассматриваем уравнения для некоторых вспомогательных функций u^*, w^*

$$\frac{u^* - u^m(\boldsymbol{X}^m)}{\tau} = -p_x^m + \mu\Delta u^*, \tag{8.22}$$

$$\frac{w^* - w^m(\boldsymbol{X}^m)}{\tau} = -p_y^m + \mu\Delta w^* + \theta^{m+1}. \tag{8.23}$$

После определения u^*, w^* решение уравнений (8.19), (8.20) разыскиваем в виде

$$u^{m+1} = u^* - \tau q_x^{m+1}, \quad w^{m+1} = w^* - \tau q_y^{m+1}. \tag{8.24}$$

Подставляя u^{m+1}, w^{m+1} в уравнение неразрывности (8.7), получим

$$u_x^{m+1} + w_y^{m+1} = u_x^* + w_y^* - \tau\Delta q^{m+1} = 0.$$

Таким образом, для определения q^{m+1} имеем уравнение

$$\Delta q^{m+1} = \frac{u_x^* + w_y^*}{\tau} = \frac{\operatorname{div}\boldsymbol{v}^*}{\tau}. \tag{8.25}$$

К этому уравнению в случае твердых границ области следует добавить краевое условие (подробнее см. в [21])

$$\left.\frac{\partial q^{m+1}}{\partial n}\right|_\Gamma = 0, \quad \left(\boldsymbol{n}\cdot\nabla q^{m+1}\big|_\Gamma = 0\right). \tag{8.26}$$

Окончательно, после решения задачи (8.25), (8.26) по формулам (8.24) определяем u^{m+1}, w^{m+1}, а также p^{m+1} по формуле (8.16).

Заметим, что поиск решения в виде (8.24) приводит к частичной порче краевых условий прилипания (8.4) или (8.11). Действительно, если считать, что для $\boldsymbol{v}^*$ выполнены краевые условия прилипания $\boldsymbol{v}^*\big|_\Gamma = 0$, то для выполнения $\boldsymbol{v}\big|_\Gamma = 0$ требуется, чтобы на границе было выполнено условие $\nabla q^{m+1}\big|_\Gamma = 0$. Однако, краевые условия (8.26) для q^{m+1} гарантируют лишь выполнение условия непротекания жидкости через границу

$$\boldsymbol{n} \cdot \boldsymbol{v}^{m+1}\big|_\Gamma = \boldsymbol{n} \cdot \boldsymbol{v}^*\big|_\Gamma - \boldsymbol{n} \cdot \nabla q^{m+1}\big|_\Gamma = 0.$$

Условие же прилипания выполнено лишь с точностью $\mathcal{O}(\tau\|\nabla q\|)$.

8.2.3 Решение задачи Неймана

Особенно обратим внимание на то, что задача (8.25), (8.26) является стационарной задачей Неймана. Как известно, для существования решения такой задачи необходимо выполнение условия разрешимости — в данном случае, равенство нулю интеграла по области от правой части уравнения (8.25). Более того, решение задачи будет не единственным — q^{m+1} определяется с точностью до константы C. Конечно, выполнение условий разрешимости должно обеспечиваться постановкой исходной задачи. Однако, в случае численного интегрирования (8.25), (8.26) необходимо предпринимать специальные меры, для того чтобы условие разрешимости выполнялось при вычислениях, а также указать способ выбора константы C.

Предположим, что в результате вычислительной погрешности условие разрешимости не выполнено, т. е.

$$\iint\limits_D \frac{\operatorname{div} \boldsymbol{v}^*}{\tau}\, dx\, dy = A \neq 0. \tag{8.27}$$

Обозначим через S меру области D (в двумерном случае, это площадь). Вместо задачи (8.25), (8.26) рассмотрим задачу, для которой условие разрешимости будет выполнено автоматически

$$\Delta q = \frac{\operatorname{div} \boldsymbol{v}^*}{\tau} - \frac{A}{S}, \qquad \left.\frac{\partial q}{\partial n}\right|_\Gamma = 0. \tag{8.28}$$

Предположим, что найдено численное решение этой задачи. Тогда в качестве q^{m+1} возьмем

$$q^{m+1} = q - \frac{1}{S} \iint\limits_D q\, dx\, dy. \tag{8.29}$$

Фактически это означает способ выбора константы C — среднее значение решения q^{m+1} считается равным нулю.

8.2.4 Общая схема решения задачи

Метод проекций для решения уравнений гидродинамики достаточно часто используется в дальнейшем. Учитывая сравнительную сложность алгоритма, приведем его общую схему, давая краткие пояснения.

0. Считаем, что скорости u^m, w^m, давление p^m и температура θ^m известны в момент времени $t = t_m$.
1. Вычисляем выражения
$$f = \texttt{convect}([u^m, w^m], -\tau, u^m), \tag{8.30}$$
$$g = \texttt{convect}([u^m, w^m], -\tau, w^m),$$
$$h = \texttt{convect}([u^m, w^m], -\tau, \theta^m).$$
2. Решаем задачу для определения температуры θ^{m+1} в момент $t = t_{m+1}$
$$\frac{\theta^{m+1} - h}{\tau} = \delta\Delta\theta^{m+1}, \quad \theta^{m+1}\big|_\Gamma = g(x,y)\big|_\Gamma. \tag{8.31}$$
3. Решаем задачи для определение скоростей u^*, w^* (заметим, что θ^{m+1} уже определено на шаге 2 и является известной функцией)
$$\frac{u^* - f}{\tau} = \mu\Delta u^* - p_x^m, \quad u^*\big|_\Gamma = 0, \tag{8.32}$$
$$\frac{w^* - g}{\tau} = \mu\Delta w^* - p_y^m + \theta^{m+1}, \quad w^*\big|_\Gamma = 0. \tag{8.33}$$
4. Для обеспечения разрешимости задачи Неймана вычисляем константу A по формуле (8.27)
$$A = \iint\limits_D \frac{\operatorname{div}\boldsymbol{v}^*}{\tau}\,dx\,dy, \quad \operatorname{div}\boldsymbol{v}^* = u_x^* + w_y^*. \tag{8.34}$$
5. Для определения q решаем задачу (8.28)
$$\Delta q = \frac{\operatorname{div}\boldsymbol{v}^*}{\tau} - \frac{A}{S}, \qquad \left.\frac{\partial q}{\partial n}\right|_\Gamma = 0. \tag{8.35}$$
6. Вычисляем q^{m+1} по формуле (8.29), т. е. удаляя среднее из поправки к давлению
$$q^{m+1} = q - \frac{1}{S}\iint\limits_D q\,dx\,dy. \tag{8.36}$$
7. Вычисляем скорости u^{m+1}, w^{m+1} и давление p^{m+1} в момент времени $t = t_{m+1}$ по формулам (8.24), (8.16)
$$u^{m+1} = u^* - \tau q_x^{m+1}, \quad w^{m+1} = w^* - \tau q_y^{m+1}, \quad p^{m+1} = p^m + q^{m+1}. \tag{8.37}$$
8. Возвращаемся к шагу 0, предварительно выполнив замены
$$u^{m+1} \to u^m, \quad w^{m+1} \to w^m, \quad p^{m+1} \to p^m, \quad \theta^{m+1} \to \theta^m. \tag{8.38}$$

8.3 Реализация алгоритма на языке FreeFem++

Приведем код программы, при помощи которой осуществлялся вычислительный эксперимент, описанный в п. 8.4.1. По сравнению с предыдущими программами здесь использованы некоторые, ранее не встречавшиеся, приемы:

1. В конструкциях `border` использованы метки (`label`). Это позволяет объединять границы в краевых условиях, помечая различные фрагменты границы одной и той же меткой.

2. При выборе способа решения в `problem` использован `solver = LU` с ключевым словом `init`. Это означает, что при решении систем линейных уравнений будет применяться метод LU-разложений. Ключевое слово `init` разрешает ($= 0$) или запрещает ($\neq 0$) вычисление матрицы системы линейных уравнений.

Например, 22 строка программы означает, что при определении θ матрица соответствующей системы уравнений будет вычислена только при $m = 0$ — на первом шаге цикла движения по времени (см. строку 40), а на последующих шагах $m = 1, 2, \ldots$ перерасчет этой матрицы не будет производиться. В частности, это позволяет ускорить проведение расчетов.

Подробно о параметрах, которые можно использовать с ключевым словом `problem` (и `solve`), см. п. 19.1.2.

3. При выборе тестовых функций использована одна и та же тестовая функция v для всех задач (ранее тестовые функции выбирались свои для каждой неизвестной функции в задаче). В частности, это позволяет экономить ресурсы (память) при вычислениях.

4. Реализовано решение задачи для определения функции тока $\psi(\boldsymbol{x}, t)$ (см. гл. 7, формулы (7.3), (7.3), (7.16), (7.17))

```
problem pb4psi(psi, v, init=m, solver=LU)...
```

Заметим, что функция ψ использована лишь для визуализации расчетов и непосредственно для решения исходной задачи не требуется.

```
1  real a=3.0, b=2.0;
2  int m;
3  // задаем границы области (прямоугольник [0,a]x[0,b])
4  // следует сохранять ориентацию контура -- против часовой стрелки
5  border a1(t=0,1){ x=a*t; y=0; label=1;};          // bottom
6  border a2(t=0,1){ x=a; y=b*t; label=2;};          // right
7  border a3(t=0,1){ x=a*(1-t); y=b; label=3;};      // top
8  border a4(t=0,1){ x=0; y=b*(1-t); label=4;};      // left
9  int n=3;
10 mesh Th = buildmesh(a1(10*n)+a2(15*n)+a3(10*n)+a4(15*n));
11 plot(Th,wait=1);
12 fespace Vh(Th,P2);
13 real mu = 0.01,  delta=0.03;
14 // Rayleigh - Ra = 1/mu/delta, Prandtl - Pr = mu/delta
15 real dt = 0.01,  t=0;
16 Vh w, u, v, p, q, theta, psi;
```

```
17  w = 0;   u = 0;   v = 0;   p = 0;   q = 0;  theta = 10*x;   psi = 0;
18  real area= int2d(Th)(1.);
19  Vh uold, wold, pold, thetaold;
20  Vh f, g, h;
21  real meandiv, meanpq;
22  problem pb4theta(theta, v, init=m, solver=LU)                 // (8.31)
23     = int2d(Th)( theta*v/dt + delta*(dx(theta)*dx(v)+dy(theta)*dy(v)) )
24       - int2d(Th)( h/dt * v );
25  //+ on(2,3,4,theta = 0) + on(1,theta=1);                     // (8.5)
26   problem pb4u(u, v, init=m, solver=LU)                       // (8.32)
27     = int2d(Th)( u*v/dt + mu*(dx(u)*dx(v)+dy(u)*dy(v)) )
28       - int2d(Th)( (f/dt-dx(p))*v )
29       + on(1,2,3,4,u = 0);
30   problem pb4w(w, v, init=m, solver=LU)                       // (8.33)
31     = int2d(Th)(w*v/dt +mu*(dx(w)*dx(v)+dy(w)*dy(v)))
32       - int2d(Th)( (g/dt-dy(p)+theta)*v )
33       + on(1,2,3,4, w = 0);
34   problem pb4p(q, v, init=m, solver=LU)                       // (8.35)
35     = int2d(Th)( dx(q)*dx(v)+dy(q)*dy(v) )
36       + int2d(Th)( (dx(u)+ dy(w)-meandiv)*v/dt );  // meandiv = A*dt/S
37   problem pb4psi(psi, v, init=m, solver=LU)
38     = int2d(Th)( dx(psi)*dx(v)+dy(psi)*dy(v) )
39       - int2d(Th)( (dx(w)- dy(u))*v ) + on(1,2,3,4, psi=0);
40   for(m=0; m<10000; m++)
41   {
42     t = t+dt;
43     uold = u;   wold = w;   pold = p;   thetaold = theta;     // (8.38)
44     f = convect([u,w],-dt,uold);                              // (8.30)
45     g = convect([u,w],-dt,wold);                              // (8.30)
46     h = convect([u,w],-dt,thetaold);                          // (8.30)
47     pb4theta;
48     pb4u;
49     pb4w;
50     meandiv = int2d(Th)(dx(u)+dy(w))/area;                    // (8.34)
51     pb4p;
52     meanpq = int2d(Th)(pold + q)/area;                        // (8.36)
53     p = pold+q-meanpq;                                        // (8.37)
54     u = u - dx(q)*dt;                                         // (8.37)
55     w = w - dy(q)*dt;                                         // (8.37)
56     pb4psi;
57     plot(psi, fill=0,cmm=+t);
58   }
```

8.4 Вычислительный эксперимент

Расчеты проводились для различных областей. Приведенная программа соответствует результатам, изложенным в п. 8.4.1. Для остальных примеров требуется модификация кода, в основном связанная с заданием областей различной формы.

8.4.1 Теплоизолированная прямоугольная область

Результаты расчетов для прямоугольной области $\overline{D} = [0,2] \times [0,3]$ при начальном распределении температуры $\theta_0(x,y) = 10x$ приведены на рис. 8.1 (изолинии температуры и функции тока). Расчеты проводились для параметров $\mu = 0{,}01$, $\delta = 0{,}03$ с шагом по времени $\tau = 0{,}01$.

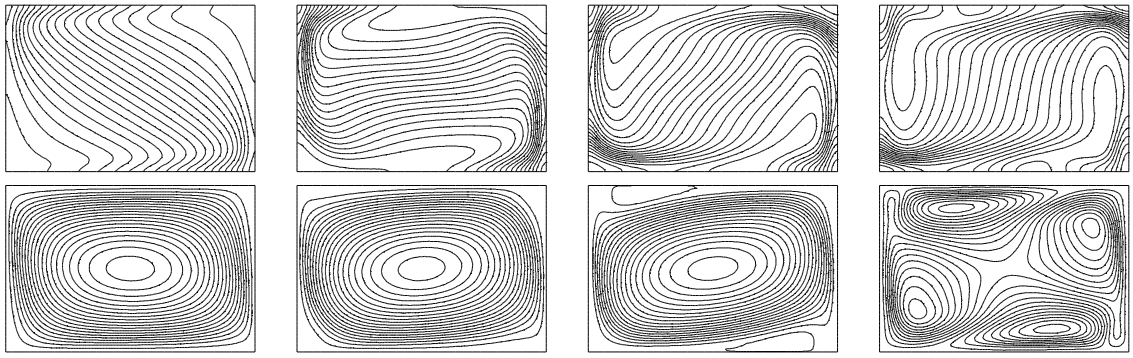

Рис. 8.1. Изолинии температуры $\theta(x,y)$ (верхний ряд) и функции тока $\psi(x,y)$ в моменты времени $t = 0{,}6;\ 1{,}0;\ 1{,}4;\ 1{,}8$

В качестве краевых условий для температуры выбраны краевые условия Неймана, т. е. граница области предполагается теплоизолированной

$$(\nabla\theta \cdot \boldsymbol{n})\big|_{\Gamma} = 0. \tag{8.39}$$

Эти условия отличаются от краевых условий (8.5). Для использования (8.5) следует удалить комментарии в строке 25, в которой записаны условия задания температуры на сторонах прямоугольника

$$\theta\big|_{\text{Bottom}} = 1, \quad \theta\big|_{\text{Top}} = 0, \quad \theta\big|_{\text{Left}} = 0, \quad \theta\big|_{\text{Right}} = 0.$$

На рис. 8.1 показаны лишь начальные стадии процесса перемешивания жидкости. С течением времени температура всей области D при краевых условиях (8.39) станет постоянной. Действительно, умножая уравнение (8.3) на θ и интегрируя по области D с учетом условия (8.39), получим

$$\frac{1}{2}\frac{d}{dt}\iint\limits_D \theta^2(x,y,t)\,dx\,dy = -\delta \iint\limits_D (\nabla\theta)^2\,dx\,dy \leqslant 0. \tag{8.40}$$

Это означает, что либо $\theta = \text{const}$, либо $\theta^2 \to 0$ при $t \to +\infty$.

Аналогично, умножая (8.2) на $\boldsymbol{v}$ и интегрируя, с учетом уравнения неразрывности (8.1) и краевых условий (8.4) при постоянной θ имеем

$$\frac{1}{2}\frac{d}{dt}\iint\limits_D \boldsymbol{v}^2(x,y,t)\,dx\,dy = -\mu \iint\limits_D (\nabla\boldsymbol{v})^2\,dx\,dy + \theta\frac{d}{dt}\iint\limits_D y\,dx\,dy \leqslant 0. \tag{8.41}$$

Здесь использовано соотношение $\boldsymbol{k}\cdot\boldsymbol{v} = w = dy/dt$. Последнее слагаемое в (8.41) обращается в нуль, т. к. интеграл по всей области не зависит от t, и из (8.41) следует, что $\boldsymbol{v} = \text{const}$ и с учетом краевых условий $\boldsymbol{v} = 0$.

Иными словами, в случае однородных краевых условий Неймана для температуры финальная стадия процесса (при $t \to +\infty$) — неподвижная жидкость с постоянной температурой.

8.4.2 Теплоизолированный круг

Результаты расчета для круга единичного радиуса с теплоизолированными стенками и с теми же самыми параметрами, что и в предыдущем случае: $\theta_0(x, y) = 10x$, $\mu = 0{,}01$, $\delta = 0{,}03$, $\tau = 0{,}01$, приведены на рис. 8.2.

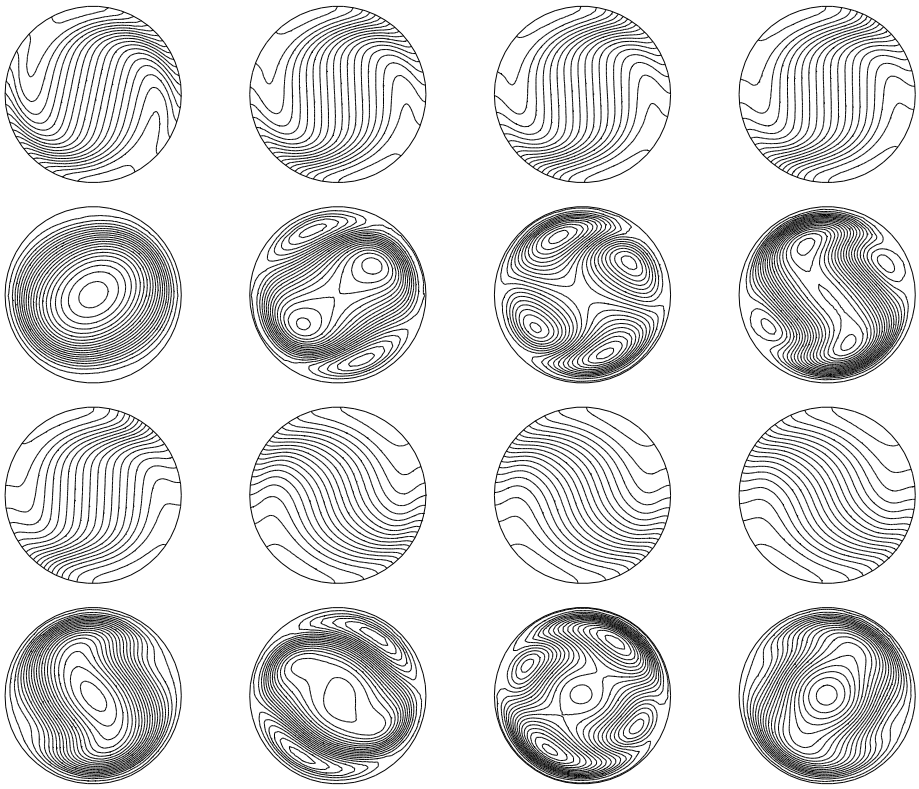

Рис. 8.2. Изолинии температуры $\theta(x, y)$ (1 и 3 ряды) и функции тока $\psi(x, y)$ (2 и 4 ряды) при $t = 1{,}4;\ 1{,}7;\ 1{,}8;\ 1{,}9;\ 2{,}0;\ 3{,}7;\ 3{,}8;\ 3{,}9$

При движении жидкости в круглом контейнере для выбранных значений параметров на начальных этапах времени наблюдается интересный, сложный, близкий к колебательному режим. Два верхних ряда на рисунке (температура и функция тока) соответствуют вращению против часовой стрелки; два нижних ряда соответствуют вращению по часовой стрелке. Примерный интервал между двумя такими «колебательными» вращениями $t \approx 2$.

Как и в случае, рассмотренном в п. 8.4.1, при $t \to +\infty$ движение жидкости прекратится и температура области станет постоянной.

8.4.3 Тепловая конвекция в «чайнике»

На рис. 8.3 приведены результаты расчетов тепловой конвекции в «чайнике». Конечно, рассматриваемый чайник весьма экзотичен — это сечение в плоскости xy бесконечного трехмерного чайника. Дно чайника поддерживается при заданной температуре $\theta = 40$, а остальная граница предполагается теплоизолированной. Начальное распределении температуры $\theta_0(x, y) = 0$. Расчеты проводились для параметров $\mu = 0{,}00001$, $\delta = 0{,}03$

с шагом по времени $\tau = 0{,}01$. Большая часть верхней границы — окружность единичного радиуса.

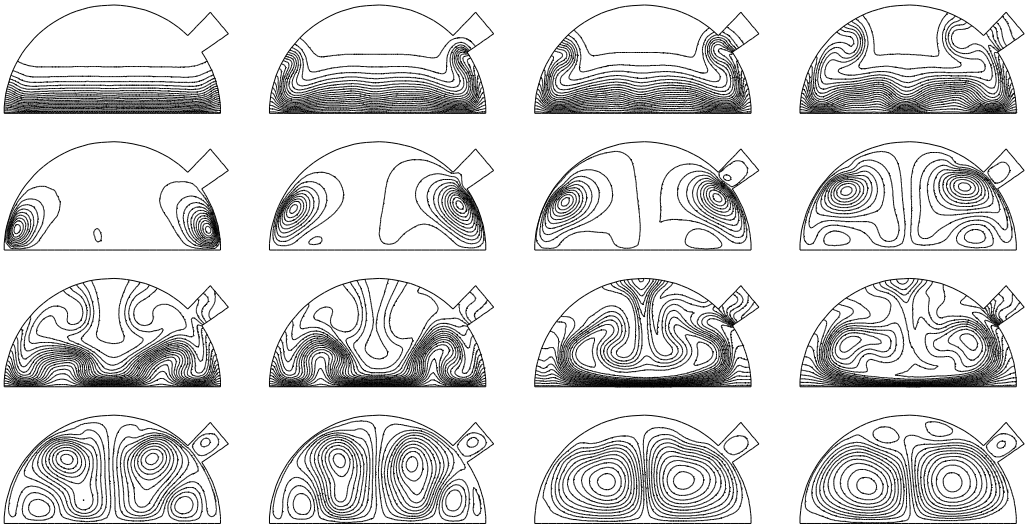

Рис. 8.3. Изолинии температуры $\theta(x, y)$ (1 и 3 ряды) и функции тока $\psi(x, y)$ (2 и 4 ряды) при $t = 0{,}6;\ 1{,}0;\ 1{,}1;\ 1{,}3;\ 1{,}5;\ 1{,}6;\ 2{,}0;\ 2{,}1$

На рисунке хорошо виден процесс интенсивного перемешивания жидкости и перераспределение температуры в подогреваемом снизу чайнике. Заметим, что перераспределение температуры в основном вызывается движением жидкости, а не процессом теплопроводности — нагретые возле дна чайника слои жидкости переносятся ее течением.

8.4.4 Тепловая конвекция в «электрочайнике»

Расчеты тепловой конвекции в случае, когда в теплоизолированной прямоугольной области $[0, 1] \times [0, 2]$ находится круг с центром в точке $(0{,}5; 0{,}5)$ и радиусом $r = 0{,}25$, на котором поддерживается постоянная температура $\theta = 40$, $\mu = 0{,}00001$, $\delta = 0{,}03$ приведены на рис. 8.4.

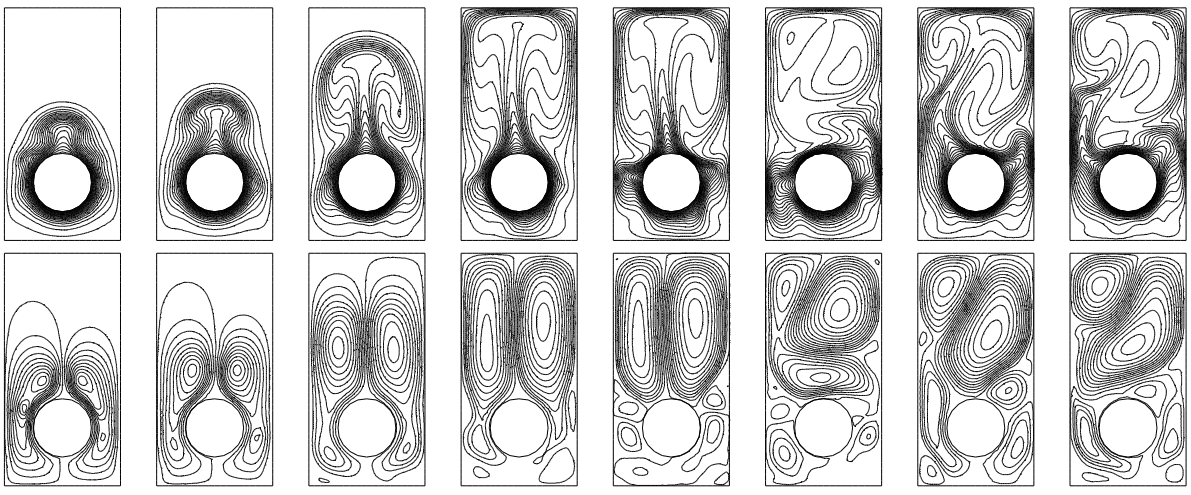

Рис. 8.4. Изолинии температуры $\theta(x, y, t)$ (вверху) и функции тока $\psi(x, y, t)$ (внизу) при $t = 0{,}34;\ 0{,}45;\ 0{,}78;\ 1{,}22;\ 1{,}44;\ 1{,}87;\ 2{,}75;\ 2{,}86$

Это фактически модель электрического чайника — роль нагревательного элемента играет круг внутри области. Начальное распределении температуры $\theta_0(x, y) = 0$. Расчеты проводились для параметров $\mu = 0{,}00001$, $\delta = 0{,}03$ с шагом по времени $\tau = 0{,}01$.

Как и в предыдущем примере, перераспределение температуры происходит в основном в результате конвективного перемешивания жидкости, а не в результате процесса теплопроводности.

8.5 Числа подобия

Обычно при описании тепловой гравитационной конвекции используются числа Рэлея (Ra), Грасгофа (Gr) и Прандтля (Pr). Связь этих чисел с параметрами μ, δ можно получить, производя формальные замены в уравнениях (8.1)–(8.3)

$$t \to t\mu^{-1}, \quad \boldsymbol{v}(x, y, t) \to \mu \boldsymbol{v}(x, y, t), \quad \theta(x, y, t) \to A\theta_0(x, y) + A\theta(x, y, t),$$

где A — характерное значение разности температур на единицу длины (характерный градиент), $\theta_0(x, y)$ — некоторое стационарное распределение температуры, такое, что характерное значение $\nabla\theta_0(x, y)$ равно единице.

Тогда уравнения (8.1)–(8.3) примут вид

$$\operatorname{div} \boldsymbol{v} = 0,$$

$$\boldsymbol{v}_t + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\nabla p + \Delta \boldsymbol{v} + \operatorname{Ra} \theta \boldsymbol{k},$$

$$\operatorname{Pr} (\theta_t + \boldsymbol{v} \cdot \nabla\theta + \boldsymbol{v} \cdot \nabla\theta_0) = \Delta\theta,$$

где

$$\operatorname{Ra} = \frac{A}{\mu\delta}, \quad \operatorname{Pr} = \frac{\mu}{\delta}, \quad \operatorname{Gr} = \frac{\operatorname{Ra}}{\operatorname{Pr}} = \frac{A}{\mu^2}.$$

Добрый день! В нашем вузе второй год внедряют программы Plany для формирования нагрузки. Подскажите, пожалуйста, 1) действительно ли обязательно использование программы Plany для аккредитации вуза? Если да, то кто в вузе обязан этим заниматься - Учебная часть, деканаты факультетов или сотрудники кафедр? 2) Вы когда-нибудь работали с планом одной кафедры, насчитывающем в листе УчНагр 800 строк? Реально работать? - заполнить список преподавателей кафедры, который не хочет сохраняться; отделить зачеты от лекций в 600 строках; для 700 строк выбрать фио преподавателя, набранного 3 пунктом; сформировать нагрузку в таком виде, чтобы реальный преподаватель мог разобраться в листе, выдаваемой программой Plany; а при формировании индивидуального плана преподавателя вдруг получить в списке курсов названия чужих курсов или часы, которые никак не получаются при суммировании по строкам и столбцам? Если да, то раскройте секрет - бесплатно?

Глава 9

Различные течения жидкости

Рассмотренный в гл. 8 проекционный алгоритм решения задач гидродинамики можно применять не только для задач, связанных с тепловой конвекцией. Программы в этом случае даже проще, т. к. для тепловой конвекции, помимо гидродинамической части задачи, приходится дополнительно решать уравнение теплопроводности.

В данной главе рассмотрены задачи о течении жидкости в каналах сложной формы и задачи об обтекании жидкостью различных препятствий. Основная цель — демонстрация возможностей языка FreeFem++ и проведение вычислительного эксперимента для некоторых наборов параметров.

9.1 Задача о течении жидкости в полости

Рассмотрим задачу о течении вязкой несжимаемой жидкости в прямоугольной области с непроницаемыми стенками в случае, когда на одной из сторон прямоугольника задана постоянная касательная скорость жидкости $u_0 \neq 0$. Решение такой задачи позволяет, например, ответить на вопрос, что происходит в ямах (или впадинах) на дне реки, текущей над ямой со скоростью u_0.

9.1.1 Постановка задачи

Уравнения Навье–Стокса, описывающие течение вязкой несжимаемой жидкости имеют вид

$$\boldsymbol{v}_t + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\nabla p + \mu \Delta \boldsymbol{v}, \quad \operatorname{div} \boldsymbol{v} = 0, \quad (x, y) \in \overline{D} = [0, a] \times [0, b]. \quad (9.1)$$

Краевые условия для прямоугольника $\overline{D} = [0, a] \times [0, b]$ зададим в форме

$$\boldsymbol{v}\big|_{\partial D \setminus \Gamma} = 0, \quad u\big|_{\Gamma} = u_0, \quad w\big|_{\Gamma} = 0, \quad \Gamma = \{(x, y) : 0 \leqslant x \leqslant a, \quad y = b\}. \quad (9.2)$$

Здесь Γ — одна из сторон прямоугольника («верхняя»).

Начальные условия, соответствующие неподвижной жидкости, будут

$$\boldsymbol{v}\big|_{t=0} = 0. \quad (9.3)$$

Заметим, что вместо прямоугольника аналогичным образом можно рассматривать и любую другую область, имеющую некоторый прямой участок, на котором задана постоянная касательная скорость v_τ и нулевая нормальная скорость v_n

$$v_n = \boldsymbol{v} \cdot \boldsymbol{n} = 0, \quad v_\tau = \boldsymbol{v} - \boldsymbol{n}(\boldsymbol{v} \cdot \boldsymbol{n}) \neq 0.$$

Остальная часть непроницаемой границы может иметь любую форму и на этой части границы для вязкой жидкости следует задавать условия прилипания $\boldsymbol{v} = 0$.

9.1.2 Реализация алгоритма на языке FreeFem++

```
int m;
real a=1, b=2;
border GammaB(t=0,1){ x=a*t;     y=0;       label=1; };
border GammaR(t=0,1){ x=a;       y=b*t;     label=1; };
border GammaT(t=0,1){ x=a*(1-t); y=b;       label=2; };
border GammaL(t=0,1){ x=0;       y=b*(1-t); label=1; };
int n=6;
mesh Th= buildmesh(GammaB(5*n)+GammaR(5*n)+GammaT(5*n)+GammaL(5*n));
fespace Vh(Th,P2);
real mu = 0.0025;    // Raynolds = 400 = Re = 1/mu
real dt = 0.01,  t=0;
Vh w, u, v, p, q, psi;
w = 0;  u = 0;  v = 0;   p = 0;   q = 0;  psi = 0;
real area= int2d(Th)(1.);
Vh uold, wold, pold, f, g;
real meandiv, meanpq;
real u0=90;
problem pb4u(u, v, init=m, solver=LU)
        = int2d(Th)( u*v/dt + mu*(dx(u)*dx(v)+dy(u)*dy(v)) )
          - int2d(Th)( (f/dt-dx(p))*v )
          + on(1,u =0) + on(2,u =u0) ;
problem pb4w(w, v, init=m, solver=LU)
        = int2d(Th)(w*v/dt +mu*(dx(w)*dx(v)+dy(w)*dy(v)))
          - int2d(Th)( (g/dt-dy(p))*v )
          + on(1,2, w = 0) ;
problem pb4p(q, v, init=m, solver=LU)
        = int2d(Th)( dx(q)*dx(v)+dy(q)*dy(v) )
          + int2d(Th)( (dx(u)+ dy(w)-meandiv)*v/dt );
problem pb4psi(psi, v, init=m, solver=LU)
        = int2d(Th)(dx(psi)*dx(v)+dy(psi)*dy(v))
          - int2d(Th)((dx(w)- dy(u))*v)
          +  on(1, 2, psi=0);
for(m=0; m<10000; m++)
{  t = t+dt;
   uold = u;   wold = w;   pold = p;
   f = convect([u,w],-dt,uold);
   g = convect([u,w],-dt,wold);
   pb4u;
```

```
39      pb4w;
40      meandiv = int2d(Th)(dx(u)+dy(w))/area;
41      pb4p;
42      meanpq = int2d(Th)(pold + q)/area;
43      if(m==50){ Th = adaptmesh(Th,u,w,q); }
44      p = pold+q-meanpq;
45      u = u - dx(q)*dt;
46      w = w - dy(q)*dt;
47      pb4psi;
48      plot(psi,fill=0,cmm=+t);
49    }
```

9.1.3 Вычислительный эксперимент

На рис. 9.1 приведены результаты расчетов при $\mu = 0{,}0025$ с шагом по времени $\tau = 0{,}01$ для области $[0,1] \times [0,2]$. На «верхней» жидкой границе прямоугольной полости задана касательная скорость $u_0 = 90$. В результате в полости развивается вихревое течение.

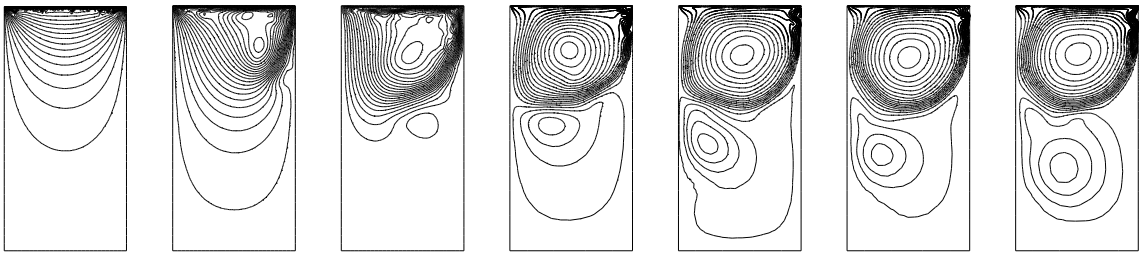

Рис. 9.1. Изолинии функции тока $\psi(x, y)$ при $t = 0{,}01;\ 0{,}2;\ 0{,}4;\ 0{,}6;\ 1{,}0;\ 1{,}2;\ 2{,}0$

Чтобы произвести расчеты в иной области, например, в полукруге единичного радиуса, достаточно заменить строки 3–8 следующими

```
border GammaT(t=0,1)    { x=1-2*t;  y=0;      label=2; };
border GammaC(t=pi,2*pi){ x=cos(t); y=sin(t); label=1; };
int n=3;
mesh Th= buildmesh(GammaT(10*n)+GammaC(20*n));
```

Результаты расчетов для параметров $\mu = 0{,}0025$ с шагом по времени $\tau = 0{,}01$ для полукруга единичного радиуса приведены на рис. 9.2. На «верхней» жидкой границе полости задана касательная скорость $u_0 = 10$. Сравнение рис. 9.1 и 9.2 показывают, что картина течения существенно зависит от формы полости.

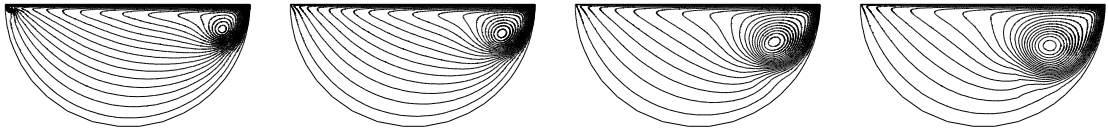

Рис. 9.2. Изолинии функции тока $\psi(x, y)$ при $t = 0{,}5;\ 0{,}6;\ 0{,}8;\ 1{,}0$

9.2 Течение в канале

Задачи о протекании жидкости через некоторую область являются достаточно сложными, как для аналитического исследования, так и для численного моделирования. В частности, трудной является проблема численной реализации краевых условий на участках границы области Γ_{in}, Γ_{out}, через которые происходит втекание (Γ_{in}) и вытекание (Γ_{out}) жидкости. Обычно, на входе в область (Γ_{in}) задается профиль скорости течения. На выходе из области (Γ_{out}) чаще всего при вычислениях задается давление и некоторые дополнительные условия для скорости.

Далее рассмотрена задача о протекании жидкости через канал Т-образной формы. В некотором смысле, решение такой задачи более корректно моделирует поведение жидкости в полости. В отличие от задач п. 9.1, течение над полостью, например, течение реки над впадиной или ямой, описывается естественным образом, а не имитируется заданием касательной скорости на границе полости, соприкасающейся с основным течением.

9.2.1 Постановка задачи

Рассмотрим Т-образную область D, через участки границы которой протекает жидкость (см. рис. 9.3). Течение вязкой несжимаемой жидкости описывается уравнениями Навье–Стокса

$$\boldsymbol{v}_t + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\nabla p + \mu \Delta \boldsymbol{v}, \quad \operatorname{div} \boldsymbol{v} = 0, \quad (x, y) \in D, \tag{9.4}$$

где $\boldsymbol{v}$ — скорость, p — давление, μ — кинематическая вязкость жидкости.

Считаем, что участок границы $\partial D \setminus (\Gamma_{\text{in}} \cup \Gamma_{\text{out}})$ является твердой непроницаемой стенкой, и зададим на этом участке условия прилипания

$$\boldsymbol{v}\big|_{\partial D \setminus (\Gamma_{\text{in}} \cup \Gamma_{\text{out}})} = 0. \tag{9.5}$$

На входной части границы Γ_{in} зададим профиль скорости $\boldsymbol{v}_0(x, y)$

$$\boldsymbol{v}\big|_{\Gamma_{\text{in}}} = \boldsymbol{v}_0. \tag{9.6}$$

Участок границы Γ_{out}, через который жидкость вытекает из области D, для простоты считаем параллельным оси y (см. рис. 9.3) и задаем следующие краевые условия

$$p\big|_{\Gamma_{\text{out}}} = p_0, \quad \left.\frac{du}{dt}\right|_{\Gamma_{\text{out}}} = 0, \quad w\big|_{\Gamma_{\text{out}}} = 0. \tag{9.7}$$

Такие условия означают, что выходной участок является свободной границей, через которую жидкость вытекает из области в окружающую среду. Естественно считать, что давление на границе в этом случае постоянно и совпадает с давлением окружающей среды. Условие $w = 0$ на границе означает отсутствие течения жидкости вдоль границы.

Наконец, условие $du/dt = 0$ на границе является условием сохранения горизонтальной компоненты скорости u в «жидкой частице». Если для интегрирования уравнений используется метод характеристик, изложенный в п. 5.2, то условие сохранения u в жидкой частице фактически означает, что на выходном участке формируется профиль скорости, такой же, как в непосредственной близости от границы.

Кроме этого, в момент времени $t = 0$ задаем условие, соответствующее начальному течению

$$\boldsymbol{v}\big|_{t=0} = \boldsymbol{h}(x, y), \tag{9.8}$$

где $\boldsymbol{h}(x, y)$ — известная функция.

Для определенности, далее рассмотрим область Т-образной формы, показанную на рис. 9.3. Границы области задаются отрезками линий со следующими координатами концов отрезка: $x_1 = 0$, $y_1 = 0$; $x_2 = 1$, $y_2 = 0$; $x_3 = 1$, $y_3 = -0{,}4$; $x_4 = 1{,}5$, $y_4 = -0{,}4$; $x_5 = 1{,}5$, $y_5 = 0$; $x_6 = 5{,}5$, $y_6 = 0$; $x_7 = 5{,}5$, $y_7 = 1$; $x_8 = 0$, $y_8 = 1$.

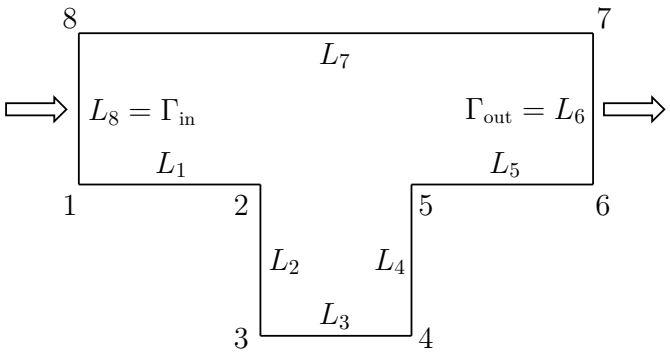

Рис. 9.3. Область D и ее границы

Отрезки, соединяющие точки с координатами (x_k, y_k) и (x_{k+1}, y_{k+1}), обозначим L_k. В этом случае

$$\Gamma_{\text{in}} = L_8, \quad \Gamma_{\text{out}} = L_6 \quad \partial D \setminus (\Gamma_{\text{in}} \cup \Gamma_{\text{out}}) = L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5 \cup L_7.$$

На входной части границы зададим следующий профиль скорости (соответствует течению Пуазейля)

$$u\big|_{L_8} = 4y(1 - y), \quad w\big|_{L_8} = 0. \tag{9.9}$$

Будем считать, что начальное распределение скорости в части канала $0 < y \leqslant 1$, $0 < x \leqslant x_6 = x_7 = 5{,}5$ — течение Пуазейля всюду, за исключением нижней полости Т-образной области

$$u\big|_{t=0} = 4y(1 - y), \quad (x, y) \in D_0 = \big\{0 < y \leqslant 1,\ 0 < x \leqslant x_6 = x_7 = 5{,}5\big\},$$
$$u\big|_{t=0} = 0, \quad (x, y) \in D \setminus D_0, \quad w\big|_{t=0} = 0, \quad (x, y) \in D. \tag{9.10}$$

Для визуализации течения решаем вспомогательную задачу для определения функции тока ψ

$$\Delta\psi = -\omega, \quad \omega = w_x - u_y, \quad u = \psi_y, \quad w = -\psi_x. \tag{9.11}$$

Краевые условия для ψ возьмем в виде

$$\left.\frac{\partial\psi}{\partial n}\right|_{L_8\cup L_6} = 0, \quad \psi\big|_{L_1\cup L_2\cup L_3\cup L_4\cup L_5} = 0, \quad \psi\big|_{L_7} = 2/3. \tag{9.12}$$

Напомним, что расход жидкости через произвольное сечение области, параллельное оси y, например, $x = x_*$, $y_{\rm bot} \leqslant y \leqslant y_{\rm top}$, вычисляется по формуле

$$Q(x_*) = \int\limits_{y_{\rm bot}}^{y_{\rm top}} u(x_*, y)\, dy = \int\limits_{y_{\rm bot}}^{y_{\rm top}} \psi_y(x_*, y)\, dy = \psi(x_*, y_{\rm top}) - \psi(x_*, y_{\rm bot}). \tag{9.13}$$

В случае несжимаемой жидкости очевидно, что расход жидкости через границу $\Gamma_{\rm in}$ должен совпадает с расходом жидкости через границу $\Gamma_{\rm out}$

$$Q\big|_{L_8} = Q\big|_{L_6}. \tag{9.14}$$

Для течения Пуазейля $u = 4y(1-y)$, $w = 0$ функция тока ψ, с точностью до несущественной постоянной, имеет вид

$$\psi = 2y^2 - \frac{4}{3}y^3.$$

Расход жидкости на входном участке канала будет

$$Q\big|_{L_8} = \int\limits_0^1 4y(1-y)\, dy = 2/3.$$

В соответствие с (9.13) это означает, что если на границе $L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$ задано условие $\psi = 0$, то на границе L_7 для ψ имеем $\psi = 2/3$, т. е. одно из условий (9.12).

Наконец, заметим, что условия для ψ на границах L_8 и L_6 означают равенство нулю соответствующей компоненты скорости $\boldsymbol{v} = (u, w)$, т. е. $w = 0$

$$\left.\frac{\partial\psi}{\partial n}\right|_{L_8\cup L_6} = \psi_x\big|_{L_8\cup L_6} = -w\big|_{L_8\cup L_6} = 0.$$

Впрочем, для границы L_8 (но не для L_6!) можно было задать условие

$$\psi\big|_{L_8} = 2y^2 - \frac{4}{3}y^3, \quad (\psi_y = u = 4y(1-y)).$$

9.2.2 Реализация алгоритма на языке FreeFem++

По сравнению с ранее указанными способами построения сетки здесь использована возможность языка FreeFem++ создавать равномерную сетку

при помощи оператора `adaptmesh` (см. строки 21–24). Это особенно важно при проведении точных расчетов. Другой способ применения оператора `adaptmesh` показан в строке 55 — размеры ячеек сетки автоматически уменьшаются (увеличиваются) в зависимости от сгущения изолиний функций u, w, q, ψ. Такой прием может оказаться не очень эффективным в случаях, когда в некоторой области функции будут сильно пространственно неоднородны, а в бóльшей части области почти постоянны. Это приведет к тому, что ячейки сетки в бóльшей части области будут крупными, что может существенно сказаться на качестве расчета.

В программе также предусмотрен контроль за точностью вычислений. Для этого в строках 61–62 вычисляются расходы $Q\big|_{L_8}$, $Q\big|_{L_6}$ и их разность $Q = Q\big|_{L_8} - Q\big|_{L_6}$.

Для визуализации некоторой части области можно использовать параметр `bb` оператора `plot`. Так, например, 59 строка означает, что вывод данных будет осуществляться в прямоугольнике с координатами левого нижнего угла $(0{,}5; -0{,}42)$ и координатами правого верхнего угла $(2; 0{,}7)$.

```
1  int m;
2  real x1,x2,x3,x4,x5,x6,x7,x8,   y1,y2,y3,y4,y5,y6,y7,y8;
3  real w1,w2,w3,h1,h2;
4  w1=1; w2=0.5;  w3=4;  h1=1;  h2=0.4;
5  x1=0;     y1=0; x2=w1;       y2=0; x3=w1;    y3=-h2; x4=w1+w2; y4=-h2;
6  x5=w1+w2; y5=0; x6=w1+w2+w3; y6=0; x7=w1+w2+w3; y7=h1;  x8=0;  y8=h1;
7  border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2; label=1; };
8  border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3; label=1; };
9  border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4; label=1; };
10 border L4(t=0,1){ x=(1-t)*x4+t*x5;  y=(1-t)*y4+t*y5; label=1; };
11 border L5(t=0,1){ x=(1-t)*x5+t*x6;  y=(1-t)*y5+t*y6; label=1; };
12 border L6(t=0,1){ x=(1-t)*x6+t*x7;  y=(1-t)*y6+t*y7; label=2; }; //out
13 border L7(t=0,1){ x=(1-t)*x7+t*x8;  y=(1-t)*y7+t*y8; label=4; };
14 border L8(t=0,1){ x=(1-t)*x8+t*x1;  y=(1-t)*y8+t*y1; label=3; }; //in
15 int n=2;
16 mesh Th = buildmesh(L1(10*n)+L2(5*n)+L3(10*n)+L4(5*n)
17                    +L5(10*n)+L6(10*n)+L7(30*n)+L8(10*n));
18 plot(Th,wait=1);
19 fespace Vh(Th,P2b);
20 real hinit = 0.05; // начальный размер сетки
21 Vh hh = hinit;     // конечно-элементная функция для размера сетки
22 // построение сетки заданного размера
23 Th = adaptmesh(Th, hh, IsMetric=1, splitpbedge=1, nbvx=10000);
24 plot(Th,wait=1);
25 real mu = 0.000025;    // Raynolds = 40000 = Re = 1/mu
26 real dt = 0.01,  t=0;
27 Vh w, u, v, p, q, psi;
28 u = 4*y*(1-y);   w = 0;   v = 0;   p = 0;   q = 0;   psi = 0;
29 real area = int2d(Th)(1.);
30 Vh uold, wold, pold, f, g;
31 real meandiv, meanpq;
32 problem pb4u(u, v, init=m, solver=LU)
```

```
33          = int2d(Th)( u*v/dt + mu*(dx(u)*dx(v)+dy(u)*dy(v)) )
34            - int2d(Th)( (f/dt-dx(p))*v )
35            + on(1,4,u = 0) + on(2,u = f) + on(3,u = 4*y*(1-y)) ;
36  problem pb4w(w, v, init=m, solver=LU)
37          = int2d(Th)(w*v/dt +mu*(dx(w)*dx(v)+dy(w)*dy(v)))
38            - int2d(Th)( (g/dt-dy(p))*v )
39            + on(1,2,3,4, w = 0);
40  problem pb4p(q, v, init=m, solver=LU)
41          = int2d(Th)( dx(q)*dx(v)+dy(q)*dy(v) )
42            + int2d(Th)( (dx(u)+ dy(w)-meandiv)*v/dt )
43            + on(2,q = 0) ;  //  meandiv = A*dt/S
44  problem pb4psi(psi, v, init=m, solver=LU)
45          = int2d(Th)(dx(psi)*dx(v)+dy(psi)*dy(v))
46            - int2d(Th)((dx(w)- dy(u))*v)
47            +  on(1, psi=0)  +  on(4, psi=2/3);
48  for(m=0; m<10000; m++)
49  { t = t+dt;    uold = u;   wold = w;   pold = p;
50    f = convect([u,w],-dt,uold);   g = convect([u,w],-dt,wold);
51    pb4u;   pb4w;
52    meandiv = int2d(Th)(dx(u)+dy(w))/area;
53    pb4p;
54    meanpq = int2d(Th)(pold + q)/area;
55    if(m==20){  Th = adaptmesh(Th,u,w,q,psi);  }
56    p = pold+q-meanpq;
57    u = u - dx(q)*dt;   w = w - dy(q)*dt;
58    pb4psi;
59    //  plot(psi, fill=0,cmm=+t,bb=[[0.5,-0.42],[2,0.7]]);
60    plot(psi, fill=0);
61    real Qin = int1d(Th,2)(u), Qout = int1d(Th,3)(u) ;
62    cout <<"Q= "<<Qin-Qout << endl;
63  }
```

9.2.3 Вычислительный эксперимент

9.2.3.1 Течение в канале T-образной формы

Результаты расчетов при $\mu = 0{,}000025$ с шагом по времени $\tau = 0{,}01$ для задачи (9.4), (9.9), (9.10) приведены на рис. 9.4 и рис. 9.5.

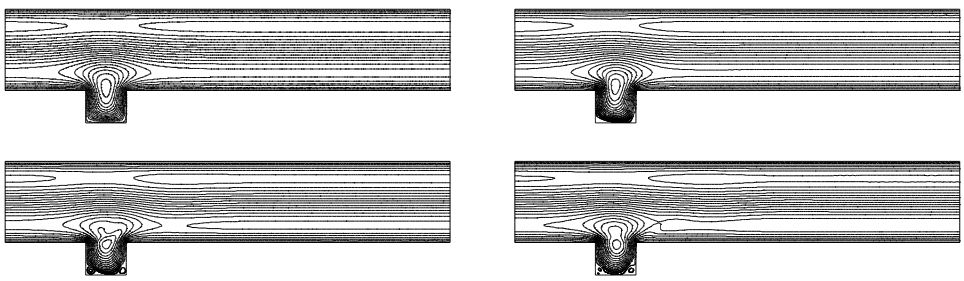

Рис. 9.4. Изолинии функции тока $\psi(x, y)$ при $t = 0{,}1;\ 0{,}5;\ 1{,}0;\ 1{,}7$

Вычисляемая для контроля разность расходов жидкости на входе и выходе каналов, т. е. величина Q, не превышала по модулю 0,006 на интервале $0 \leqslant t \leqslant 1{,}7$.

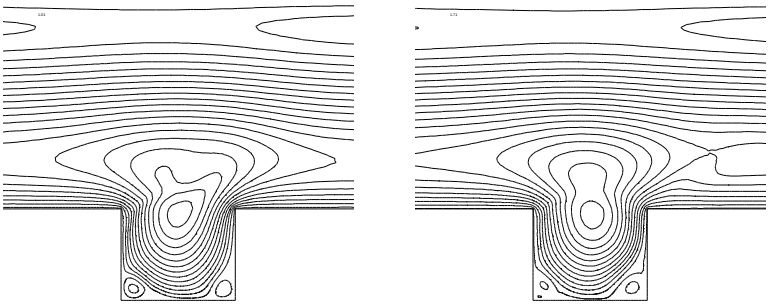

Рис. 9.5. Изолинии функции тока $\psi(x, y)$ при $t = 1{,}0;\ 1{,}7$ (увеличено)

9.2.3.2 Обтекание препятствия в канале

На рис. 9.6, 9.7 приведены результаты расчетов при обтекании жидкостью препятствия. Постановка задачи остается прежней — изменяется лишь область D. Все параметры взяты такими же, как и для предыдущего примера, за исключением координат следующих точек: $x_3 = 1$, $y_3 = 0{,}4$; $x_4 = 1{,}1$, $y_4 = 0{,}4$; $x_5 = 1{,}1$, $y_5 = 0$; $x_6 = 5{,}1$, $y_6 = 0$; $x_7 = 5{,}1$, $y_7 = 1$.

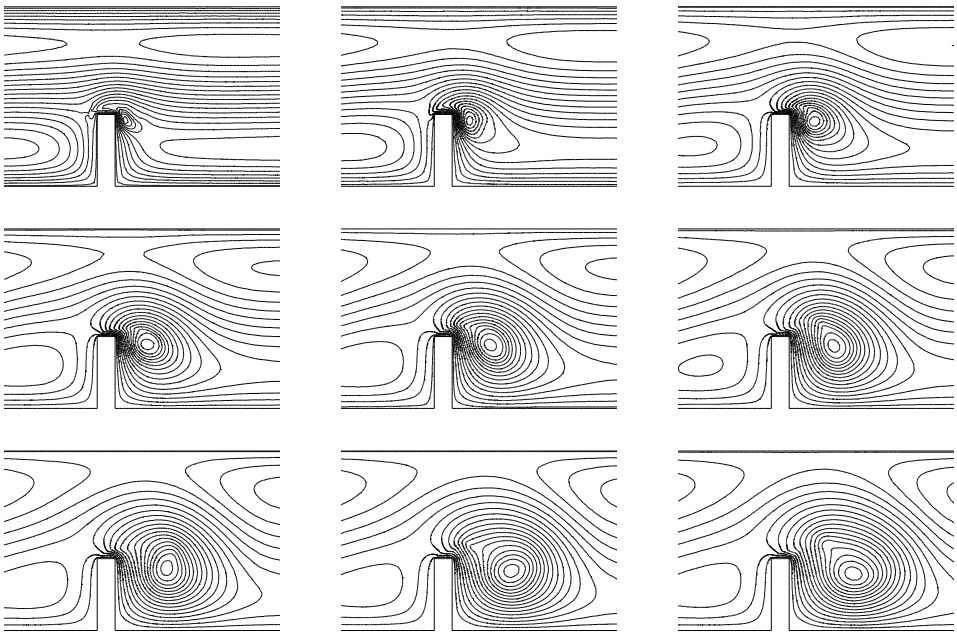

Рис. 9.6. Изолинии функции тока $\psi(x, y)$ при $t = 0{,}1k$, $k = 1, \ldots, 9$ (увеличено)

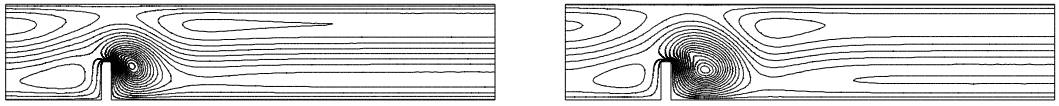

Рис. 9.7. Изолинии функции тока $\psi(x, y)$ при $t = 0{,}5;\ 0{,}9$

9.3 Обтекание тел жидкостью. Дорожка Кармана

Предыдущие программы практически без изменений можно использовать для решения задач об обтекании тел потоком жидкости. Для этого достаточно из области D удалить некоторую подобласть, соответствующую телу. Естественно, на границе твердого тела в случае его обтекания вязкой жидкостью следует поставить условия прилипания. Приведем два примера — обтекание прямоугольного тела и обтекание круглого тела.

9.3.1 Обтекание прямоугольного тела вязкой жидкостью

В случае прямоугольного тела фрагмент кода, задающий область D, имеет вид

```
w1=10;   h1=1;
x1=0;  y1=0;    x2=w1; y2=0;
x3=w1; y3=h1;   x4=0;  y4=h1;
x0=1.;    y0=0.5*h1;   r0=0.1;
w2=0.02;  h2=0.15;
x5=x0-w2; y5=y0-h2;    x6=x0+w2; y6=y0-h2;
x7=x0+w2; y7=y0+h2;    x8=x0-w2; y8=y0+h2;
// внешний прямоугольник
border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2;  };
border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3;  };
border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4;  };
border L4(t=0,1){ x=(1-t)*x4+t*x1;  y=(1-t)*y4+t*y1;  };
// внутренний прямоугольник
border L5(t=0,1){ x=(1-t)*x5+t*x6;  y=(1-t)*y5+t*y6;  };
border L6(t=0,1){ x=(1-t)*x6+t*x7;  y=(1-t)*y6+t*y7;  };
border L7(t=0,1){ x=(1-t)*x7+t*x8;  y=(1-t)*y7+t*y8;  };
border L8(t=0,1){ x=(1-t)*x8+t*x5;  y=(1-t)*y8+t*y5;  };
int n=2;
mesh Th = buildmesh(L1(10*n)+L2(5*n)+L3(10*n)+L4(5*n)
                  +L5(-10*n)+L6(-5*n)+L7(-10*n)+L8(-5*n));
```

9.3.2 Обтекание круглого тела вязкой жидкостью

Операторы, задающие область D в случае круглого тела, можно записать в виде

```
w1=3;   h1=1;
x1=0; y1=0;    x2=w1; y2=0;    x3=w1;  y3=h1;    x4=0; y4=h1;
x0=0.5;    y0=0.5;   r0=0.1;
border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2;  };
border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3;  };
border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4;  };
border L4(t=0,1){ x=(1-t)*x4+t*x1;  y=(1-t)*y4+t*y1;  };
border C0(t=0,2*pi){ x=x0+r0*cos(t);  y=y0+r0*sin(t); };
int n=2;
mesh Th = buildmesh(L1(10*n)+L2(5*n)+L3(10*n)+L4(5*n)+C0(-10*n));
```

9.3.3 Вычислительный эксперимент

На рис. 9.8 приведены результаты расчетов при обтекании прямоугольного тела потоком жидкости при $\mu = 0{,}0025$ с шагом $\tau = 0{,}01$. На входном участке канала задавался профиль скорости $u = 60y(1-y)$, $w = 0$.

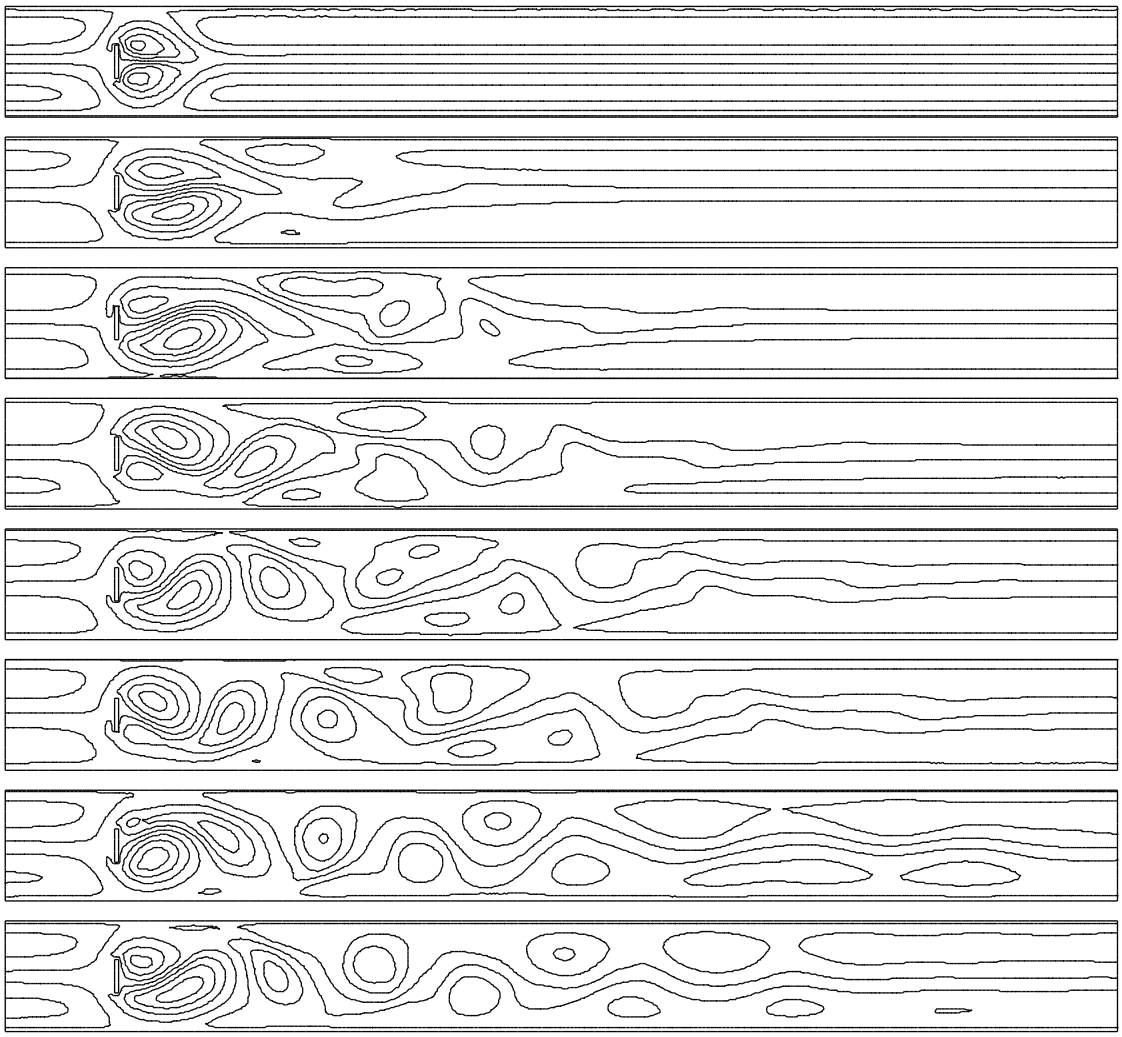

Рис. 9.8. Изолинии функции тока $\psi(x, y)$ в моменты времени $t = 0{,}06$; 0,31; 0,48; 0,55; 0,64; 0,68; 1,06; 7,39

Хорошо видно образование дорожки (цепочки) Ка́рмана — вихревого следа за телом. На рис. 9.8 при $t = 1{,}06$ и $t = 7{,}39$ отчетливо видна цепочка вихрей, образовавшаяся за препятствием — чередующиеся вихри, разделенные волнистыми линиями уровня функции тока. Рис. 9.8 при $t = 0{,}31$; 0,48; 0,55; 0,64; 0,68 демонстрируют процесс образования цепочки. От верхнего вихря ($t = 0{,}31$) отрывается вихрь ($t = 0{,}48$), который уносится потоком. Затем аналогичный вихрь отрывается от нижнего вихря ($t = 0{,}55$), затем вновь от верхнего ($t = 0{,}64$) и опять от нижнего ($t = 0{,}68$). Заметим, что частота образования вихрей за препятствием, как правило, пропорциональна скорости течения жидкости.

На рис. 9.9 приведены результаты расчетов при обтекании круглого тела потоком жидкости при $\mu = 0{,}0025$ с шагом $\tau = 0{,}01$. На входном участке канала задавался профиль скорости $u = 40y(1-y)$, $w = 0$. Хорошо виден

процесс образования дорожки Кармана вниз по потоку за круглым телом.

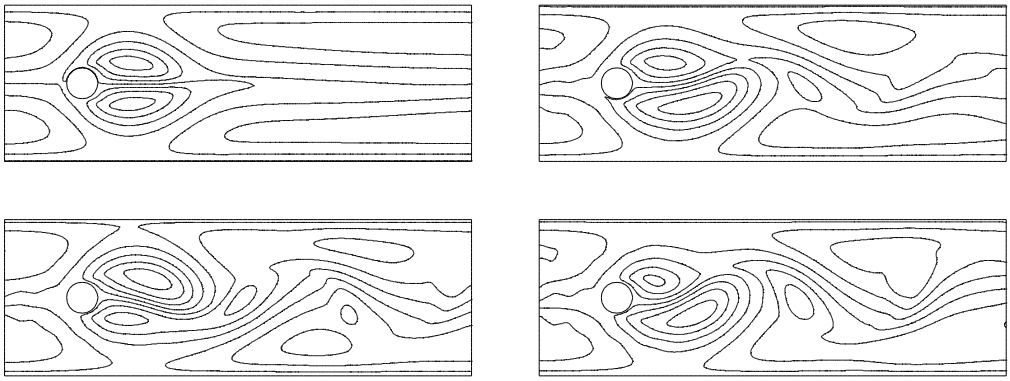

Рис. 9.9. Изолинии функции ψ при $t = 0{,}30;\ 0{,}63$ (верхний ряд) и при $t = 0{,}69;\ 0{,}79$

Глава 10

Перенос пассивной примеси

Задачи о переносе примесей возникают при исследовании процессов загрязнения рек, процессов разделения многокомпонентных смесей, процессов переноса лекарства в крови, при изучении движения микроорганизмов (которые также в некоторых случаях можно считать примесями).

10.1 Задача о движении пассивной примеси в жидкости

Под пассивной примесью понимается примесь, которая не взаимодействует с окружающей средой и ее перенос осуществляется за счет каких-либо внешних воздействий, например, течением жидкости или под действием внешнего электрического поля. Поведение такой не взаимодействующей с окружающей средой примеси в отсутствии процессов диффузии описывается уравнением переноса (см. гл. 5)

$$\frac{dc}{dt} = 0, \quad \frac{d}{dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla, \tag{10.1}$$

где $c(x, y, t)$ — концентрация примеси, $\boldsymbol{v} = (u, w)$ — скорость движения сплошной среды.

При наличии эффектов диффузии уравнение имеет вид

$$\frac{dc}{dt} = \varepsilon \Delta c, \quad (c_t + \boldsymbol{v} \cdot \nabla c = \varepsilon \Delta c), \tag{10.2}$$

здесь ε — коэффициент диффузии.

Собственно говоря, задача о переносе примеси уже рассматривалась в гл. 5 в случае, когда поле скорости $\boldsymbol{v} = \boldsymbol{v}(x, y, t)$ было задано в виде явной формулы. Однако на практике, чаще всего, поле скорости получается путем численного решения некоторой задачи (см., например, гл. 9). В такой ситуации наиболее естественно одновременно решать задачи об определении поля скорости и переносе примеси. Даже если задача об определении скорости $\boldsymbol{v}$ может быть решена независимо от задачи для определения c, то хранение информации о полученных численных значениях функции трех переменных $\boldsymbol{v}(x, y, t)$ практически неосуществимо.

10.2 Постановка задачи

Рассмотрим математическую модель распространения загрязнения при течении жидкости. Предполагаем, что через некоторую область D протекает жидкость (см. гл. 9) и имеется начальное распределение концентрации примеси (загрязнение). Система уравнений, описывающая процессы переноса, имеет вид

$$\boldsymbol{v}_t + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\nabla p + \mu \Delta \boldsymbol{v}, \quad \operatorname{div} \boldsymbol{v} = 0, \tag{10.3}$$

$$\frac{dc}{dt} = \varepsilon \Delta c. \tag{10.4}$$

Как уже говорилось, уравнение (10.3) для определения скорости $\boldsymbol{v}$ может решаться независимо от уравнения диффузии (10.4). Именно это и означает, что примесь пассивна, т. е. не оказывает никакого действия на течение жидкости.

Для определенности рассматриваем задачу о протекании, описанную в гл. 9. Уравнения (10.3) дополним краевыми и начальными условиями (9.5)–(9.10) (по поводу обозначений см. п. 9.2.1)

$$\boldsymbol{v}\big|_{\partial D \setminus (\Gamma_{\text{in}} \cup \Gamma_{\text{out}})} = 0, \quad \boldsymbol{v}\big|_{\Gamma_{\text{in}}} = \boldsymbol{v}_0, \quad p\big|_{\Gamma_{\text{out}}} = p_0, \quad \left.\frac{du}{dt}\right|_{\Gamma_{\text{out}}} = 0, \quad w\big|_{\Gamma_{\text{out}}} = 0, \tag{10.5}$$

$$\boldsymbol{v}\big|_{t=0} = \boldsymbol{h}(x, y). \tag{10.6}$$

Краевые условия для примеси, соответствующие непроницаемости границ, имеют вид (см. п. 3.3.2)

$$\left.\frac{\partial c}{\partial n}\right|_{\Gamma} = \boldsymbol{n} \cdot \nabla c = 0. \tag{10.7}$$

Здесь $\boldsymbol{n}$ — нормаль к границе.

Считаем также, что начальное распределение примеси $c_0(x, y)$ известно

$$c\big|_{t=0} = c_0(x, y). \tag{10.8}$$

10.3 Алгоритм решения задачи на языке FreeFem++

Обратим внимание, что процессы диффузии примеси и переноса тепла описываются фактически одними и теми же уравнениями. В частности, это означает, что для численного решения задачи (10.4), (10.7), (10.8) можно использовать фрагменты кодов, приведенные в гл. 4 и 5.

Схема построения решения на языке FreeFem++ мало отличается от схемы, предложенной для решения задачи протекания (см. с. 116) — к ней лишь добавлена часть для решения задачи диффузии, которую, как уже говорилось ранее, можно рассматривать отдельно после того, как определено поле скоростей течения жидкости.

При решении задач о переносе примесей следует иметь в виду, что по физическому смыслу концентрация примесей — величина неотрицательная. С математической точки зрения это означает, что постановка задачи должна обеспечивать неотрицательность решения. При вычислениях, строго говоря, для этих целей следовало бы использовать специальные, достаточно сложные, алгоритмы.

Одним из приемов, который можно использовать для сохранения свойства неотрицательности, является принудительная обработка на каждом временном шаге дискретного набора концентраций в узлах сетки. Попросту говоря, если шаг расчета достаточно мал и при вычислении решения в каких-либо узлах появляются отрицательные значения концентраций, то их можно полагать равными нулю. Именно это сделано в строках 73–74. Опасность использования такого приема наглядно демонстрируется следующим примером.

Пусть примесь находится в области с непроницаемыми стенками (условие (10.7)). В этом случае масса M примеси (интеграл от концентрации по области D), естественно, должна сохраняться (при выводе использовано уравнение $\operatorname{div} \boldsymbol{v} = 0$)

$$\iint\limits_D \frac{dc}{dt}\,dx\,dy = \frac{d}{dt}\iint\limits_D c\,dx\,dy = \varepsilon \iint\limits_D \Delta c\,dx\,dy = -\varepsilon \int\limits_{\partial D} \boldsymbol{n}\cdot\nabla \boldsymbol{v}\,ds = 0,$$

$$M = \iint\limits_D c\,dx\,dy, \quad \frac{dM}{dt} = 0.$$

Предположим, что имеется алгоритм, позволяющий автоматически выполнять закон сохранения массы (например, консервативные разностные схемы). Если в результате расчетов концентрации в каких-либо узлах сетки становятся отрицательными и полагаются затем равными нулю, то очевидно, закон сохранения массы будет нарушаться (масса увеличится).

Для того, чтобы закон сохранения массы при вычислениях не нарушался, можно использовать дополнительный перерасчет концентрации на каждом временном шаге. Пусть в некоторый момент t_m получено решение задачи, которое обозначим $c_*(x, y, t_m)$. Определим отклонение среднего значения вычисленной концентрации от истинного среднего значения

$$\frac{1}{\operatorname{mes} D}\iint\limits_D (c_*(x, y, t_m) - M)\,dx\,dy = \overline{c}, \tag{10.9}$$

где M — масса примеси в области, вычисленная, например, по начальному распределению концентрации примеси, $\operatorname{mes} D$ — площадь двумерной области, занимаемой примесью.

Концентрацию $c(x, y, t_m)$ в момент времени t_m вычисляем по формуле

$$c(x, y, t_m) = c_*(x, y, t_m) - \overline{c}. \tag{10.10}$$

Очевидно, что в этом случае масса будет сохраняться автоматически

$$\iint\limits_D c(x, y, t_m)\,dx\,dy = M. \tag{10.11}$$

Формулы (10.9)–(10.11) реализованы в алгоритме строками 32, 75, 76.

Конечно, таким алгоритмом можно пользоваться, когда a priori известно, что на каждом временном шаге примесь сосредоточена в какой-либо ограниченной части области D. В случае, когда примесь «размазана» по всей области, такой прием может привести к существенными ошибкам.

```
1  int m, n=2;
2  real x1,x2,x3,x4,x5,x6,x7,x8, y1,y2,y3,y4,y5,y6,y7,y8;
3  real w1,w2,w3,h1,h2;
4  w1=1;    w2=0.1;    w3=4;    h1=1;   h2=-0.4;
5  x1=0;          y1=0;         x2=w1;         y2=0;
6  x3=w1;         y3=-h2;       x4=w1+w2;      y4=-h2;
7  x5=w1+w2;      y5=0;         x6=w1+w2+w3;   y6=0;
8  x7=w1+w2+w3;   y7=h1;        x8=0;          y8=h1;
9  border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2; label=1; };
10 border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3; label=1; };
11 border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4; label=1; };
12 border L4(t=0,1){ x=(1-t)*x4+t*x5;  y=(1-t)*y4+t*y5; label=1; };
13 border L5(t=0,1){ x=(1-t)*x5+t*x6;  y=(1-t)*y5+t*y6; label=1; };
14 border L6(t=0,1){ x=(1-t)*x6+t*x7;  y=(1-t)*y6+t*y7; label=2; }; //out
15 border L7(t=0,1){ x=(1-t)*x7+t*x8;  y=(1-t)*y7+t*y8; label=4; };
16 border L8(t=0,1){ x=(1-t)*x8+t*x1;  y=(1-t)*y8+t*y1; label=3; }; //in
17 mesh Th= buildmesh(L1(10*n) + L2(5*n) + L3(10*n) + L4(5*n)
18                   + L5(10*n) + L6(10*n) + L7(30*n) + L8(10*n));
19 fespace Vh(Th,P2b);
20 real hinit=0.05;
21 Vh hh=hinit;
22 Th=adaptmesh(Th,hh,IsMetric=1,splitpbedge=1,nbvx=10000);
23 plot(Th,wait=1);
24 real mu = 0.0025, epsilon=0.002;    // Raynolds = 400 = Re = 1/mu
25 real dt = 0.01,  t=0;
26 Vh w, u, v, p, q, psi, c, cold;
27 u = 4*y*(1-y);
28 w = 0;   v = 0;   p = 0;   q = 0;  psi = 0;
29 real area= int2d(Th)(1.);
30 real Ampl=1, x01=0.5, y01=0.3, beta=50, r01=0.05;
31 c = Ampl*(1+tanh( -beta*((x-x01)^2 + (y-y01)^2 - r01^2) ));
32 real massac0= int2d(Th)(c);
33 Vh uold, wold, pold, f, g, h;
34 real meandiv, meanpq;
35 problem pb4c(c, v, init=m, solver=LU)
36         = int2d(Th)( c*v/dt + epsilon*(dx(c)*dx(v)+dy(c)*dy(v)) )
37           -int2d(Th)( h/dt * v )
38           + on(1,2,3,4, c = h);
39 problem pb4u(u, v, init=m, solver=LU)
40         = int2d(Th)( u*v/dt + mu*(dx(u)*dx(v)+dy(u)*dy(v)) )
41           - int2d(Th)( (f/dt-dx(p))*v )
42           + on(1,4,u = 0)
43           + on(2,u = f)
44           + on(3,u = 4*y*(1-y)) ;
45 problem pb4w(w, v, init=m, solver=LU)
```

```
46          = int2d(Th)(w*v/dt +mu*(dx(w)*dx(v)+dy(w)*dy(v)))
47            - int2d(Th)( (g/dt-dy(p))*v )
48            + on(1,2,3,4, w = 0);
49  problem pb4p(q, v, init=m, solver=LU)
50          = int2d(Th)( dx(q)*dx(v)+dy(q)*dy(v) )
51            + int2d(Th)( (dx(u)+ dy(w)-meandiv)*v/dt )
52            + on(2,q = 0) ;  //  meandiv = A*dt/S
53  problem pb4psi(psi, v,  solver=LU)
54          = int2d(Th)(dx(psi)*dx(v)+dy(psi)*dy(v))
55            - int2d(Th)((dx(w)- dy(u))*v)
56            +  on(1, psi=0)
57            +  on(4, psi=2/3);
58  for(m=0; m<10000; m++)
59  { t = t+dt;
60    uold = u;   wold = w;   pold = p;  cold = c;
61    f = convect([u,w],-dt,uold);
62    g = convect([u,w],-dt,wold);
63    h = convect([u,w],-dt,cold);
64    pb4u;
65    pb4w;
66    meandiv = int2d(Th)(dx(u)+dy(w))/area;
67    pb4p;
68    meanpq = int2d(Th)(pold + q)/area;
69    p = pold+q-meanpq;
70    u = u - dx(q)*dt;
71    w = w - dy(q)*dt;
72    pb4c;
73    for (int j=0; j<c[].n ; j++)
74      { if (c[][j]<0) c[][j]=0; }
75    real averC=(int2d(Th)(c)-massac0)/area;
76    c=c-averC;
77    pb4psi;
78    real Qin = int1d(Th,2)(u), Qout = int1d(Th,3)(u),massa=int2d(Th)(c);
79    cout <<"Q= "<<Qin-Qout << endl;
80    cout <<"massa= "<<massa << endl;
81    // plot(c, bw=1, bb=[[0.5,-0.1],[2,0.7]]);
82    plot(c, fill=0,cmm=+t,wait=0);
83  }
```

10.4 Вычислительный эксперимент

Результаты расчетов при $\mu = 0{,}0025$, $\varepsilon = 0{,}002$ с шагом $\tau = 0{,}01$ для задачи (10.3)–(10.8) приведены на рис. 10.1.

Начальное распределение концентрации задавалось «пятном» примеси с центром в точке ($x_0 = 0{,}5$, $y_0 = 0{,}3$) радиуса $r_0 = 0{,}05$ функцией

$$c_0(x, y) = 1 + \mathrm{th}\big(-\beta((x - x_0)^2 + (y - y_0)^2 - r_0^2)\big),$$

где $\beta = 50$ — параметр сглаживания.

При $\beta \to +\infty$ величина $c_0(x, y) \to 0$ для $((x - x_0)^2 + (y - y_0)^2) > r_0^2$.

Все остальные параметры выбирались такими же, как и для задачи п. 9.2.3.2, т. е. $x_1 = 0$, $y_1 = 0$; $x_2 = 1$, $y_2 = 0$; $x_3 = 1$, $y_3 = 0{,}4$; $x_4 = 1{,}1$, $y_4 = 0{,}4$; $x_5 = 1{,}1$, $y_5 = 0$; $x_6 = 5{,}1$, $y_6 = 0$; $x_7 = 5{,}1$, $y_7 = 1$, $x_8 = 0$, $y_8 = 1$.

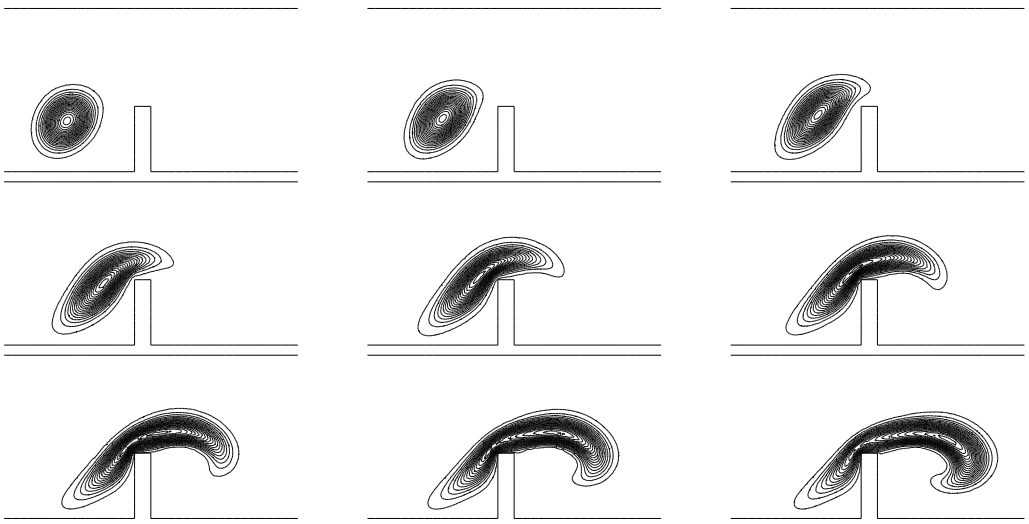

Рис. 10.1. Изолинии концентрации $c(x, y)$ при $t = 0{,}1k$, $k = 1, \ldots, 9$

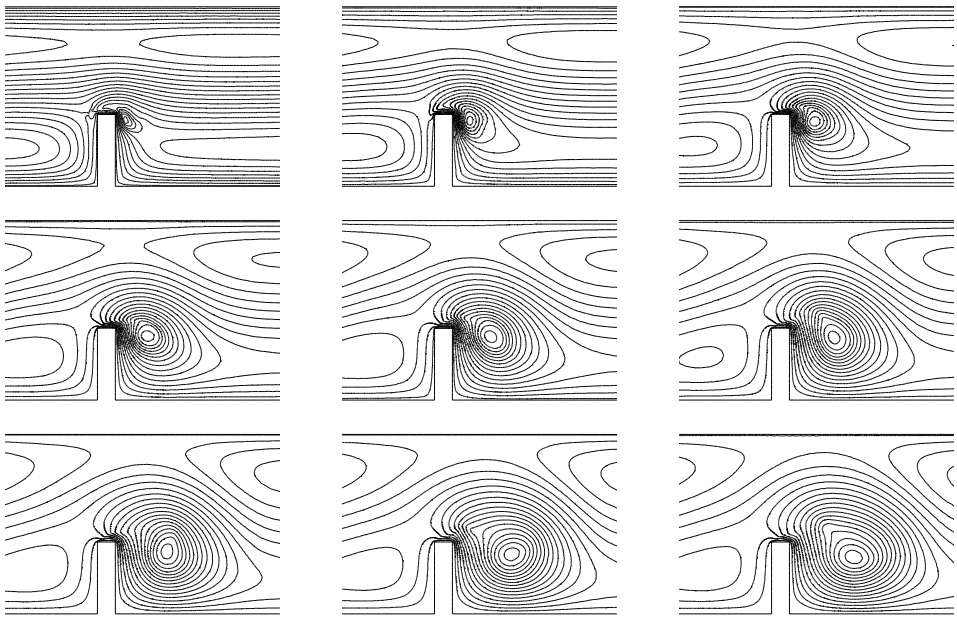

Рис. 10.2. Изолинии функции тока $\psi(x, y)$ при $t = 0{,}1k$, $k = 1, \ldots, 9$

Изолинии концентрации на рис. 10.1 приведены в те же самые моменты времени, что и изолинии функции тока на рис. 9.6, который для удобства сравнения дублируется на рис. 10.2. Именно в течении, показанном на рис. 9.6, 9.7, и переносится пассивная примесь. Хорошо видно, что примесь в основном следует линиям тока жидкости, слегка размываясь за счет диффузии.

Глава 11

Перенос примеси электрическим полем

Исследование процессов переноса примесей электрическим полем имеет важное прикладное значение. Это, в первую очередь, задачи электрофореза — метода разделения многокомпонентных смесей на отдельные компоненты при помощи внешнего электрического поля. Суть этого метода заключается в следующем: вещества многокомпонентной смеси, обладающие различной подвижностью в электрическом поле, помещаются в так называемую электрофоретическую камеру. Создаваемое в камере электрическое поле, двигая вещества с различной скоростью, разделяет смесь на отдельные компоненты. Заметим, что электрическое поле может быть не единственной причиной, которая заставляет двигаться элементы смеси. Возможно комбинированное воздействие на смесь течения жидкости, электрического, магнитного, гравитационного полей и др. (см. гл. 13).

С математической точки зрения задача о переносе примеси интересна тем, что в довольно широкой области параметров эффекты диффузии не существенны по сравнению с эффектами переноса электрическим полем. Важную роль в процессе переноса в этом случае играет зависимость характеристик смеси от концентрации ее компонент. Иными словами, процесс переноса происходит в среде с сильно нелинейными физико-химическими параметрами.

11.1 Основные уравнения

Наиболее простой является задача о переносе *одной* примеси в среде, электрическая проводимость которой зависит от концентрации. Процесс переноса заряженной примеси описывается уравнением

$$c_t + \operatorname{div} \boldsymbol{i} = 0, \quad \boldsymbol{i} = -\varepsilon \nabla c + \mu c \boldsymbol{E}, \tag{11.1}$$

где c — концентрация примеси, $\boldsymbol{i}$ — плотность потока примеси, $\boldsymbol{E}$ — напряженность электрического поля, μ — электрофоретическая подвижность примеси (скорость на единицу электрического поля), ε — коэффициент диффузии.

Считаем, что электрическое поле потенциально

$$\boldsymbol{E} = -\nabla \varphi, \tag{11.2}$$

где φ — электрический потенциал.

Плотность электрического тока $\boldsymbol{j}$ возьмем в виде (закон Ома)

$$\boldsymbol{j} = \sigma \boldsymbol{E}, \tag{11.3}$$

где σ — проводимость среды.

В предположении, что в целом смесь электронейтральна, уравнение неразрывности для плотности электрического тока имеет вид

$$\operatorname{div} \boldsymbol{j} = 0. \tag{11.4}$$

Проводимость смеси σ может быть величиной постоянной и в этом случае примесь пассивно переносится электрическим полем. Гораздо интереснее ситуация, когда проводимость среды зависит от концентрации $\sigma = \sigma(c)$. Это означает, что в процессе движения примеси проводимость (и электрическое поле) перераспределяется и в результате изменяется характер движения примеси. Иными словами, перенос примеси осуществляется в среде, свойства которой изменяются в процессе переноса. В этом случае говорят о так называемых ***электромиграционных эффектах***.

Простейший вид зависимости проводимости от концентрации имеет вид

$$\sigma = \sigma_0(1 + \alpha c), \tag{11.5}$$

где σ_0 — проводимость среды в отсутствии примеси, α — коэффициент влияния примеси на проводимость.

Заметим, что параметр α может быть как положительным, так и отрицательным. В случае $\alpha > 0$ проводимость примеси больше, чем проводимость «чистой» среды, а в случае $\alpha < 0$ проводимость примеси меньше проводимости «чистой» среды.

Уравнения (11.1)–(11.5) после соответствующих преобразований могут быть записаны лишь для неизвестных величин концентрации $c(x, y, t)$ и электрического потенциала $\varphi(x, y, t)$. Подставляя (11.3) в (11.4), с учетом (11.2) имеем

$$\sigma(c)\Delta\varphi + \nabla\varphi \cdot \nabla\sigma(c) = 0. \tag{11.6}$$

Это уравнение служит для определения потенциала, если концентрация c и зависимость $\sigma = \sigma(c)$ известны.

Выражая при помощи (11.3) $\boldsymbol{E}$ через $\boldsymbol{j}$ и подставляя в (11.1), выводим

$$c_t - \varepsilon\Delta c + \operatorname{div}\left(\frac{\mu c}{\sigma(c)}\boldsymbol{j}\right) = 0 \tag{11.7}$$

или, с учетом (11.4)

$$c_t - \varepsilon\Delta c + \boldsymbol{j} \cdot \nabla\left(\frac{\mu c}{\sigma(c)}\right) = 0. \tag{11.8}$$

Исключая при помощи (11.2), (11.3) величину $\boldsymbol{j}$, получим

$$c_t - \varepsilon\Delta c - \sigma(c)\nabla\varphi \cdot \nabla\left(\frac{\mu c}{\sigma(c)}\right) = 0. \tag{11.9}$$

Это уравнение можно записать в следующей форме

$$c_t - \varepsilon\Delta c + \boldsymbol{v}_e \cdot \nabla c = 0,$$

$$\boldsymbol{v}_e = -\sigma(c)\nabla\varphi \left(\frac{\mu c}{\sigma(c)}\right)'_c = -\mu\frac{\sigma(c) - c\sigma'(c)}{\sigma(c)}\nabla\varphi. \tag{11.10}$$

Здесь $\boldsymbol{v}_e$ — скорость переноса примеси в электрическом поле, аналогичная скорости течения жидкости в уравнениях (10.2).

В простейшем варианте, когда проводимость постоянна ($\alpha = 0$), имеем

$$\boldsymbol{v}_e = -\mu\nabla\varphi. \tag{11.11}$$

В случае (11.5), т. е. когда проводимость зависит от концентрации, скорость переноса будет

$$\boldsymbol{v}_e = -\frac{\mu}{1 + \alpha c}\nabla\varphi. \tag{11.12}$$

Обратим внимание, что $\boldsymbol{v}_e$ ***не зависит*** от проводимости «чистой» среды σ_0. Более того, уравнение для определения потенциала (11.6) также ***не зависит*** от σ_0. В частности, это означает, что для рассматриваемой математической модели фон («чистая» среда), на котором осуществляется процесс переноса, не существенен — важен лишь характер влияния примеси на скорость переноса.

Формула (11.12) позволяет дать предварительный качественный анализ поведения примеси при ее переносе электрическим полем. Предположим, что $\nabla\varphi$ слабо меняется (это заведомо неверно, т. к. φ зависит от c, см. (11.6)). Пусть $\alpha > 0$, тогда с ростом концентрации c величина скорости будет уменьшаться. Если имеется, например, область, вне которой концентрация обращается в нуль («пятно» примеси), то при $\alpha > 0$ участки «пятна» с высокой концентрацией движутся медленнее, чем участки «пятна» с более низкой концентрацией.

Рассмотрим окрестность «границы» пятна. Примесь, пытаясь пересечь границу (например, за счет диффузии) и имея вне области пятна почти нулевую концентрацию, будет двигаться быстрее, чем примесь внутри пятна. Это означает, что граница пятна будет с течением времени «размываться». Возникнет так называемая волна разрежения. Заметим, что размытие пятна происходит за счет нелинейного характера зависимости скорости переноса от концентрации, роль диффузии сводится лишь к инициализации процесса размытия.

Напротив, при $\alpha < 0$ наблюдается обратный эффект. Примесь, пытаясь пересечь границу, замедляет свое движение и граница пятна становится более «резкой», т. к. примесь внутри пятна движется быстрее, чем вне пятна. Возникает так называемая ударная волна.

11.2 Постановка задачи

Рассмотрим процесс переноса примеси электрическим полем в некоторой двумерной области D, на части границы которой задан электрический потенциал, а оставшаяся часть границы области изолирована.

Запишем систему уравнений для нахождения c и φ, полученную в п. 11.1

$$c_t - \varepsilon\Delta c + \boldsymbol{v}_e \cdot \nabla c = 0, \quad \boldsymbol{v}_e = -\mu\frac{\sigma(c) - c\sigma'(c)}{\sigma(c)}\nabla\varphi, \tag{11.13}$$

$$\sigma(c)\Delta\varphi + \nabla\varphi \cdot \nabla\sigma(c) = 0, \quad (\operatorname{div}(\sigma(c)\nabla\varphi) = 0). \tag{11.14}$$

Дополним уравнение (11.13) краевыми условиями, соответствующими непроницаемости границ области D для примеси

$$(\boldsymbol{i}\cdot\boldsymbol{n})\big|_{\partial D} = 0, \quad \left(\varepsilon\frac{\partial c}{\partial n} + \mu c\boldsymbol{n}\cdot\nabla\varphi\right)_{\partial D} = 0. \tag{11.15}$$

Считаем, что часть границы Γ_{i} области D изолирована, а на оставшейся части границы Γ_φ задан электрический потенциал. Этому соответствуют следующие краевые условия

$$(\boldsymbol{n}\cdot\nabla\varphi)\big|_{\Gamma_{\mathrm{i}}} = \left(\frac{\partial\varphi}{\partial n}\right)_{\Gamma_{\mathrm{i}}} = 0, \quad \varphi\big|_{\Gamma_\varphi} = \varphi_0(x,y)\big|_{\Gamma_\varphi}, \quad \partial D = \Gamma_{\mathrm{i}} \cup \Gamma_\varphi, \tag{11.16}$$

где $\varphi_0(x,y)$ — известная функция на границе.

Условие (11.15) на границе Γ_{i} с учетом (11.16) принимает вид

$$\left(\frac{\partial c}{\partial n}\right)_{\Gamma_{\mathrm{i}}} = 0. \tag{11.17}$$

Кроме этого, для (11.13) задаем начальное условие

$$c(x,y,0) = c_0(x,y), \tag{11.18}$$

где $c_0(x,y)$ — известная функция.

11.3 Алгоритм решения

Слабая формулировка задачи (11.13), (11.15), т. е. задачи диффузии с переносом, уже много раз была получена (см., например, (4.9)). Приведем лишь слабую формулировку задачи (11.14), (11.16) для определения электрического потенциала

$$\iint\limits_D \theta\operatorname{div}(\sigma\nabla\varphi)\,dx\,dy = -\iint\limits_D \sigma\nabla\theta\cdot\nabla\varphi\,dx\,dy + \int\limits_{\partial D}\sigma\theta\frac{\partial\varphi}{\partial n}\,ds = 0. \tag{11.19}$$

Как обычно, обозначим, $c^m = c(x,y,t_m)$, $\varphi^m = \varphi(x,y,t_m)$. Используя результаты п. 5.5 для уравнений диффузии-переноса, запишем неявную схему аппроксимации по времени

$$\frac{c^{m+1}(\boldsymbol{x}) - c^m(\boldsymbol{X}^m(\boldsymbol{x}))}{\tau} - \mu\Delta c^{m+1}(\boldsymbol{x}) = 0. \tag{11.20}$$

Запись на языке FreeFem++ вариационной формы задачи будет такой же, как и в п. 5.5 (в случае краевых условий $c = 0$ на границе)

```
problem DifConv(c,vv) =
              int2d(Th)( c*vv + delta*dt*(dx(c)*dx(vv)+dy(c)*dy(vv)) )
            - int2d(Th)( vv*convect([v1,v2],-dt,cOld) )
            + on(GammaB, GammaL, GammaR, GammaT, c=0);
```

Здесь `cOld` $= c^m$, `c` $= c^{m+1}$, `[v1,v2]` — скорость переноса, `vv` — пробная функция.

Величины `v1`, `v2` вычисляем с использованием формулы (11.10)

$$\texttt{v1} = -\mu\frac{\sigma(c^m) - c^m\sigma'(c^m)}{\sigma(c^m)}\varphi_x^m, \quad \texttt{v2} = -\mu\frac{\sigma(c^m) - c^m\sigma'(c^m)}{\sigma(c^m)}\varphi_y^m. \qquad (11.21)$$

Код, соответствующий (11.19) и краевым условиям (11.16), будет

```
problem pbPhi(phi,vv) =
                   int2d(Th)( sigma*(dx(phi)*dx(vv)+dy(phi)*dy(vv)) )
                 + on(GammaPhi, phi=phi0);
```

11.4 Реализация алгоритма на языке FreeFem++

```
1   int m, n=2;
2   real x1,x2,x3,x4,x5,x6,x7,x8,y1,y2,y3,y4,y5,y6,y7,y8,w1,w2,w3,h1,h2;
3   w1=1;    w2=1;    w3=3;    h1=1;   h2=1.5;
4   x1=0; y1=0; x2=w1; y2=0; x3=w1; y3=-h2; x4=w1+w2; y4=-h2;
5   x5=w1+w2; y5=0; x6=w1+w2+w3; y6=0; x7=w1+w2+w3; y7=h1; x8=0; y8=h1;
6   border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2; };
7   border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3; };
8   border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4; };
9   border L4(t=0,1){ x=(1-t)*x4+t*x5;  y=(1-t)*y4+t*y5; };
10  border L5(t=0,1){ x=(1-t)*x5+t*x6;  y=(1-t)*y5+t*y6; };
11  border L6(t=0,1){ x=(1-t)*x6+t*x7;  y=(1-t)*y6+t*y7; };   // out
12  border L7(t=0,1){ x=(1-t)*x7+t*x8;  y=(1-t)*y7+t*y8; };
13  border L8(t=0,1){ x=(1-t)*x8+t*x1;  y=(1-t)*y8+t*y1; };   // in
14  mesh Th= buildmesh(L1(10*n)+L2(5*n)+L3(10*n)+L4(5*n)
15                     +L5(10*n)+L6(10*n)+L7(30*n)+L8(10*n));
16  fespace Vh(Th,P2b);
17  real hinit=0.05;
18  Vh hh=hinit;
19  Th=adaptmesh(Th,hh,IsMetric=1,splitpbedge=1,nbvx=10000);
20  real epsilon=0.002, phi8=10, phi6=0, phi3=3, mu0=0.51, alpha=-0.49;
21  real dt = 0.01,  t=0,  area0=int2d(Th)(1.0);
22  Vh ue, we, v, phi, c, cold, h, sigma;
23  real Ampl=1, x01=0.9, y01=0.3, beta=50, r01=0.05;
24  c = Ampl*(1+tanh( -beta*((x-x01)^2 + (y-y01)^2 - r01^2) ));
25  real massac0= int2d(Th)(c);
26  problem pb4c(c, v, init=m, solver=LU)
27          = int2d(Th)( c*v/dt + epsilon*(dx(c)*dx(v)+dy(c)*dy(v)) )
28            - int2d(Th)( h/dt * v )
```

```
            + on(L1,L2,L3,L4,L5,L6,L7,L8, c = h);
problem pbPhi(phi, v,  solver=LU)
        = int2d(Th)( sigma*(dx(phi)*dx(v)+dy(phi)*dy(v)) )
          + on(L8, phi=phi8) + on(L3, phi=phi3) + on(L6, phi=phi6) ;
sigma=1+alpha*c;
pbPhi;
for(m=0; m<10000; m++)
{ t = t+dt;
  cold = c;               sigma=1+alpha*cold;
  ue=-mu0/sigma*dx(phi);  we=-mu0/sigma*dy(phi);
  h = convect([ue,we],-dt,cold);
  pb4c;
  for (int j=0; j<c[].n ; j++) { if (c[][j]<0) c[][j]=0; }
  real averC=(int2d(Th)(c)-massac0)/area0;       c=c-averC;
  sigma=1+alpha*c;
  pbPhi;
  adaptmesh(Th,c);
  real massa=int2d(Th)(c) ;
  cout <<"massa= "<<massa << endl;
  plot(c,cmm=+t);      //   plot(c,cmm=+t,bb=[[0.5,-0.2],[1.5,0.6]]);
}
```

11.5 Бездиффузионная модель переноса

В пространственно одномерном случае задача (11.13)–(11.18) при отсутствии диффузии ($\varepsilon = 0$) существенно упрощается и решение можно получить аналитическими методами (см. [17, 18, 24–26]). Правомерность замены исходной модели бездиффузионным приближением вытекает из того факта, что наиболее существенную роль в искажении формы первоначального распределения концентрации при малой диффузии играют нелинейные эффекты — зависимость скорости переноса от концентрации. Особенно это заметно в интересном для практики случае, когда начальное распределение примеси близко к «ступенчатому» (кусочно-разрывному) распределению по пространству. Заметим, что рассмотрение бездиффузионной одномерной модели, помимо всего прочего, важно для интерпретации результатов расчета исходной задачи.

11.5.1 Условия на разрыве

Рассмотрим одномерный вариант модели, описанной в п. 11.1, в случае, когда $\varepsilon = 0$. Уравнения (11.1)–(11.4) примут вид

$$c_t + i_x = 0, \quad i = \mu c E, \quad j = \sigma E, \quad j_x = 0. \tag{11.22}$$

Эти уравнения представляют собой систему уравнений в частных производных первого порядка и их решение можно разыскивать в классе кусочно-разрывных функций. В этом случае (11.22) следует дополнить условиями Рэнкина–Гюгонио на разрывах, которые вытекают из законов сохранения массы и заряда (см., например, [24–26])

$$V[\,c\,] = [\,i\,], \quad [\,j\,] = 0, \quad V = \frac{dx}{dt}. \tag{11.23}$$

Здесь $x = x(t)$ — уравнение линии, на которой решение, например, функция $c(x,t)$, может иметь разрыв первого рода: значения функции справа $c(x(t)+0,t)$ и слева $c(x(t)-0,t)$ от линии разрыва $x = x(t)$ не совпадают, т. е. $c(x(t)+0,t) \neq c(x(t)-0,t)$; V — скорость движения линии разрыва; символ [...] означает величину разрыва функции, например, $[\,c\,] = c(x(t)+0,t) - c(x(t)-0,t)$.

11.5.2 Задача о распаде начального разрыва

Ограничимся рассмотрением некоторой частной задачи, более простой по сравнению с (11.13)–(11.18). Из последнего уравнения (11.22) следует, что $j = j(t)$. Полагаем, что $j = \text{const}$ задано, и решение (11.22), (11.23) ищем на всей прямой $-\infty < x < +\infty$. В этом случае условия для электрического потенциала φ можно не задавать и величина E (и φ) полностью исключается из уравнений (11.22), (11.23).

Пусть зависимость проводимости σ от концентрации c имеет вид (11.5). Тогда, после несложных преобразований, аналогичных п. 11.1, из (11.22), (11.23) получим

$$c_t + \left(\frac{\mu_0 c}{1+\alpha c}\right)_x = 0, \quad -\infty < x < +\infty; \quad V[\,c\,] = \left[\frac{\mu_0 c}{1+\alpha c}\right]; \quad \mu_0 = \frac{j\mu}{\sigma_0}. \tag{11.24}$$

Начальное условие для $c(x,t)$ задаем в виде кусочно-постоянной функции

$$c(x,0) = \begin{cases} c_0, & x \in [x_1, x_2], \\ 0, & x \notin [x_1, x_2], \end{cases} \tag{11.25}$$

где $c_0 = \text{const}$, x_1, x_2 — заданы.

Задача (11.24), (11.25) называется задачей о распаде начального разрыва или задачей Римана (см., например, [17, 18]). Подробное решение данной конкретной задачи хорошо известно и приведено, в частности, в [24–26]. Далее ограничимся лишь записью решения в начальные моменты времени.

Для дальнейших целей удобно записывать решения не для концентрации $c(x,t)$, а для проводимости $\sigma(x,t) = \sigma_0(1 + \alpha c(x,t))$ (см. рис. 11.1).

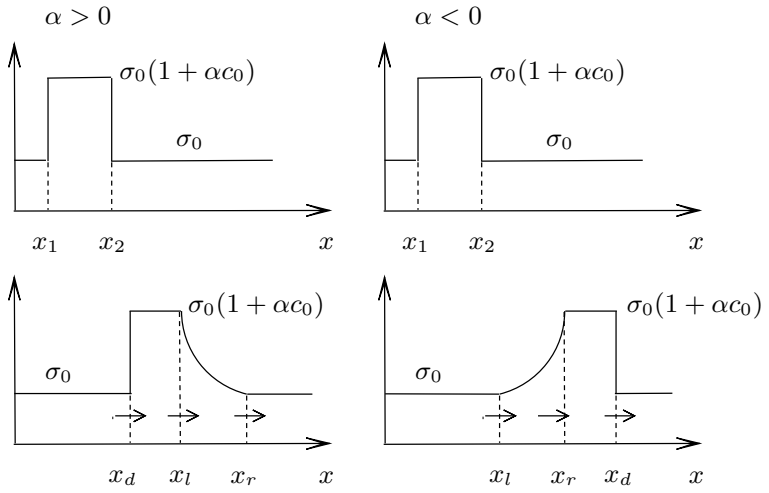

Рис. 11.1. Схема распада начального разрыва

При $\alpha > 0$ имеем

$$\sigma(x,t) = \begin{cases} \sigma_0, & -\infty < x < x_d(t) = x_1 + \dfrac{\mu_0 t}{1+\alpha c_0}, \\ \sigma_0(1+\alpha c_0), & x_d(t) < x < x_l(t) = x_2 + \dfrac{\mu_0 t}{(1+\alpha c_0)^2}, \\ \sigma_0\left(\dfrac{\mu_0 t}{x - x_2}\right)^{1/2}, & x_l(t) < x < x_r(t) = x_2 \mu_0 t, \\ \sigma_0, & x_r(t) < x < +\infty. \end{cases} \tag{11.26}$$

При $\alpha < 0$ имеем

$$\sigma(x,t) = \begin{cases} \sigma_0, & -\infty < x < x_l(t) = x_1 + \mu_0 t, \\ \sigma_0 \left(\dfrac{\mu_0 t}{x - x_1} \right)^{1/2}, & x_l(t) < x < x_r(t) = x_1 + \dfrac{\mu_0 t}{(1+\alpha c_0)^2}, \\ \sigma_0(1+\alpha c_0), & x_r(t) < x < x_d(t) = x_2 + \dfrac{\mu_0 t}{1+\alpha c_0}, \\ \sigma_0, & x_d(t) < x < +\infty. \end{cases} \tag{11.27}$$

Коротко опишем схему распада начального разрыва для случая $\alpha > 0$ (рис. 11.2 при $\alpha > 0$). Начальный кусочно-постоянный разрыв в точке $x = x_1$ при $t > 0$ начинает двигаться вправо по закону $x = x_d(t)$. Это ***ударная волна*** — сильный разрыв решения. В точке $x = x_2$ при $t > 0$ образуется ***волна разрежения*** — из первоначального сильного разрыва возникает два слабых разрыва (разрывы производных), которые, соответственно, называются левый $x_l(t)$ и правый $x_r(t)$ ***фронт волны разрежения***. Фронты волны разрежения также движутся вправо и скорость движения разрыва $x_r(t)$ больше скорости движения разрыва $x_l(t)$. В частности, это означает, что ширина области $[x_l(t), x_r(t)]$ растет со временем (отсюда и название — волна разрежения).

Приведенная на рис. 11.1 картина будет существовать до момента времени $t = t_i$. Дело в том, что скорость движения ударной волны x_d больше скорости движения левого фронта волны разрежения $x_l(t)$. В момент $t = t_i$ ударная волна догонит фронт волны разрежения ($x_d(t_i) = x_l(t_i)$) и произойдет взаимодействие разрывов. Дальнейшая эволюция также может быть описана аналитически (см., например, [24–26]). Опуская подробности, укажем, что в конечном итоге профиль распределения концентрации примет «треугольную» форму (см. рис. 11.2).

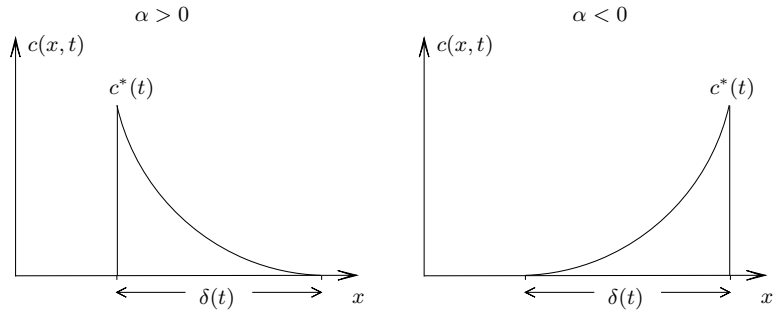

Рис. 11.2. Финальная стадия процесса

При $t \to +\infty$ высота $c^*(t)$ и ширина $\delta(t)$ треугольника определяются соотношениями (независимо от знака α)

$$c^*(t) \sim \left(\frac{M}{|\alpha|\mu_0 t} \right)^{1/2}, \quad \delta(t) \sim 2\left(|\alpha|\mu_0 M t\right)^{1/2}, \quad \frac{1}{2} c^*(t)\delta(t) \sim M = c_0(x_2 - x_1), \tag{11.28}$$

где M — масса примеси.

Обратим внимание, что «размазывание» концентрации примеси происходит в отсутствии процессов диффузии(!)

$$c^*(t) = \mathcal{O}(t^{-1/2}), \quad \delta(t) = \mathcal{O}(t^{1/2}), \quad t \to +\infty.$$

Этот типично нелинейный эффект называется ***электромиграционным размытием*** и обусловлен зависимостью скорости переноса примеси от концентрации.

При $\alpha < 0$ будет наблюдаться похожая картина эволюции. Однако в этом случае волна разрежения будет возникать в точке $x = x_1$, а ударная волна будет начинать двигаться от точки $x = x_2$ (см. рис. 11.2 при $\alpha < 0$).

11.6 Вычислительный эксперимент

11.6.1 Случай $\alpha > 0$ (увеличение σ с ростом c)

Результаты расчетов для задачи (11.5), (11.13)–(11.18) при $\varepsilon = 0{,}002$, $\mu = 0{,}51$, $\alpha = 2$ с шагом $\tau = 0{,}01$ приведены на рис. 11.3.

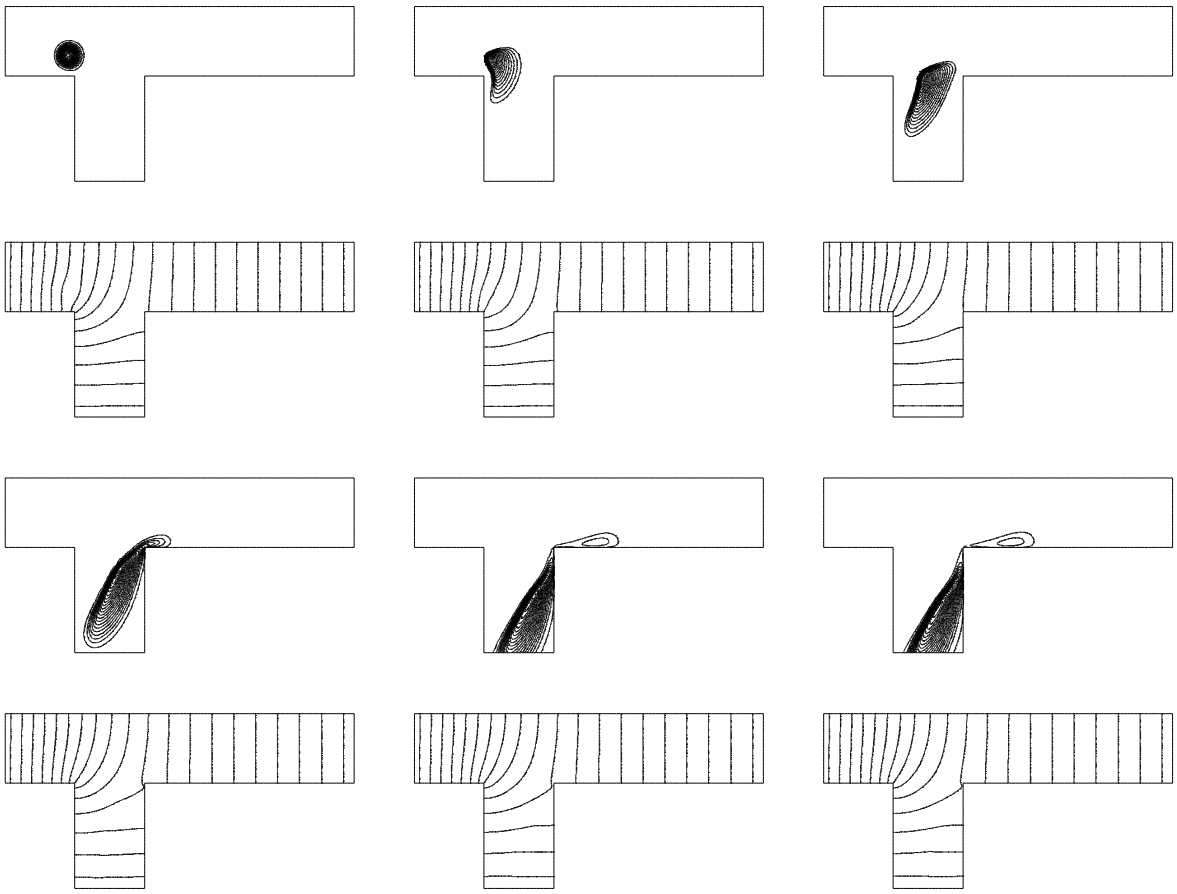

Рис. 11.3. Изолинии концентрации $c(x, y, t)$ и потенциала $\varphi(x, y, t)$ в случае $\alpha = 2$ в моменты времени $t = 0{,}0;\ 0{,}4;\ 1{,}1;\ 1{,}8;\ 2{,}5;\ 2{,}6$

Как и в случае, рассмотренном в п. 9.2, область имеет Т-образный вид (см. рис. 9.3). Границы области задаются отрезками линий со следующими координатами концов отрезка: $x_1 = 0$, $y_1 = 0$; $x_2 = 1$, $y_2 = 0$; $x_3 = 1$, $y_3 = -1{,}5$; $x_4 = 2$, $y_4 = -1{,}5$; $x_5 = 2$, $y_5 = 0$; $x_6 = 5$, $y_6 = 0$; $x_7 = 5$, $y_7 = 1$; $x_8 = 0$, $y_8 = 1$.

Отрезок, соединяющий точки с координатами (x_k, y_k) и (x_{k+1}, y_{k+1}), обозначен L_k. Считаем, что $\Gamma_{\mathrm{i}} = L_1 \cup L_2 \cup L_4 \cup L_5 \cup L_7$, $\Gamma_\varphi = L_3 \cup L_6 \cup L_8$. На участках L_3, L_6, L_8 заданы, соответственно, следующие значения потенциала: $\varphi_3 = 3$, $\varphi_6 = 0$, $\varphi_8 = 10$.

Начальное распределение концентрации задавалось в виде «пятна» радиуса $r_0 = 0{,}05$ с центром в точке $x_0 = 0{,}9$, $y_0 = 0{,}3$ при помощи функции

$$c_0(x, y) = 1 + \mathrm{th}\big(-\beta((x - x_0)^2 + (y - y_0)^2 - r_0^2)\big),$$

где $\beta = 50$ — параметр сглаживания.

Обратим внимание на то, что примесь влияет на распределение потенциала в области — в местах сосредоточения примеси изолинии потенциала сгущаются. Заметим, что потенциал на границах задан таким образом, что примесь должна двигаться вправо и вниз. На рисунке хорошо видно как примесь почти вся уходит в нижнюю область канала, медленно перемещаясь вправо. В момент времени $t \approx 2{,}6$ происходит отрыв части примеси, которая в дальнейшем движется в правой части канала, в то время как основная часть остается в нижней части канала.

11.6.2 Влияние параметра α на перенос примеси

На рис. 11.4–11.6 приведены результаты расчетов для различных α. Все остальные параметры выбраны такими же, как в п. 11.6.1. Увеличенное изображение показано в прямоугольнике с координатами левого нижнего угла $(0{,}5; -0{,}2)$ и правого верхнего угла $(1{,}5; 0{,}6)$.

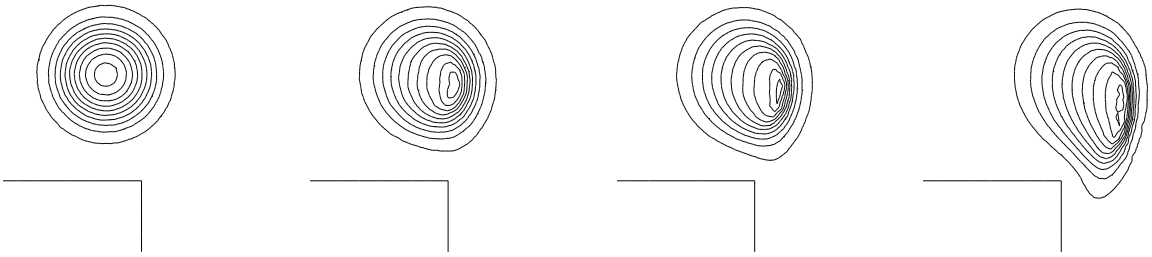

Рис. 11.4. Изолинии концентрации $c(x, y, t)$ при $t = 0{,}00;\ 0{,}03;\ 0{,}05;\ 0{,}11;\ \alpha = -0{,}49$

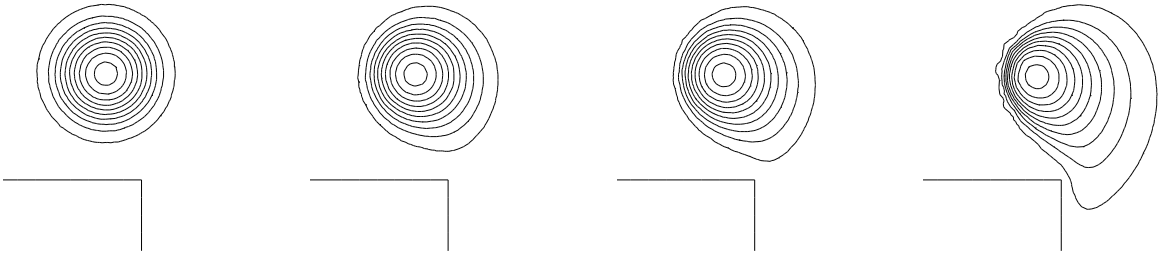

Рис. 11.5. Изолинии концентрации $c(x, y, t)$ при $t = 0{,}00;\ 0{,}03;\ 0{,}05;\ 0{,}11;\ \alpha = 2{,}00$

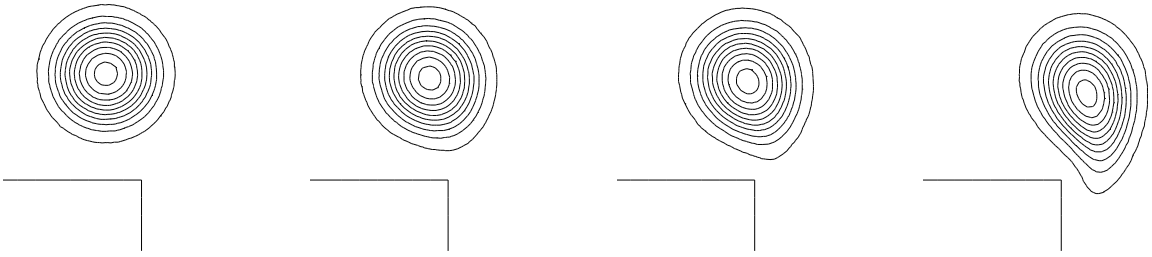

Рис. 11.6. Изолинии концентрации $c(x, y, t)$ при $t = 0{,}00;\ 0{,}03;\ 0{,}05;\ 0{,}11;\ \alpha = 0$

Хорошо видно, что по пути следования примеси (вправо и вниз) при $\alpha < 0$ (рис. 11.4) на переднем фронте образуется ударная волна (сгущение

изолиний), а на заднем фронте — волна разрежения (разрежение изолиний). При $\alpha > 0$ (рис. 11.5) на переднем фронте образуется волна разрежения, а задний фронт является ударной волной. Это приводит к достаточно заметному искажению изолиний. Результаты расчетов находятся в хорошем соответствии с бездиффузионной теорией, изложенной для пространственно одномерного случая в п. 11.5, в частности, см. рис. 11.1, 11.2.

При $\alpha = 0$ (рис. 11.6) нелинейные эффекты отсутствуют и искажения изолиний происходят в результате диффузии и неравномерности распределения электрического потенциала.

11.7 Метод решения задачи, не использующий оператор `convect`

Задачу (11.13)–(11.18) не обязательно решать, используя метод характеристик (ключевое слово `convect`). Запишем слабую (вариационную) формулировку уравнений (11.1). Явно-неявная аппроксимация имеет вид

$$\frac{c^{m+1} - c^m}{\tau} - \varepsilon \Delta c^{m+1} - \operatorname{div}(\mu c^{m+1} \nabla \varphi^m) = 0. \tag{11.29}$$

Умножая на тестовую функцию θ и интегрируя, получим

$$\iint\limits_D \theta \frac{c^{m+1} - c^m}{\tau}\, dx\, dy + \iint\limits_D \varepsilon \nabla\theta \cdot \nabla c^{m+1}\, dx\, dy - \int\limits_{\partial D} \varepsilon\theta \frac{\partial c^{m+1}}{\partial n}\, ds + \\ + \iint\limits_D \mu c^{m+1} \nabla\theta \cdot \nabla\varphi^m\, dx\, dy - \int\limits_{\partial D} \mu\theta c^{m+1} \nabla\varphi^m \cdot \boldsymbol{n}\, ds = 0. \tag{11.30}$$

В силу (11.15) интегралы по границе исчезают и имеем

$$\iint\limits_D \left(\theta \frac{c^{m+1} - c^m}{\tau} + \varepsilon \nabla\theta \cdot \nabla c^{m+1} + \mu c^{m+1} \nabla\theta \cdot \nabla\varphi^m \right) dx\, dy = 0. \tag{11.31}$$

Слабую формулировку задачи запишем в виде (`v` $= \theta$, `cold` $= c^m$)

```
26 problem pb4c(c, v, init=m, solver=LU)
27             = int2d(Th)( c*v/dt + epsilon*(dx(c)*dx(v)+dy(c)*dy(v)) )
28               + int2d(Th)( mu0*c*(dx(phi)*dx(v)+dy(phi)*dy(v)) )
29               - int2d(Th)( cold/dt * v );
```

В коде предыдущей программы (см. с. 132) достаточно заменить строки 26–29, удалить `ue`, `we` и `h` в строке 22 и удалить строку 39.

На рис. 11.7 приведены результаты расчетов по предлагаемому алгоритму (верхний ряд) и по алгоритму п. 11.3. Расчеты проведены для значений параметров, таких же, как в п. 11.6.1: $\varepsilon = 0{,}002$, $\mu = 0{,}51$, $\alpha = 2$, $\tau = 0{,}01$.

Начальное распределение концентрации задано в виде «пятна» радиуса $r_0 = 0{,}05$ с центром в точке $x_0 = 0{,}9$, $y_0 = 0{,}3$, $\beta = 50$

$$c_0(x, y) = 1 + \operatorname{th}\bigl(-\beta((x - x_0)^2 + (y - y_0)^2 - r_0^2)\bigr),$$

Область Т-образной формы задается отрезками со следующими координатами концов: $x_1 = 0$, $y_1 = 0$; $x_2 = 1$, $y_2 = 0$; $x_3 = 1$, $y_3 = -1{,}5$; $x_4 = 2$, $y_4 = -1{,}5$; $x_5 = 2$, $y_5 = 0$; $x_6 = 5$, $y_6 = 0$; $x_7 = 5$, $y_7 = 1$; $x_8 = 0$, $y_8 = 1$.

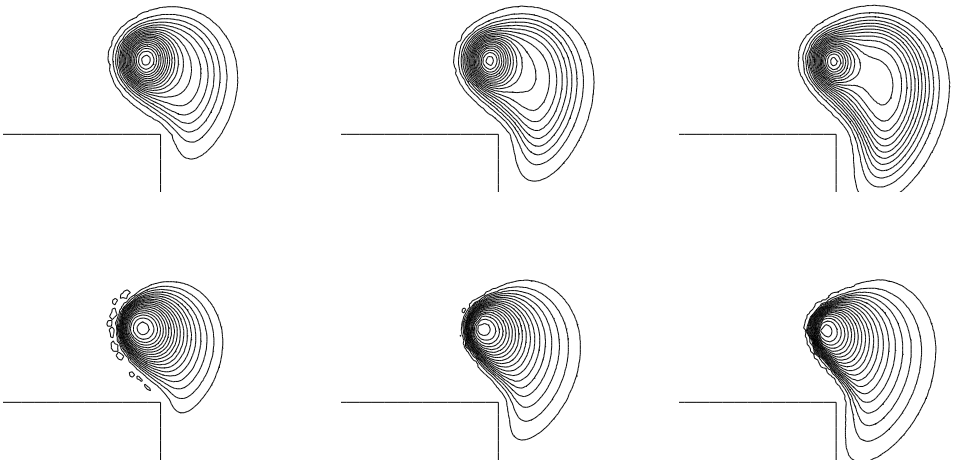

Рис. 11.7. Сравнение различных методов решения (нижний ряд — с использованием метода характеристик). Изолинии $c(x, y, t)$ при $t = 0{,}10;\ 0{,}15;\ 0{,}20;\ \alpha = 2$

Видно, что результаты различаются уже на первых шагах по времени, причем наиболее существенно на заднем фронте пятна — образование ударной волны на рисунках верхнего ряда слабо заметно.

Чтобы наглядно представить себе различия двух методов расчета движения примеси, приведем результаты вычислений для прямоугольного канала. В этом случае более четко можно оценить различие в форме движущегося пятна, т. к. отсутствуют искажения электрического поля, вносимые углами области. Можно уверенно сказать, что наиболее правильный расчет возможен при использовании метода характеристик (нижний ряд на рис. 11.7, 11.8).

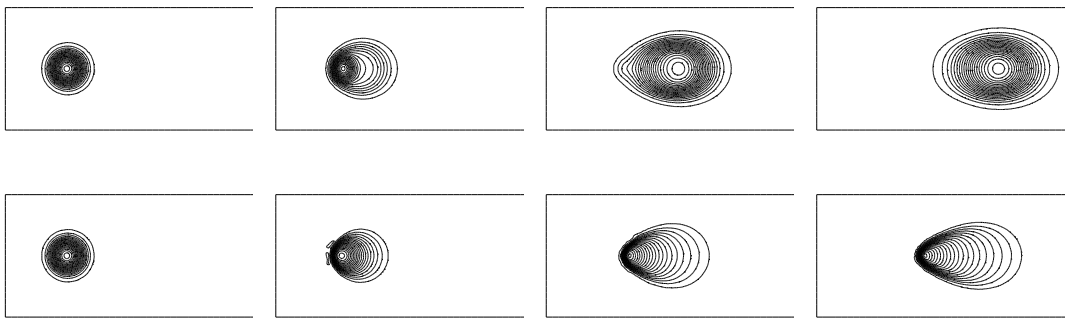

Рис. 11.8. Сравнение различных методов решения (нижний ряд — с использованием метода характеристик). Изолинии $c(x, y, t)$ при $t = 0{,}0;\ 0{,}2;\ 0{,}6;\ 1{,}0;\ \alpha = 2$

Глава 12

Задача о движении двух примесей

Рассмотрим задачу о движении двух не взаимодействующих непосредственно между собой примесей. В случае, когда концентрации примесей не влияют на проводимость среды, примеси будут двигаться, конечно же, независимо и задачи об их движении могут решаться отдельно. Интересно исследовать случай, когда концентрации примесей влияют на проводимость среды, тем самым изменяя по мере своего движения электрическое поле, которым, в свою очередь, и осуществляется перенос примесей.

12.1 Основные уравнения и постановка задачи

Уравнения движения примесей с учетом эффектов диффузии имеют вид (ср. с п. 11.1)

$$\frac{\partial c_k}{\partial t} + \operatorname{div} \boldsymbol{i}_k = 0, \quad \boldsymbol{i}_k = -\varepsilon_k \nabla c_k + \mu_k c_k \boldsymbol{E}, \quad k = 1, 2, \ldots, \tag{12.1}$$

где c_k, $\boldsymbol{i}_k$ — концентрация и плотность потока k-ой примеси, $\boldsymbol{E}$ — напряженность электрического поля, μ_k, ε_k — электрофоретическая подвижность и коэффициент диффузии k-ой примеси.

Предполагаем, что электрическое поле потенциально, справедлив закон Ома и выполнено уравнение неразрывности для плотности электрического тока (см. уравнения (11.2)–(11.4))

$$\boldsymbol{E} = -\nabla \varphi, \quad \boldsymbol{j} = \sigma \boldsymbol{E}, \quad \operatorname{div} \boldsymbol{j} = 0, \tag{12.2}$$

где φ — электрический потенциал, σ — проводимость среды, $\boldsymbol{j}$ — плотность электрического тока.

Выберем зависимость проводимости среды от концентраций в форме

$$\sigma = \sigma_0 \left(1 + \sum_{k=1} \alpha_k c_k \right), \tag{12.3}$$

где σ_0 — проводимость среды в отсутствии примесей, α_k — коэффициент влияния k-ой примеси на проводимость.

Уравнения для определения потенциала, как и в п. 11.1 (см. (11.6)), будут следующими

$$\sigma\Delta\varphi + \nabla\varphi\cdot\nabla\sigma = 0. \tag{12.4}$$

Как и в п. 11.1 считаем, что часть границы области (Γ_{i}) электроизолирована, на части границы области (Γ_φ) задан потенциал и вся граница непроницаема для концентрации

$$(\boldsymbol{i}_k\cdot\boldsymbol{n})\big|_{\partial D} = 0, \quad \left(\varepsilon_k\frac{\partial c_k}{\partial n} + \mu_k c_k\boldsymbol{n}\cdot\nabla\varphi\right)_{\partial D} = 0, \tag{12.5}$$

$$(\boldsymbol{n}\cdot\nabla\varphi)\big|_{\Gamma_{\mathrm{i}}} = \left(\frac{\partial\varphi}{\partial n}\right)_{\Gamma_{\mathrm{i}}} = 0, \quad \varphi\big|_{\Gamma_\varphi} = \varphi_0(x,y)\big|_{\Gamma_\varphi}, \quad \partial D = \Gamma_{\mathrm{i}}\cup\Gamma_\varphi, \tag{12.6}$$

где $\varphi_0(x,y)$ — известная функция на границе.

Используя явно-неявную аппроксимацию по времени для уравнения переноса примесей (12.1), запишем слабую формулировку задачи с учетом краевых условий (12.5)

$$\sum_{k=1}\iint\limits_D\left(\theta_k\frac{c_k^{m+1}-c_k^m}{\tau} + \nabla\theta_k\cdot(\varepsilon_k\nabla c_k^{m+1} + \mu_k c_k^{m+1}\nabla\varphi^m)\right)dx\,dy = 0. \tag{12.7}$$

Напомним, что θ_k — тестовые функции для концентраций c_k и, как уже говорилось в п. 6.1, вариационная формулировка задачи записывается в виде *одного* соотношения для всех неизвестных функций c_k, а не для каждой функции в отдельности.

✓. *Результаты расчетов предыдущей главы показали, что задачу о переносе лучше решать методом характеристик. Непосредственное использование слабой формулировки (*12.7*) может повлечь погрешности расчета, если его проводить с большим шагом по времени и крупной сеткой.*

12.2 Реализация алгоритма решения на языке FreeFem++

Код программы на языке FreeFem++ приведен в случае двух примесей.

```
1   int m, n=2;
2   real x1,x2,x3,x4;
3   real y1,y2,y3,y4;
4   real w1=5, h1=1;
5   x1=0; y1=0;   x2=w1; y2=0;   x3=w1; y3=h1;   x4=0; y4=h1;
6   border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2; };
7   border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3; };
8   border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4; };
9   border L4(t=0,1){ x=(1-t)*x4+t*x1;  y=(1-t)*y4+t*y1; };
10  mesh Th= buildmesh(L1(10*n)+L2(5*n)+L3(10*n)+L4(5*n));
11  fespace Vh(Th,P2);
12  real hinit=0.05;
13  Vh hh=hinit;
```

```
14  Th=adaptmesh(Th,hh,IsMetric=1,splitpbedge=1,nbvx=10000);
15  real epsilon1=0.002, epsilon2=0.002, phi4=0, phi2=-10,
16  mu1=0.51, mu2=-0.41, alpha1=2, alpha2=4;
17  real dt = 0.01,  t=0;
18  Vh v1, v2, v, phi, c1, c2, cold1, cold2, sigma;
19  real area0=int2d(Th)(1.0);
20  real Ampl1=1, x01=2.2, y01=0.5, beta1=50, r01=0.05;
21  real Ampl2=1, x02=2.8, y02=0.5, beta2=50, r02=0.05;
22  c1 = Ampl1*(1+tanh( -beta1*((x-x01)^2 + (y-y01)^2 - r01^2) ));
23  c2 = Ampl2*(1+tanh( -beta2*((x-x02)^2 + (y-y02)^2 - r02^2) ));
24  real massac1= int2d(Th)(c1),  massac2= int2d(Th)(c2);
25  problem pb4c(c1, c2, v1, v2, init=m, solver=LU)
26     = int2d(Th)( c1*v1/dt + epsilon1*(dx(c1)*dx(v1)+dy(c1)*dy(v1)) )
27       + int2d(Th)( mu1*c1*(dx(phi)*dx(v1)+dy(phi)*dy(v1)) )
28       - int2d(Th)( cold1/dt * v1 )
29       + int2d(Th)( c2*v2/dt + epsilon2*(dx(c2)*dx(v2)+dy(c2)*dy(v2)) )
30       + int2d(Th)( mu2*c2*(dx(phi)*dx(v2)+dy(phi)*dy(v2)) )
31       - int2d(Th)( cold2/dt * v2 )          ;
32  problem pbPhi(phi, v,  solver=LU)
33          = int2d(Th)( sigma*(dx(phi)*dx(v)+dy(phi)*dy(v)) )
34            + on(L4, phi=phi4)
35            + on(L2, phi=phi2) ;
36  sigma=1+alpha1*c1+alpha2*c2;
37  pbPhi;
38  for(m=0; m<10000; m++)
39  { t = t+dt;
40    cold1 = c1;    cold2 = c2;
41    sigma=1+alpha1*cold1+alpha2*cold2;
42    pb4c;
43    for (int j=0; j<c1[].n ; j++)   { if (c1[][j]<0) c1[][j]=0; }
44    for (int j=0; j<c2[].n ; j++)   { if (c2[][j]<0) c2[][j]=0; }
45    real averC1=(int2d(Th)(c1)-massac1)/area0;
46    c1=c1-averC1;
47    real averC2=(int2d(Th)(c2)-massac2)/area0;
48    c2=c2-averC2;
49    sigma=1+alpha1*c1+alpha2*c2;
50    pbPhi;
51    adaptmesh(Th, c1, c2);
52    plot(sigma, cmm=+t);
53   }
```

12.3 Вычислительный эксперимент

Результаты расчетов движения двух примесей в прямоугольнике $\overline{D} = [0,5] \times [0,1]$ для параметров $\varepsilon_1, \varepsilon_2 = 0{,}002$, $\varphi_1 = 0$, $\varphi_2 = -10$, $\mu_1 = 0{,}51$, $\mu_2 = -0{,}41$, $\alpha_1 = 2$, $\alpha_2 = 4$ приведены на рис. 12.1 ($\tau = 0{,}01$). Начальное распределение задавалось в виде

$$c_k = A_k\big(1 + \mathrm{th}(-\beta_k((x - x_k)^2 + (y - y_k)^2 - r_k^2))\big),$$

где $A_1 = A_2 = 1$, $\beta_1 = \beta_2 = 50$, $r_1 = r_2 = 0{,}05$, $x_1 = 2{,}2$, $y_1 = 0{,}5$, $x_2 = 2{,}8$, $y_2 = 0{,}5$.

На рис. 12.1 приведены изолинии функции

$$\sigma(x, y, t) = 1 + \alpha_1 c_1(x, y, t) + \alpha_2 c_2(x, y, t).$$

Хорошо видно, что примеси движутся навстречу друг другу, сливаются и затем начинают расходиться в разные стороны — первая примесь движется вправо, а вторая — влево (примерно при $t > 1$). Искажение формы «пятен» примеси вызвано электромиграционными и диффузионными эффектами.

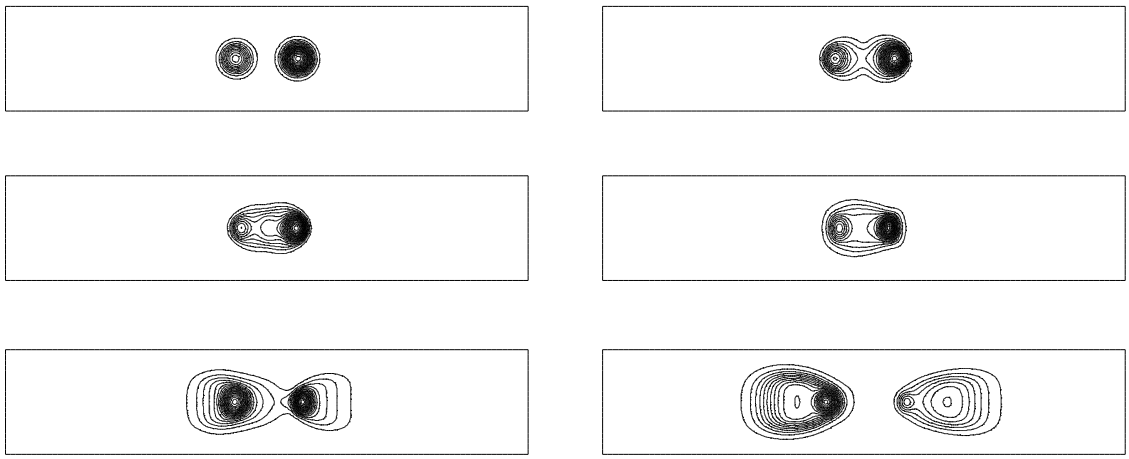

Рис. 12.1. Изолинии функции $\sigma(x, y, t)$ при $t = 0{,}0;\ 0{,}1;\ 0{,}2;\ 0{,}6;\ 1{,}0;\ 1{,}4$

На одном рисунке можно изображать изолинии сразу нескольких функций (см. рис. 12.2), для этого следует записать

```
plot(c1, c2, phi, nbiso=80);
```

где `nbiso` — общее количество выводимых изолиний (см. также с. 239).

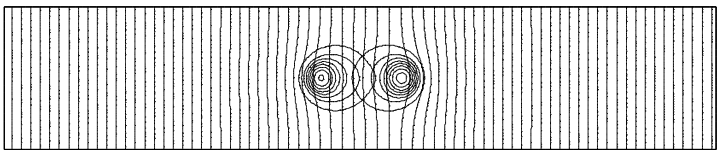

Рис. 12.2. Изолинии функций $\varphi(x, y, t)$, $c_1(x, y, t)$, $c_2(x, y, t)$ при $t = 0{,}12$

12.4 Решение задачи с использованием ключевого слова `convect`

При помощи (12.2) выразим $\boldsymbol{E}$ через $\boldsymbol{j}$ и, учитывая равенство $\operatorname{div} \boldsymbol{j} = 0$, получим

$$\frac{\partial c_k}{\partial t} - \varepsilon_k \Delta c_k + \boldsymbol{j} \cdot \nabla \left(\frac{\mu_k c_k}{\sigma} \right) = 0 \tag{12.8}$$

или

$$\frac{\partial c_k}{\partial t} - \varepsilon_k \Delta c_k - \sigma \nabla \varphi \cdot \nabla \left(\frac{\mu_k c_k}{\sigma} \right) = 0. \tag{12.9}$$

Дальнейшие преобразования приводят к следующим уравнениям

$$\frac{\partial c_k}{\partial t} - \varepsilon_k \Delta c_k - \sigma \nabla \varphi \cdot \left(\frac{\mu_k \nabla c_k}{\sigma} - \frac{\mu_k c_k \nabla \sigma}{\sigma^2} \right) = 0. \tag{12.10}$$

Окончательно, с учетом (12.4) выводим

$$\frac{\partial c_k}{\partial t} - \varepsilon_k \Delta c_k - \mu_k \nabla \varphi \cdot \nabla c_k - \mu_k c_k \Delta \varphi = 0, \quad k = 1, 2, \ldots \tag{12.11}$$

Интересно заметить, что уравнения (12.11) можно трактовать, формально, как уравнения переноса концентраций со скоростями $\mu_k \nabla \varphi$ и источниками концентраций $\mu_k c_k \Delta \varphi$.

Использование явно-неявной схемы аппроксимации по времени дает

$$\frac{c_k^{m+1} - c^m(\boldsymbol{X}^m(\boldsymbol{x}))}{\tau} - \varepsilon_k \Delta c_k^{m+1} - \mu_k c_k^{m+1} \Delta \varphi^m = 0, \tag{12.12}$$

$$\sigma^m \Delta \varphi^m + \nabla \varphi^m \cdot \nabla \sigma^m = 0. \tag{12.13}$$

Здесь, как обычно, обозначено $c^m = c(x, y, t_m)$, $\varphi^m = \varphi(x, y, t_m)$ и

$$c_k^m(\boldsymbol{X}^m(\boldsymbol{x})) = \texttt{convect([v1,v2],-dt,}c_k^m\texttt{)},$$

$$\texttt{v1} = -\mu_k \varphi_x^m, \quad \texttt{v2} = -\mu_k \varphi_y^m.$$

Глава 13

Перенос примеси электрическим полем и жидкостью

Используя результаты гл. 10 и 11, нетрудно рассмотреть движение примеси под действием электрического поля и течения жидкости.

13.1 Постановка задачи

Система уравнений, описывающая поведение примеси, имеет вид

$$\frac{d\boldsymbol{v}}{dt} = -\nabla p + \mu\Delta\boldsymbol{v}, \quad \operatorname{div}\boldsymbol{v} = 0, \quad \frac{d}{dt} = \frac{\partial}{\partial t} + \boldsymbol{v}\cdot\nabla, \tag{13.1}$$

$$\frac{dc}{dt} + \operatorname{div}\boldsymbol{i} = 0, \quad \boldsymbol{i} = -\varepsilon\nabla c + \mu_e c\boldsymbol{E}, \tag{13.2}$$

$$\boldsymbol{E} = -\nabla\varphi, \quad \boldsymbol{j} = \sigma\boldsymbol{E}, \quad \operatorname{div}\boldsymbol{j} = 0, \tag{13.3}$$

где $\boldsymbol{v}$ — скорость течения жидкости, p — давление, μ — коэффициент вязкости, c — концентрация примеси, $\boldsymbol{i}$ — плотность потока примеси, $\boldsymbol{E}$ — напряженность электрического поля, μ_e — электрофоретическая подвижность примеси (скорость на единицу электрического поля), ε — коэффициент диффузии, φ — электрический потенциал, $\boldsymbol{j}$ — плотность электрического тока, σ — проводимость среды.

Возьмем зависимость проводимости от концентрации в виде

$$\sigma = \sigma_0(1 + \alpha c), \tag{13.4}$$

где σ_0 — проводимость среды в отсутствии примеси, α — коэффициент влияния примеси на проводимость.

Приведенная система уравнений соответствует случаю, когда перенос примеси жидкостью осуществляется пассивно, т. е. движение примеси не влияет на течение жидкости. Перенос же в электрическом поле зависит от процесса движения примеси, т. к. в силу (13.4) примесь влияет на проводимость, изменяя тем самым напряженность электрического поля.

Можно рассмотреть и более сложный случай, когда движение примеси влияет на течение жидкости, учитывая, например, действие силы тяжести

на примесь и добавив для этого в уравнение (13.1) член, соответствующий архимедовой силе (ср. с уравнением тепловой конвекции в гл. 8)

$$\frac{d\boldsymbol{v}}{dt} = -\nabla p + \mu\Delta\boldsymbol{v} - \boldsymbol{k}c, \tag{13.5}$$

где $\boldsymbol{k}$ — единичный орт в направлении, противоположном действию силы тяжести (знак «минус» соответствует случаю, когда плотность примеси больше плотности жидкости — тяжелая примесь).

Далее ограничимся рассмотрением задачи о протекании жидкости через Т-образную область (см. гл. 10) и действии на примесь электрического поля (см. гл. 11). Краевые и начальные условия для уравнений (13.1)–(13.4) возьмем такими же, как в указанных главах, т. е. (10.5), (10.6), (11.15), (11.16), (11.18)

$$\boldsymbol{v}\big|_{\partial D\setminus(\Gamma_{\text{in}}\cup\Gamma_{\text{out}})} = 0, \quad \boldsymbol{v}\big|_{\Gamma_{\text{in}}} = \boldsymbol{v}_0, \quad p\big|_{\Gamma_{\text{out}}} = p_0, \quad \left.\frac{du}{dt}\right|_{\Gamma_{\text{out}}} = 0, \quad w\big|_{\Gamma_{\text{out}}} = 0, \tag{13.6}$$

$$\boldsymbol{v}\big|_{t=0} = \boldsymbol{h}(x,y), \tag{13.7}$$

$$(\boldsymbol{i}\cdot\boldsymbol{n})\big|_{\partial D} = 0, \quad \left(\varepsilon\frac{\partial c}{\partial n} + \mu c\boldsymbol{n}\cdot\nabla\varphi\right)_{\partial D} = 0, \tag{13.8}$$

$$(\boldsymbol{n}\cdot\nabla\varphi)\big|_{\Gamma_{\text{i}}} = \left(\frac{\partial\varphi}{\partial n}\right)_{\Gamma_{\text{i}}} = 0, \quad \varphi\big|_{\Gamma_\varphi} = \varphi_0(x,y)\big|_{\Gamma_\varphi}, \quad \partial D = \Gamma_{\text{i}}\cup\Gamma_\varphi, \tag{13.9}$$

$$c(x,y,0) = c_0(x,y), \tag{13.10}$$

где $c_0(x,y)$, $\varphi_0(x,y)$, $\boldsymbol{h}(x,y)$ — известные функции.

13.2 Вычислительный эксперимент

Программу на языке FreeFem++ для рассматриваемой задачи легко записать на основе кода из п. 10.3.

13.2.1 Перенос примеси жидкостью и электрическим полем

На рис. 13.1 представлены результаты расчетов для значений параметров: $\mu_e = -1{,}51$, $\alpha = 2$, $\mu = 0{,}0025$, $\varepsilon = 0{,}002$ с шагом по времени $\tau = 0{,}01$.

Будем решать задачу для области, представленной на рис. 9.3. Координаты угловых точек возьмем следующими: $x_1 = 0$, $y_1 = 0$; $x_2 = 1$, $y_2 = 0$; $x_3 = 1$, $y_3 = -1$; $x_4 = 2$, $y_4 = -1$; $x_5 = 2$, $y_5 = 0$; $x_6 = 3$, $y_6 = 0$; $x_7 = 3$, $y_7 = 1$; $x_8 = 0$, $y_8 = 1$.

На входной части границы (L_8) профиль скорости задаем в виде

$$u\big|_{L_8} = 4y(1-y), \quad w\big|_{L_8} = 0. \tag{13.11}$$

Граница L_6 предполагается открытой для течения жидкости, а на остальной части границы задаем условия прилипания ($u = 0$, $w = 0$).

На участках границы L_3, L_6, L_8 задаем электрический потенциал, соответственно $\varphi_3 = 10$, $\varphi_6 = 10$, $\varphi_8 = 0$, а остальную часть границы считаем изолированной.

Начальное распределение скорости выберем в виде

$$u\big|_{t=0} = 4y(1-y), \quad (x,y) \in D_0 = \{0 < y \leqslant 1,\ 0 < x \leqslant 3\},$$
$$u\big|_{t=0} = 0, \quad (x,y) \in D \setminus D_0, \quad w\big|_{t=0} = 0, \quad (x,y) \in D. \tag{13.12}$$

Начальное распределение примеси задаем соотношением

$$c = 1 + \operatorname{th}\big(-\beta((x-x_0)^2 + (y-y_0)^2 - r_0^2)\big), \tag{13.13}$$

где $x_0 = 0{,}5$, $y_0 = 0{,}3$, $\beta = 50$, $r_0 = 0{,}05$.

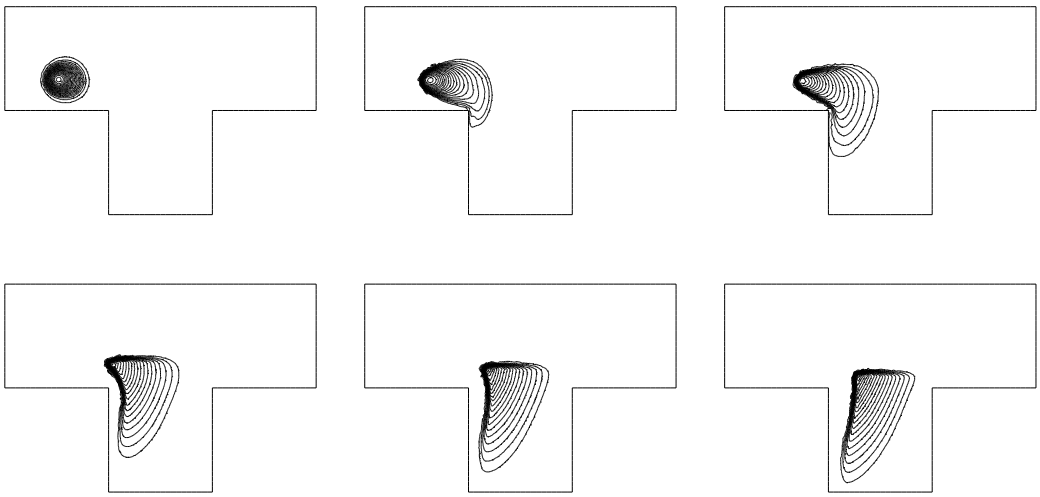

Рис. 13.1. Изолинии концентрации c при $t = 0{,}00$; $0{,}05$; $0{,}10$; $0{,}20$; $0{,}25$; $0{,}30$

На рис. 13.2 приведено увеличенное изображение изолиний функции тока (слева) и электрического потенциала в начальный момент времени.

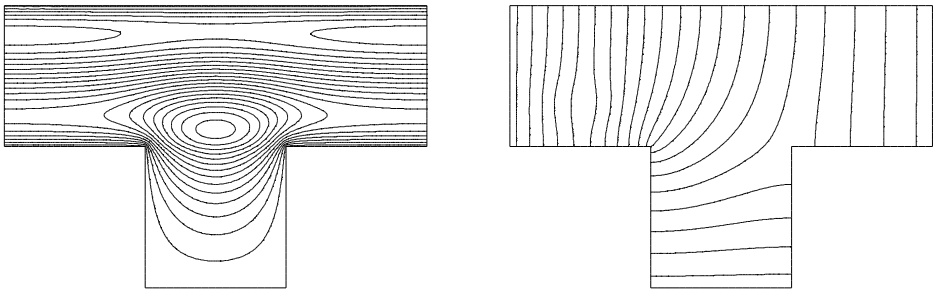

Рис. 13.2. Изолинии функции тока $\psi(x,y,t)$ и потенциала $\varphi(x,y,t)$ при $t = 0$

Отметим, что картина течения слабо меняется со временем, т. к. примесь не вносит искажения в движение жидкости и течение почти стационарно. Напротив, примесь при своем движении изменяет проводимость

среды и изолинии потенциала существенно изменяются во времени. В частности, в области, занимаемой примесью, изолинии потенциала расположены менее плотно, чем в остальной области (см. левую верхнюю часть Т-образного канала).

13.2.2 Сравнение различных типов переноса

Интересно исследовать влияние на процесс переноса примеси различных факторов — движение жидкости, действие электрического поля и т. п. Результаты сравнения для некоторого интервала времени см. на рис. 13.3.

1. Верхний ряд. Перенос примеси жидкостью и электрическим полем.

2. Средний ряд. Перенос примеси электрическим полем в неподвижной жидкости ($u = 0$, $w = 0$).

3. Нижний ряд. Перенос примеси жидкостью в отсутствии электрического поля ($u \neq 0$, $w \neq 0$, $\mu_e = 0$).

При параметрах, использованных для расчета, интенсивность течения жидкости достаточно слабая по сравнению с интенсивностью электрического поля. Тем не менее, хорошо видны искажения, которые вносит жидкость в перенос примеси электрическим полем.

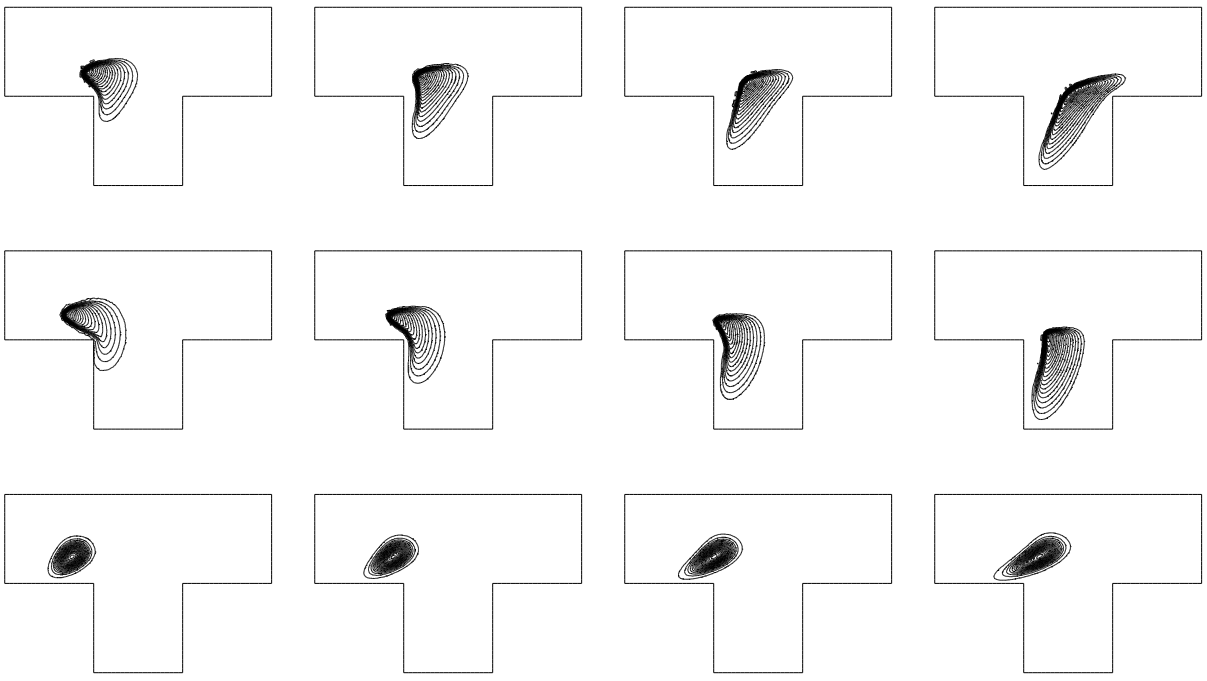

Рис. 13.3. Изолинии концентрации $c(x, y, t)$ при $t = 0{,}30;\ 0{,}45;\ 0{,}60;\ 0{,}80$

Глава 14

Решение задачи со свободной границей

FreeFem++ предоставляет возможность решать задачи со свободной границей. Для этих целей предусмотрен оператор `movemesh`, позволяющий осуществлять деформирование сетки в процессе построения решения.

Пусть, например, имеется сетка `Th`, дискретизирующая исходную область D. Запишем следующее выражение

```
Th = movemesh(Th, [x+xx,y+yy]);
```
(14.1)

где `xx`, `yy` — функции, определенные на пространстве конечных элементов.

Предполагается, что функции `xx`, `yy` вычислены или заданы до выполнения оператора `movemesh`. Например,

```
Vh xx=sin(x*y), yy=cos(y)*exp(x);
```

Действие `movemesh` приведет к тому, что исходная сетка `Th` будет «сдвинута» вдоль осей x, y

$$\texttt{x} \to \texttt{x+xx}, \qquad \texttt{y} \to \texttt{y+yy}.$$

Заметим, что такая операция небезопасна, т. к. при деформации сетки некоторые ее элементы (треугольники) могут исчезнуть или в результате деформации площадь некоторых треугольников станет отрицательной. Для обработки таких случаев язык FreeFem++ предлагает ключевое слово `checkmovemesh`, которое можно использовать следующим образом

```
real minarea = checkmovemesh(Th, [x+xx,y+yy]);
if (minarea >0 ) // деформация будет правильной
  {
    Th = movemesh(Th, [x+xx,y+yy]);
  }
```

Иными словами, следует предварительно проверить правильность деформации сетки и лишь затем ее деформировать.

Наличие оператора `movemesh` позволяет решать задачи, по крайней мере, в случае, когда имеются движущиеся области или части границ области, которые не сильно изменяются в процессе эволюции. Продемонстрируем это на примере задачи для жидкости со свободной границей.

14.1 Постановка задачи

Пусть имеется прямоугольный контейнер, заполненный жидкостью (область D). Уравнения движения жидкости запишем в виде

$$\frac{d\boldsymbol{v}}{dt} = -\nabla p + \mu\Delta\boldsymbol{v}, \quad \operatorname{div}\boldsymbol{v} = 0, \quad (x,y) \in D. \tag{14.2}$$

Здесь $\boldsymbol{v} = \boldsymbol{v}(x,y,t)$ — скорость, $p = p(x,y,t)$ — давление, μ — коэффициент кинематической вязкости.

Предположим, что область D имеет свободную границу Γ_0, а остальная часть границы $\partial D \setminus \Gamma_0$ — твердая и непроницаема для жидкости. На твердой границе поставим условие прилипания жидкости

$$\boldsymbol{v}\big|_{\partial D\setminus\Gamma_0} = 0. \tag{14.3}$$

Кинематическое условие на свободной границе, соответствующее случаю, когда частицы жидкости движутся вместе с границей (точнее, частицы жидкости на границе не могут ее покинуть), имеет вид

$$\left.\frac{d\boldsymbol{v}}{dt}\right|_{\Gamma_0} = 0. \tag{14.4}$$

Кроме этого, на свободной границе необходимо задавать так называемое динамическое условие, в качестве которого выберем условие, отвечающее случаю, когда свободная граница жидкости контактирует с окружающим воздухом и давление окружающей среды равняется нулю

$$p\big|_{\Gamma_0} = 0. \tag{14.5}$$

Заметим, что если уравнение границы известно, например, задается соотношением $y = F(x,t)$, то кинематическое условие имеет вид

$$w\big|_{\Gamma_0} = \frac{dy}{dt}, \quad w\big|_{\Gamma_0} = F_t + uF_x. \tag{14.6}$$

Зададим также начальное условие

$$\boldsymbol{v} = \boldsymbol{h}(x,y), \tag{14.7}$$

где $\boldsymbol{h}(x,y)$ — известное начальное распределение скорости.

Рассмотрим случай, когда функция $\boldsymbol{h}(x,y)$ отлична от нуля лишь в малой окрестности свободной границы Γ_0. В качестве $\boldsymbol{h}(x,y)$ можно взять функцию, которая на пересечении круга C с областью D отлична от нуля, а в оставшейся части области D равна нулю (см. рис. 14.1)

$$\boldsymbol{h}(x,y) = \begin{cases} (0,0), & (x,y) \in S, \\ (0,w_0), & (x,y) \in D \setminus S, \end{cases} \tag{14.8}$$

$$C = \left\{(x,y) : (x-x_0)^2 + (y-y_0)^2 \leqslant r_0^2\right\}, \quad S = C \cap D.$$

Здесь x_0, y_0, r_0, w_0 — известные параметры.

Такое начальное условие будет очень грубо имитировать вертикальный удар по свободной границе жидкости.

14.2 Код на языке FreeFem++

Для определенности рассмотрим прямоугольный контейнер (рис. 14.1), у которого границы L_1, L_2, L_4 твердые, а граница L_3 — свободная.

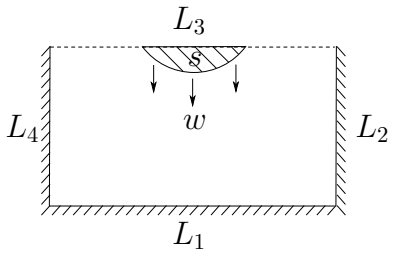

Рис. 14.1. Контейнер со свободной границей L_3 и твердыми границами L_1, L_2, L_4

В соответствие с (14.8) запишем функцию $\boldsymbol{h}(x, y)$

```
Vh u=0, w=0;
w = w0*((x-x0)^2+(y-y0)^2<=r0^2);
```

Еще раз подчеркнем, что это слишком упрощенная имитация удара круга о границу. Основная цель, которая преследовалась при рассмотрении данной задачи, — продемонстрировать возможности оператора деформирования сетки `movemesh`. Можно предсказать результаты расчетов. На границах L_1, L_2, L_4 скорости равны нулю, и эти границы будут оставаться неподвижными. В некоторой области границы L_3 задано начальное распределение скорости. Эта часть границы будет перемещаться в соответствии с течением жидкости внутри области.

Полный текст программы имеет вид

```
1  int m;
2  real x1,x2,x3,x4;
3  real y1,y2,y3,y4;
4  real w1=1,  h1=1;
5  x1=0; y1=0;    x2=w1; y2=0;    x3=w1; y3=h1;    x4=0; y4=h1;
6  real x0=0.5*w1, dr=0.305, y0=h1+dr, r0=0.33;
7  border L1(t=0,1){ x=(1-t)*x1+t*x2;  y=(1-t)*y1+t*y2; };
8  border L2(t=0,1){ x=(1-t)*x2+t*x3;  y=(1-t)*y2+t*y3; };
9  border L3(t=0,1){ x=(1-t)*x3+t*x4;  y=(1-t)*y3+t*y4; };
10 border L4(t=0,1){ x=(1-t)*x4+t*x1;  y=(1-t)*y4+t*y1; };
11 border C0(t=0,2*pi){ x=x0+r0*cos(t);  y=y0+r0*sin(t);};
12 int n=2;
13 mesh Th= buildmesh(L1(10*n)+L2(5*n)+L3(10*n)+L4(5*n));
14 plot(Th, wait=1);
15 fespace Vh(Th,P2);
16 real hinit=0.02; // начальный размер сетки
17 Vh hh=hinit;     // FE-функция для задания размера сетки
18 // построение сетки с заданным размером: hinit
19 Th = adaptmesh(Th, hh, IsMetric=1, splitpbedge=1, nbvx=10000);
20 plot(Th, wait=1);
```

```
real w0 = -0.5;
real mu = 0.025;
real dt = 0.01,  t=0;
Vh w, u, v, p, q, psi;
w = 0;   u = 0;
w = w0*((x-x0)^2+(y-(y0))^2<=r0^2);
v = 0;   p = 0;   q = 0;  psi = 0;
real area = int2d(Th)(1.);
Vh uold, wold, pold;
Vh f, g;
real meandiv, meanpq;
problem pb4u(u, v, solver=UMFPACK)
             = int2d(Th)( u*v/dt + mu*(dx(u)*dx(v)+dy(u)*dy(v)) )
               - int2d(Th)( (f/dt-dx(p))*v )
               + on(L1,L2,L4,u = 0)
               + on(L3,u = f);
problem pb4w(w, v, solver=UMFPACK)
             = int2d(Th)(w*v/dt +mu*(dx(w)*dx(v)+dy(w)*dy(v)))
               - int2d(Th)( (g/dt-dy(p))*v )
               + on(L1,L2,L4, w = 0)
               + on(L3,w =g);
problem pb4p(q, v, solver=UMFPACK)
             = int2d(Th)( dx(q)*dx(v)+dy(q)*dy(v) )
               + int2d(Th)( (dx(u)+ dy(w)-meandiv)*v/dt )
               + on(L3, q=0);    //  meandiv = A*dt/S
problem pb4psi(psi, v, solver=UMFPACK)
             = int2d(Th)(dx(psi)*dx(v)+dy(psi)*dy(v))
               - int2d(Th)((dx(w)- dy(u))*v)
               +  on(L1,L2,L4, psi=0);
for(m=0; m<10000; m++)
  {
    t = t + dt;
    uold = u;   wold = w;   pold = p;
    f = convect([u,w],-dt,uold);
    g = convect([u,w],-dt,wold);
    pb4u;
    pb4w;
    meandiv = int2d(Th)(dx(u)+dy(w))/area;
    pb4p;
    meanpq = int2d(Th)(pold + q)/area;
    p = pold + q - meanpq;
    u = u - dx(q)*dt;
    w = w - dy(q)*dt;
    real minarea = checkmovemesh(Th, [x+u*dt,y+w*dt]);
    if (minarea >0 )
    // проверка положительности площади треугольников при деформации
      { Th = movemesh(Th, [x+u*dt,y+w*dt]); }
    pb4psi;
    plot(psi, fill=0, cmm=+t, wait=0);
  }
```

14.3 Результаты расчетов

На рис. 14.2 приведены результаты расчетов при $\mu = 0{,}025$ с шагом по времени $\tau = 0{,}01$ (остальные параметры, задающие границы области и начальное распределение скорости, см. в тексте программы). Вязкость жидкости выбрана достаточно большой, т. к. в противном случае свободная граница деформируется слишком быстро и проведение расчетов с большим шагом по времени невозможно.

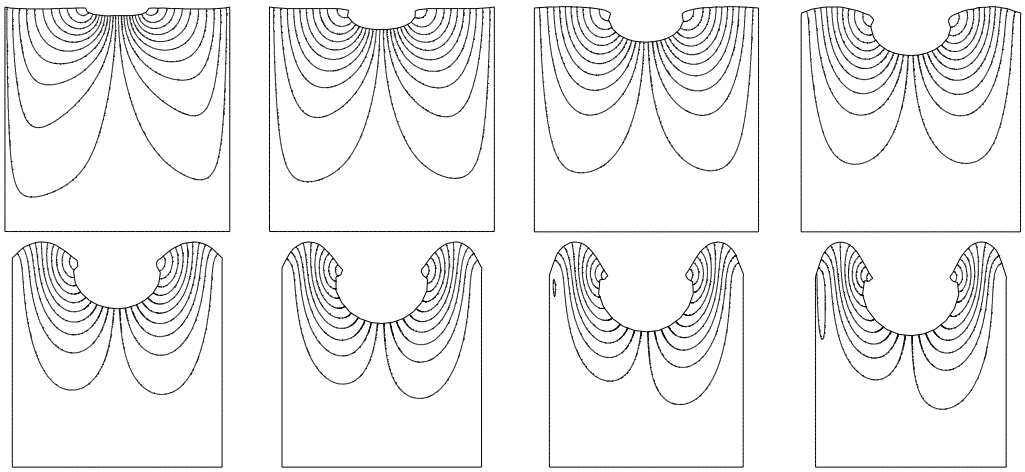

Рис. 14.2. Изолинии функции тока ψ при $t = 0{,}15;\ 0{,}35;\ 0{,}55;\ 0{,}75;\ 1{,}05;\ 1{,}35;\ 1{,}55;\ 1{,}65$

Начиная с момента $t = 1{,}69$ расчет невозможен. Дело в том, что генератор сетки FreeFem++ далее не сможет создать сетку, т. к. в результате деформации возникнут треугольники с отрицательной площадью.

На рис. 14.3 приведены изолинии давления (изобары).

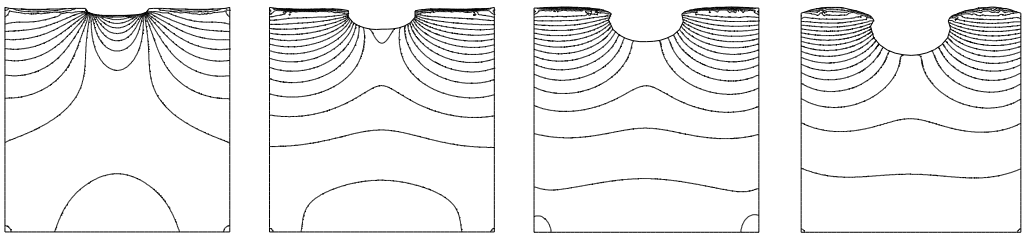

Рис. 14.3. Изолинии давления p при $t = 0{,}15;\ 0{,}35;\ 0{,}55;\ 0{,}75;$

Глава 15

Течение Куэтта-Тейлора между вращающимися цилиндрами

Используя FreeFem++, можно решать и 3D вращательно-симметричные задачи, которые, фактически, являются двумерными задачами. К сожалению, специальных средств, таких как, например, вычисление интегралов в случае цилиндрических или сферических координат, язык FreeFem++ не предоставляет. Не предусмотрены и зарезервированные идентификаторы, скажем, для цилиндрических координат (r, θ, z), и приходится пользоваться идентификаторами x, y, подразумевая $r = x$, $y = z$.

В качестве примера рассмотрим задачу о вращательно-симметричном течении между двумя соосными цилиндрами — течении Куэтта–Тейлора.

15.1 Постановка задачи

Пусть область $\overline{D} = [r_1, r_2] \times [-L, L]$ между двумя соосными цилиндрами с радиусами r_1, r_2 и высотой $2L$ заполнена вязкой несжимаемой жидкостью. Предполагаем, что в осевом направлении область ограничена твердыми плоскими непроницаемыми стенками («крышки») и цилиндры вращаются с угловыми скоростями ω_1, ω_2.

Система уравнений Навье–Стокса в предположении вращательной симметрии (отсутствует зависимость функций от θ) в цилиндрических координатах (r, θ, z) имеет вид (см., например, [16])

$$(ru)_r + rw_z = 0, \tag{15.1}$$

$$u_t + \boldsymbol{v} \cdot \nabla u - \frac{v^2}{r} = -p_r + \mu \left(\Delta u - \frac{u}{r^2} \right), \tag{15.2}$$

$$v_t + \boldsymbol{v} \cdot \nabla v + \frac{uv}{r} = \mu \left(\Delta v - \frac{v}{r^2} \right), \tag{15.3}$$

$$w_t + \boldsymbol{v} \cdot \nabla w = -p_z + \mu \Delta w, \tag{15.4}$$

$$\Delta = \frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial}{\partial r} + \frac{\partial^2}{\partial z^2}, \quad \boldsymbol{v}\cdot\nabla = u\frac{\partial}{\partial r} + w\frac{\partial}{\partial z}, \quad \boldsymbol{v} = (u, v, w). \tag{15.5}$$

Здесь $\boldsymbol{v}$ — скорость течения жидкости (u, v, w — соответственно, радиальная, азимутальная и осевая компоненты скорости), p — давление, μ — кинематическая вязкость.

Краевые условия возьмем в виде

$$u\big|_{r=r_1} = 0, \quad u\big|_{r=r_2} = 0, \quad w\big|_{r=r_1} = 0, \quad w\big|_{r=r_2} = 0, \tag{15.6}$$

$$v\big|_{r=r_1} = \omega_1 r_1, \quad v\big|_{r=r_2} = \omega_2 r_2, \tag{15.7}$$

$$u\big|_{z=\mp L} = 0, \quad v\big|_{z=\mp L} = 0, \quad w\big|_{z=\mp L} = 0. \tag{15.8}$$

Кроме этого, зададим начальное распределение скорости $\boldsymbol{v}_0(r, z)$

$$\boldsymbol{v}\big|_{t=0} = \boldsymbol{v}_0. \tag{15.9}$$

Краевые условия (15.8) можно заменить условиями периодичности, что позволит моделировать течение между цилиндрами, бесконечными в осевом направлении

$$u\big|_{z=-L} = u\big|_{z=+L}, \quad v\big|_{z=-L} = v\big|_{z=+L}, \quad w\big|_{z=-L} = w\big|_{z=+L} = 0. \tag{15.10}$$

Можно также одну (или обе) из крышек цилиндров считать свободной границей. Например, для случая плоской свободной недеформируемой границы жидкости на «крышке» $z = +L$, краевые условия, соответствующие отсутствию касательных напряжений и непроницаемости границы для жидкости, имеют вид

$$\left(v_r - \frac{v}{r}\right)\Big|_{z=+L} = 0, \quad v_z\big|_{z=+L} = 0, \quad w\big|_{z=+L} = 0. \tag{15.11}$$

Заметим, что в случае бесконечных цилиндров задача (15.1)–(15.7) имеет точное решение, которое и называется течением Куэтта-Тейлора

$$v = \frac{\omega_2 r_2^2 - \omega_2 r_2^2}{r_2^2 - r_1^2} \cdot r + \frac{(\omega_1 - \omega_2) r_1^2 r_2^2}{r_2^2 - r_1^2} \cdot \frac{1}{r}, \quad u = 0, \quad w = 0. \tag{15.12}$$

15.1.1 Функция тока

Приведем соотношения для функции тока ψ, которые, в частности, понадобятся для визуализации течения при численном решении задачи.

Уравнение неразрывности (15.1) будет тождественно выполнено, если ввести функцию тока ψ

$$u = \psi_z, \quad rw = -(r\psi)_r. \tag{15.13}$$

Дифференцируя, получим

$$u_z = \psi_{zz}, \quad w_r = -\left(\psi_{rr} + \frac{\psi_r}{r} - \frac{\psi}{r^2}\right). \tag{15.14}$$

Уравнение для определения функции тока будет следующим

$$\Delta\psi - \frac{\psi}{r^2} = u_z - w_r. \tag{15.15}$$

В случае краевых условий (15.8) для функции ψ на границе имеем

$$\psi|_{r=r_1} = 0, \quad \psi|_{r=r_2} = 0, \quad \psi|_{z=\mp L} = 0. \tag{15.16}$$

15.2 Алгоритм решения

Для решения задачи (15.1)–(15.9) используем проекционный алгоритм, подробно описанный в гл. 8. Фактически все соотношения гл. 8 претерпевают лишь незначительные изменения, связанные с записью уравнений Навье–Стокса в цилиндрической системе координат. В связи с этим, далее ограничиваемся лишь записью необходимых формул, сокращая до минимума пояснения.

15.2.1 Аппроксимация по времени

Явно-неявная аппроксимация уравнений (15.2)–(15.4) приводит к соотношениям

$$\frac{u^{m+1} - f^m}{\tau} - \frac{(v^m)^2}{r} = -p_r^m - q_r^{m+1} + \mu\left(\Delta u^{m+1} - \frac{u^{m+1}}{r^2}\right), \tag{15.17}$$

$$\frac{v^{m+1} - h^m}{\tau} + \frac{u^m v^m}{r} = \mu\left(\Delta v^{m+1} - \frac{v^{m+1}}{r^2}\right), \tag{15.18}$$

$$\frac{w^{m+1} - g^m}{\tau} = -p_z^m - q_z^{m+1} + \mu\Delta w^{m+1}, \tag{15.19}$$

где, как обычно, использовано ключевое слово `convect` языка FreeFem++

$$f^m \equiv u^m(\boldsymbol{X}^m(\boldsymbol{x})) = \texttt{convect}([u^m(\boldsymbol{x}), w^m(\boldsymbol{x})], -\tau, u^m(\boldsymbol{x})); \tag{15.20}$$

$$h^m \equiv v^m(\boldsymbol{X}^m(\boldsymbol{x})) = \texttt{convect}([u^m(\boldsymbol{x}), w^m(\boldsymbol{x})], -\tau, v^m(\boldsymbol{x})); \tag{15.21}$$

$$g^m \equiv w^m(\boldsymbol{X}^m(\boldsymbol{x})) = \texttt{convect}([u^m(\boldsymbol{x}), w^m(\boldsymbol{x})], -\tau, w^m(\boldsymbol{x})); \tag{15.22}$$

Напомним, что давление p^{m+1} представлено в виде

$$p^{m+1} = p^m + q^{m+1}. \tag{15.23}$$

15.2.2 Проекционный метод

Систему (15.17)–(15.19) заменяем следующими уравнениями

$$\frac{u^* - f^m}{\tau} - \frac{(v^m)^2}{r} = -p_r^m + \mu\left(\Delta u^* - \frac{u^*}{r^2}\right), \tag{15.24}$$

$$\frac{v^{m+1} - h^m}{\tau} + \frac{u^m v^m}{r} = \mu\left(\Delta v^* - \frac{v^*}{r^2}\right), \tag{15.25}$$

$$\frac{w^* - g^m}{\tau} = -p_z^m + \mu\Delta w^*. \tag{15.26}$$

Краевые условия для этой системы имеют вид

$$u^*\big|_{r=r_1} = 0, \quad u^*\big|_{r=r_2} = 0, \quad w^*\big|_{r=r_1} = 0, \quad w^*\big|_{r=r_2} = 0, \tag{15.27}$$

$$v^*\big|_{r=r_1} = \omega_1 r_1, \quad v^*\big|_{r=r_2} = \omega_2 r_2, \tag{15.28}$$

$$u^*\big|_{z=\mp L} = 0, \quad v^*\big|_{z=\mp L} = 0, \quad w\big|_{z=\mp L} = 0. \tag{15.29}$$

Именно задача (15.24)–(15.29) решается методом конечных элементов.

Если решение задачи (15.24)–(15.29) определено, то на $(m+1)$-ом временном шаге значения u^{m+1}, w^{m+1} разыскиваем в виде

$$u^{m+1} = u^* - \tau q_r^{m+1}, \quad w^{m+1} = w^* - \tau q_z^{m+1}, \quad v^{m+1} = v^*. \tag{15.30}$$

Считаем, что u^{m+1}, w^{m+1} удовлетворяют уравнениям неразрывности (15.1), и получаем уравнение для определения поправки к давлению q^{m+1}

$$\tau\Delta q^{m+1} = \frac{1}{r}(ru^*)_r + w_z^* \quad \Rightarrow \quad \Delta q^{m+1} = \frac{\operatorname{div} \boldsymbol{v}^*}{\tau}. \tag{15.31}$$

Краевым условием для этого уравнения в случае твердых непроницаемых границ Γ будет условие Неймана

$$\left.\frac{\partial q^{m+1}}{\partial n}\right|_{\Gamma} = 0. \tag{15.32}$$

15.2.3 Слабая формулировка задачи

Слабую формулировку задач (15.24)–(15.29) получим, умножая (15.24)–(15.29), соответственно, на тестовые функции U, V, W и интегрируя по частям по трехмерной области $\overline{D} \times [0, 2\pi]$. При этом следует помнить, что r, θ, z являются цилиндрическими координатами и элемент объема в этом случае будет $r\,dr\,d\theta\,dz$.

Разыскивается вращательно-симметричное решение и зависимость всех функций от азимутального угла θ отсутствует. Это означает, что

$$\iiint\limits_{D\times[0,2\pi]} (\dots) r\,dr\,d\theta\,dz = 2\pi \iint\limits_{D} (\dots) r\,dr\,dz. \tag{15.33}$$

Запишем окончательный вид слабых формулировок задач для определения u^*, v^*, w^* (использованная аппроксимация по времени позволяет решать эти задачи по отдельности)

$$\iint\limits_D \frac{u^* - f^m}{\tau} U r\,dr\,dz - \iint\limits_D \frac{(v^m)^2}{r} U r\,dr\,dz = -\iint\limits_D p_r^m U r\,dr\,dz -$$

$$- \iint\limits_D \mu(u_r^* U_r + u_z^* U_z) r\,dr\,dz - \iint\limits_D \mu \frac{u^*}{r^2} U r\,dr\,dz, \tag{15.34}$$

$$\iint\limits_D \frac{v^* - h^m}{\tau} V r\,dr\,dz + \iint\limits_D \frac{u^m u^m}{r} V r\,dr\,dz =$$

$$= - \iint\limits_D \mu(v_r^* V_r + v_z^* V_z) r\,dr\,dz - \iint\limits_D \mu \frac{v^*}{r^2} V r\,dr\,dz, \tag{15.35}$$

$$\iint\limits_D \frac{w^* - g^m}{\tau} W r\,dr\,dz = -\iint\limits_D p_z^m W r\,dr\,dz -$$

$$- \iint\limits_D \mu(w_r^* W_r + w_z^* W_z) r\,dr\,dz. \tag{15.36}$$

Здесь предполагается, что на границе Γ выполнены условия

$$U\big|_\Gamma = 0, \quad V\big|_\Gamma = 0, \quad W\big|_\Gamma = 0. \tag{15.37}$$

15.3 Реализация алгоритма на языке FreeFem++

Полный код программы на языке FreeFem++ имеет вид (ср. п. 8.3)

```
real r1=1.0, r2=2.0, L=1.0;
int m;
border a1(t=0,1){ x=r2*t+r1*(1-t); y=-L;           label=1;};// bottom
border a2(t=0,1){ x=r2;            y=-L+2*L*t;     label=2;};// outer
border a3(t=0,1){ x=r1*t+r2*(1-t); y=L;            label=3;};// top
border a4(t=0,1){ x=r1;            y=-L+2*L*(1-t); label=4;};// inner
int n=3;
mesh Th = buildmesh(a1(10*n)+a2(10*n)+a3(10*n)+a4(10*n));
plot(Th, wait=1);
fespace Vh(Th, P2);
real hinit = 0.08; // начальный размер сетки
Vh hh = hinit;     // FE-функция для задания размера сетки
Th = adaptmesh(Th, hh, IsMetric=1, splitpbedge=1, nbvx=10000);
plot(Th, wait=1);
real mu = 0.01;
```

```
real dt = 0.01,  t = 0;
Vh w, u, v, p, q, psi, F;
w = 0;   u = 0;   v = 0;   p = 0;   q = 0;
psi = 0;
real area = int2d(Th)(1.0*x);
real Vr = 3.0;
Vh um, wm, pm, vm;
Vh f, g, h;
real meandiv, meanpq;
problem pb4v(v, F, init=m, solver=UMFPACK)     // (15.35)
  = int2d(Th)( x*(v*F/dt + mu*(dx(v)*dx(F)+dy(v)*dy(F)+v/x^2*F)) )
    -int2d(Th)( (x*h/dt + um*vm)*F  )
    + on(1,2,3,v = 0) + on(4,v=Vr);
problem pb4u(u, F, init=m, solver=UMFPACK)    // (15.34)
  = int2d(Th)( x*(u*F/dt + mu*(dx(u)*dx(F)+dy(u)*dy(F)+u/x^2*F)) )
    - int2d(Th)( x*(f/dt-dx(p)+vm^2/x)*F )
    + on(1,2,3,4,u = 0);
problem pb4w(w, F, init=m, solver=UMFPACK)     // (15.36)
  = int2d(Th)(x*(w*F/dt + mu*(dx(w)*dx(F)+dy(w)*dy(F)))  )
    - int2d(Th)( (g/dt-dy(p))*F )
    + on(1,2,3,4, w = 0);
problem pb4p(q, F, init=m, solver=UMFPACK)
  = int2d(Th)( x*(dx(q)*dx(F)+dy(q)*dy(F)) )
    + int2d(Th)( (x*dx(u)+ u + x*dy(w)-meandiv)*F/dt );
problem pb4psi(psi, F, init=m, solver=UMFPACK)
  = int2d(Th)(x*(dx(psi)*dx(F)+dy(psi)*dy(F) +psi/x^2*F ))
    - int2d(Th)(x*(dx(w)- dy(u))*F) + on(1,2,3,4, psi=0);
for(m=1; m<100000; m++)
{
  t = t+dt;
  um = u;   wm = w;   pm = p;   vm = v;
  f = convect([u,w],-dt,um);
  g = convect([u,w],-dt,wm);
  h = convect([u,w],-dt,vm);
  pb4v;
  pb4u;
  pb4w;
  meandiv = int2d(Th)(x*dx(u) + u + x*dy(w))/area;
  pb4p;
  meanpq = int2d(Th)(x*q)/area;
  p = pm+q-meanpq;
  u = u - dx(q)*dt;
  w = w - dy(q)*dt;
  pb4psi;
  // организация вывода данных в файлы через 10 шагов
     if ((m%10)==0)
     plot(psi, fill=0,cmm=+t,value=1);
 }
```

15.4 Вычислительный эксперимент

Результаты расчетов приведены на рис. 15.1 для следующих значений параметров: $r_1 = 1{,}0$, $r_2 = 2{,}0$, $L = 1{,}0$, $r_1\omega_1 = 3{,}0$, $\omega_2 = 0$, $\mu = 0{,}01$, $\tau = 0{,}01$ при $t = 1{,}0;\ 3{,}0;\ 5{,}0;\ 9{,}0$. Естественно, на рис. 15.1 изображена область

$$r_1 < r < r_2, \quad -L < z < L.$$

Хорошо видно, что с течением времени, в зазоре между цилиндрами образуются так называемые вихри Тейлора.

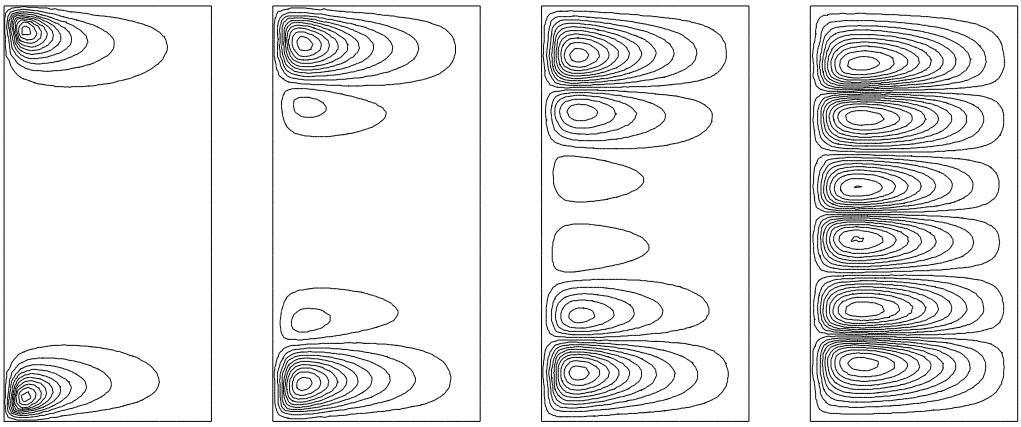

Рис. 15.1. Изолинии функции тока $\psi(x, y)$ в моменты времени $t = 1{,}0;\ 3{,}0;\ 5{,}0;\ 9{,}0$

Часть III
Конструкции языка FreeFem++ (краткий обзор)

Материалы этой части носят справочный характер и могут существенно облегчить понимание приведенных в предыдущих главах кодов программ и помочь создавать более сложные алгоритмы вычислений.

В гл. 16 содержится достаточно подробное описание синтаксиса языка FreeFem++. Многие конструкции языка полностью аналогичны C++ (запись арифметических выражений, организация циклов, условных операторов и т. п.). Помимо этого имеется большой набор ключевых слов, непосредственно предназначенных для решения задач МКЭ — `solver` (выбор метода решения систем линейных уравнений), `matrix` (разреженные матрицы), `mesh`, `buildmesh` (идентификация и создание сетки) и т. д.

Вопросам, связанным с триангуляцией области, посвящена гл. 17. В ней описаны различные приемы создания сеток и управления ими в процессе расчета. Особенно важно изучение материалов гл. 17 в случае, когда FreeFem++ предполагается использовать для решения нестандартных задач — в областях сложной формы, в областях с деформируемой границей, в случаях сильной пространственной неоднородности решений.

Базисные функции, используемые для построения решения, подробно описаны в гл. 18. Правильный выбор базисных функций существенно влияет на точность проводимых расчетов и тесно связан со свойствами разыскиваемых решений (например, с их гладкостью). Глубокое понимание способов интерполяции финитными функциями также важно для интерпретации результатов. В гл. 18 приведены квадратурные формулы, применяемые в FreeFem++ при вычислении матриц систем линейных уравнений.

В гл. 19 перечислены методы решения систем линейных алгебраических уравнений и указаны используемые при этом параметры — точность, способ преобразования матриц и т. д. Приводится также описание нестандартных приемов построения решения при помощи прямого вычисления билинейных и линейных форм. Эта глава тесно связана с гл. 20, в которой описаны способы решения краевой задачи на собственные значения.

Различные способы визуализации результатов расчетов даны в гл. 21. Использование специальных средств визуализации позволяет сохранять результаты в виде текстовых и графических файлов и получать более полную информацию о решении — строить линии уровня и т. п.

Глава 16

Синтаксис

Программа на языке FreeFem++ представляет собой последовательность операторов, разделенных точкой с запятой.

Каждая используемая переменная (за исключением зарезервированных глобальных переменных, см. п. 16.1.2) должна быть описана в виде:

```
ТипПеременной ИмяПеременной;
```

Здесь `ТипПеременной` — один из имеющихся в FreeFem++ типов (см. ниже);
`ИмяПеременной` — буквенно-цифровая строка (FreeFem++ различает прописные и строчные буквы; использование символа «_» не допускается). Несколько переменных одного типа можно описывать через запятую.

При описании переменной возможно присвоение ей начального значения

```
ТипПеременной ИмяПеременной = начальное значение;
```

В следующем фрагменте описаны переменные целого типа `i` и `n`, причем переменная `n` проинициализирована при описании значением 10

```
int i, n = 10;
i = 15;
cout << i << "  " << n << endl; // вывод на экран
```

16.1 Типы данных

Язык FreeFem++ имеет обширный набор типов данных: целочисленный (`int`), вещественный (`real`), строковый (`string`), массивы (например, вещественный массив: `real[int]`), 2D конечноэлементные сетки (`mesh`), 2D пространства конечных элементов (`fespace`), аналитические функции (`func`), массивы конечноэлементных функций (`func[basic type]`), линейные и билинейные операторы (`varf`), разреженные матрицы (`matrix`), векторы и т. д.

Пример 16.1. Следующая программа демонстрирует описание переменных простых типов и массивов, а также заполнение массивов некоторыми значениями, посчитанными по заданным формулам

```
int i, n=10;
real[int] xx(n), yy(n);
for (i=0; i<n; i++)
{ xx[i]= cos(i*pi/n);   yy[i]= sin(i*pi/n);
  //  cout << xx[i] << "  " << yy[i] << endl; // вывод на экран
}
```

✓. *Переменные, описанные внутри текущего блока (блоком в FreeFem++ является последовательность операторов, ограниченная фигурными скобками* `{...}`*), являются локальными, т. е. после закрытия блока (...}) они уничтожаются. Исключением являются* `fespace`*-переменные.*

Например, в результате работы следующего кода

```
 { // начало блока
   int i=5, n=10;
   cout << "i=" << i << "  n=" << n << endl;
 } // конец блока
 int i=3, n=18;
 cout << "i=" << i << "  n=" << n << endl;
```

на экране дисплея будет выведено

```
i=5  n=10
i=3  n=18
```

Результатом работы программы

```
real r = 7.5;
cout << "r=" << r << "  ";
{ // начало блока
  real r = 2; // новая переменная r
  cout << "r=" << r << "  ";
} // конец блока
// r возвращается прежнее значение 7.5
cout << "r=" << r << endl;
```

будет строка

```
r=7.5  r=2  r=7.5
```

Компиляция следующего кода вызовет ошибку в 6 строке, т. к. `Vh` является именем глобальной `fespace`-переменной.

```
mesh Th = square(5,5); // генерация сетки 5 на 5 в области [0,1]x[0,1]
fespace Vh(Th, P1);    // определение пространства конечных элементов
Vh u = x + exp(y);     // определение FE-функции на пространстве Vh
plot(u, wait=true);
{ // начало блока
  fespace Vh(Th, P1);    // ошибка
} // конец блока
```

16.1.1 Основные типы данных

`bool` — логический (булев) тип данных. Переменные этого типа могут принимать значения `true` (или 1) и `false` (или 0). Например,

```
bool Bo = true,  B = 0,  Bol = (1<2);
```

Операции, применимые к данным логического типа, это операции сравнения: `==` (равно), `!=` (не равно), `>`, `<`, `>=`, `<=` и логические операции: `!` (отрицание), `||` (дизъюнкция), `&&` (конъюнкция).

В следующем примере переменная `MyBol` логического типа проинициализирована значением выражения, проверяющего принадлежность точки x интервалу $(a, b]$.

```
int a=-3, b=3, x=-2;
bool MyBol = ((x>a) && (x<=b));
cout << "x in (a,b] = " << MyBol << endl;
```

`int` — целочисленный тип данных.

`string` — строковый тип данных. Под строкой понимается набор символов, заключенный в двойные кавычки. Например, `"This is a string"`.

`real` — вещественный тип данных (`-3.21`, `12.345`, `7.5e-1`, `7.5E+2`).

`complex` — определяет комплексные числа типа $a+ib$, где $i = \sqrt{-1}$ (для мнимой единицы используется обозначение `1i`.). Допустимые операции над данным типа **complex**: арифметические операции (`+`, `-`, `*`, `/`). Например,

```
  complex a = 1i, b = 2.5 + 3i;
  cout << "a + b  = " << a + b << endl;
  cout << "2 * b  = " << 2 * b << endl;
```

В результате выполнения получим

```
a + b  = (2.5,4)
2 * b  = (5,6)
```

`ofstream` объявляет выходной файл (файл для записи данных, см. 16.8).

`ifstream` объявляет входной файл (файл для чтения данных, см. 16.8).

`real[int]` — массив вещественных данных с целочисленными индексами. Имеется также возможность работы с массивами вида: `int[int]`, `real[string]`, `string[string]`.

```
int[int] a(3);
a[0]=1;  a[1]=3;  a[2]=2;
// I способ вывода массива на печать
for(int i=0; i<3; i++)
  {
    cout << "a[" << i << "]=" << a[i] << "  ";
  }
// II способ вывода массива на печать
cout << endl << "a=" << a << endl;
```

На экране дисплея получим

```
1  3  2
a=3
       1       3       2
```

`func` объявляет функцию с аргументами по умолчанию `x` и `y`. Например,

```
func F1 = y*cos(x) + x*sin(y);
```

`mesh` генерация сетки (триангуляция области) (см. гл. 17);

`fespace` определяет тип пространства конечных элементов (см. гл. 18);

`problem` определяет слабую формулировку задачи в частных производных без ее решения (см. гл. 19);

`solve` определяет слабую формулировку задачи в частных производных и решает ее (см. гл. 19);

`varf` определяет линейные и билинейные формы (см. гл. 19);

`matrix` определяет разреженную матрицу (см. гл. 19).

16.1.2 Глобальные переменные

Имена `x`, `y`, `z`, `label`, `region`, `P`, `N`, `nu_triangle` являются в языке FreeFem++ зарезервированными словами. Опишем их подробнее:

`x` — координата x текущей точки (вещественное значение);

`y` — координата y текущей точки (вещественное значение);

`z` — координата z текущей точки (вещественное значение), имя зарезервировано для использования в будущем;

`label` — позволяет задавать числовую метку границы (подробнее см. [1]);

`region` — возвращает номер области, которой принадлежит текущая точка (x, y) (целочисленное значение);

`P` — дает координаты текущей точки (значение в $\mathbb{R}^2$). Использование операторов `P.x`, `P.y` позволяет получить значения x и y текущей точки. Значение `P.z` также является зарезервированным;

`N` — позволяет получить компоненты единичного вектора внешней нормали к границе, определенной при помощи `border`. Если текущая точка (x, y) принадлежит границе, то `N.x` и `N.y` являются x и y компонентами вектора нормали. Имя `N.z` зарезервировано;

`lenEdge` — возвращает длину ребра треугольника;

`hTriangle` — возвращает размер текущего треугольника;

`nuTriangle` — возвращает индекс текущего треугольника (`integer`);

`nuEdge` — возвращает индекс текущей ребра в треугольнике (`integer`);

`nTonEdge` — возвращает количество треугольников, прилегающих к текущему ребру треугольника (`integer`);

`area` — площадь текущего треугольника (`real`);

`cout` — оператор вывода (стандартное выходное устройство — консоль по умолчанию, `ostream`, в MS Windows только консоль);

`cin` — оператор ввода (стандартное входное устройство — клавиатура по умолчанию, `istream`, в MS Windows работает, начиная с версии FreeFem++2.24);

`endl` — символ конца строки в операторах ввода/вывода;
`true` означает «true» в логическом выражении;
`false` означает «false» в логическом выражении;
`pi` — приближенное значение числа π.

16.2 Системные команды

Просмотреть все типы, операторы и функции языка FreeFem++ позволяет команда:

```
dumptable(cout);
```

Выполнить «внешнюю» системную команду (`shell command`), например, запустить какую-либо программу на выполнение можно с помощью строки

```
exec("shell command");
```

В аргументе `exec` необходимо указать имя файла, как правило, с указанием полного пути файла.

Имя файла можно указывать и без пути, если выполняемый файл лежит в каталоге, из которого запущено приложение, либо в текущем каталоге, либо в системном каталоге Windows (System32 для Windows NT), либо в каталоге Windows.

Например, для запуска на выполнение файла «devcpp.exe» из каталога `c:\dev-cpp` надо написать

```
exec("c:\\dev-cpp\\devcpp.exe");
```

Другой пример использования команды `exec` — запуск служебной программы Windows «калькулятор» (исполняемый файл — `calc.exe`)

```
exec("%SystemRoot%\\system32\\calc.exe");
```

(здесь `%SystemRoot%` — путь на каталог с ОС Windows).

16.3 Математические операции

Для целых чисел операции `+`, `-`, `*` означают, соответственно, обычное арифметическое суммирование, вычитание и умножение, `/` и `%` — вычисление частного и остаток от деления первого выражения на второе. Для вычисления минимального и максимального значений из двух целых чисел a, b используются функции `max(a,b)` и `min(a,b)`.

Вычисление степени (a^b, a, b — целые числа) — `a^b`.

Операция `a ? b : c` — означает проверку выражения `a`: если оно истинно, то результату присваивается значение выражения `b`, иначе — `c`

```
cout << " min(a,b) = " << (a < b ? a : b) << endl;
```

Операции `+`, `-`, `*`, `/`, `%`, `^`, `max` и `min` применимы также к данным вещественного типа. При использовании операции `%` остаток от деления находится в результате деления целых частей вещественных чисел:

```
real a=5.1, b=2.85;
cout << a << " % " << b << " = " << a % b << endl;
```

Результат работы данного кода:

```
5.1 % 2.85 = 1
```

В силу вложения

$$\texttt{bool} \subset \texttt{int} \subset \texttt{real} \subset \texttt{complex},$$

операции `+`, `-`, `*`, `/`, `^` также применимы и к комплексным типам данных, а применение операций `%`, `max` и `min` даст неверный результат. При записи комплексных чисел $a + ib$, $i = \sqrt{-1}$ для обозначения мнимой единицы используется `1i`. Например,

```
complex z1 = a + b * 1i;
```

Число, сопряженное данному комплексному числу `z`, можно получить с помощью функции `conj(z)`. Например,

```
real a = 5.1, b = 2.85;
complex z = a + b * 1i;
cout << z << " -- " << conj(z) << endl;
```

Результат работы данного кода:

```
(5.1,2.85) -- (5.1,-2.85)
```

Вещественную и мнимую части комплексного числа `z`, можно получить с помощью функций `real(z)` и `imag(z)`. Например,

```
real a = 5.1, b = 2.85;
complex z = a + b * 1i;
cout << "Re z = " <<  real(z) << endl;
cout << "Im z = " <<  imag(z) << endl;
```

Результат работы данного кода:

```
Re z = 5.1, Im z = 2.85.
```

Имеется возможность выделения модуля комплексного числа `z` и его аргумента при помощи функций `abs(z)` и `arg(z)`. При помощи функции `polar(abs(z),arg(z))` можно задавать комплексное число по его модулю и аргументу. Например,

```
1 real absZ = 1, argZ = pi/3;
2 complex z;
3 z = polar(absZ, argZ);
4 cout << "z = " <<  z << endl;
5 cout << "abs(z) = " <<  abs(z) << endl;
6 cout << "arg(z) = " <<  arg(z) << endl;
7 cout << "pi/3   = " <<  pi/3   << endl;
```

Результат работы данного кода:

```
(0.5,0.866025), abs(z) = 1, arg(z) = 1.0472, pi/3 = 1.0472
```

16.4 Функции одной переменной

16.4.1 Встроенные функции

16.4.1.1 Элементарные функции

Степенная функция x^y: `x^y`.
Экспоненциальная функция e^x: `exp(x)`.
Логарифмические функции $\ln x$ и $\log_{10} x$: `log(x)` и `log10(x)`.
Тригонометрические функции $\sin x$, $\cos x$, $\operatorname{tg} x$: `sin(x)`, `cos(x)`, `tan(x)` (аргументы — радианная мера угла).

✓. *В функциях* `exp(z)`, `cos(z)`, `sin(z)`, `log(z)` *аргумент* z *может быть комплексным,* $z = x + iy$. *При этом будут возвращаться значения:*

$$\texttt{exp(z)} = e^x(\cos y + i \sin y) \quad \textit{(формула Эйлера)},$$

$$\texttt{log(z)} = \ln|z| + i \arg z.$$

Напомним, что

$$\sin iy = i \operatorname{sh} y, \quad \cos iy = \operatorname{ch} y.$$

В частности,

$$\sin z = \sin(x + iy) = \sin x \cos iy + \cos x \sin iy = \operatorname{ch} y \sin x + i \operatorname{sh} y \cos x.$$

Обратные тригонометрические функции $\arcsin x$, $\arccos x$, $\arctan x$:

`asin(x)`, `acos(x)`, `atan(x)`.

Функция `atan2(x,y)` применяется для вычисления главного значения $\arctan y/x$. Знаки аргументов используются для определения квадранта, которому соответствует возвращаемое значение $\arctan y/x$.

✓. *Функцию* `atan2(x,y)` *можно, например, использовать при определении полярных координат по известным декартовым координатам. Пусть* $x = r\cos\theta$, $y = r\sin\theta$. *Тогда* $r = \sqrt{x^2 + y^2}$, $\theta =$ `atan2(x,y)`. *Использование выражения* $\theta =$ `atan(y/x)` *приведет к ошибкам при обратном восстановлении величин* x *и* y *по значениям* r *и* θ.

Гиперболические функции

$$\operatorname{sh} x = \frac{e^x - e^{-x}}{2}, \quad \operatorname{ch} x = \frac{e^x + e^{-x}}{2}, \quad \operatorname{th} x = \frac{\operatorname{sh} x}{\operatorname{ch} x}$$

в языке FreeFem++: `sinh(x)`, `cosh(x)`, `tanh(x)`.

Функции `asinh(x)`, `acosh(x)` и `atanh(x)` соответствуют математическим функциям (символ -1 означает обратную функцию)

$$\operatorname{sh}^{-1} x = \operatorname{Arsh} x = \ln(x + \sqrt{x^2 + 1}),$$

$$\operatorname{ch}^{-1} x = \operatorname{Arch} x = \ln(x + \sqrt{x^2 - 1}), \quad (|x| \geqslant 1),$$

$$\mathrm{th}^{-1}\, x = \mathrm{Arth}\, x = \frac{1}{2} \ln \frac{1+x}{1-x} \quad (|x| < 1).$$

`floor(x)` — целая часть вещественного числа $[x]$ (напомним, что целой частью вещественного числа x называется целое число, не превышающее число x, например, $[3.1] = 3$, $[-3.1] = -4$, $[0.1] = 0$, $[-0.1] = -1$).

`ceil(x)` — округление вещественного числа до наименьшего целого, не меньшего, чем x. Примеры действия функции `ceil(x)`: `ceil(2.51)` $= 3$; `ceil(-2.51)` $= -2$; `ceil(-2.11)` $= -2$; `ceil(2.19)` $= 3$.

`rint(x)` — округление вещественного числа до ближайшего целого (возвращаемое значение `real`). Примеры действия функции `rint(x)`:

$$\texttt{rint(2.51)} = 3; \quad \texttt{rint(-2.51)} = -3; \quad \texttt{rint(-2.11)} = -2;$$

$$\texttt{rint(2.19)} = 2; \quad \texttt{rint(-4.0)} = -4; \quad \texttt{rint(4.0)} = 4.$$

В языке FreeFem++ можно создавать любые элементарные функции. Под *элементарными функциями* в FreeFem++ понимаются функции, состоящие из степенных, показательных, логарифмических, тригонометрических, обратных тригонометрических функций и функции, полученные из них с помощью четырех арифметических операций `+`, `-`, `*`, `/`, а также полученные при помощи операции суперпозиции $f(g(x))$.

Пример 16.2. Сформируем границу области в виде кардиоиды с помощью элементарных функций.

```
real b = 1.0,    a = b;    //   кардиоида
// real b = 4.0, a = 3.0;  //   улитка Паскаля без самопересечения
// real b = 1.0, a = 3.0;  //   улитка Паскаля с самопересечением
func real phix(real t)
{
  return a*cos(t)*cos(t) + b*cos(t);
}
func real phiy(real t)
{
  return a*sin(t)*cos(t) + b*sin(t);
}
border C(t=0,2*pi) { x=phix(t); y=phiy(t); }
plot(C(50), wait=1);   // рисование границы области
```

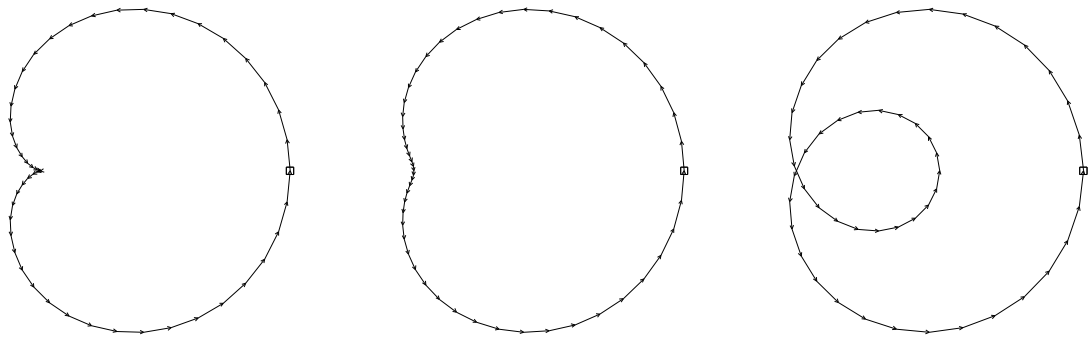

Рис. 16.1. Кардиоида, улитка Паскаля (без самопересечения и с самопересечением)

16.4.1.2 Случайные функции

В FreeFem++ реализован быстрый генератор случайных чисел с периодом, равным максимально возможному в 32-разрядных системах целому числу $2^{219937} - 1$.

`randint32()`, `randint31()` генерируют, соответственно, 32 и 31-разрядные целые числа без знака;

`randreal1()`, `randreal2()`, `randreal3()` генерируют вещественные числа из интервалов $[0, 1]$, $[0, 1)$ и $(0, 1)$, соответственно (32-разрядная разрешающая способность);

`randres53()` генерирует вещественное число из $[0, 1)$ (53-разрядная разрешающая способность);

`randinit(seed)` инициализация генератора случайных чисел на основе 32-разрядного целого числа `seed`, которое может быть нулевым ([1]).

16.4.1.3 Дополнительные математические функции

В FreeFem++, начиная с версии 2.17, имеется библиотека дополнительных математических функций.

`j0(x)`, `j1(x)`, `jn(n,x)`, `y0(x)`, `y1(x)`, `yn(n,x)` — функции Бесселя первого и второго рода.

Функции `j0(x)`, `j1(x)` и `jn(n, x)` вычисляют функции Бесселя первого рода нулевого, первого и n порядков, соответственно.

Функции `y0(x)`, `y1(x)` и `yn(n,x)` вычисляют функции Бесселя второго рода нулевого, первого и n порядков, соответственно, для целых положительных значений x.

Функция `tgamma(x)` вычисляет Г-функцию от x. `lgamma(x)` вычисляет натуральный логарифм абсолютного значения от $\Gamma(x)$.

Функция `erf(x)` вычисляет функцию ошибок (интеграл ошибок)

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Для вычисления дополнительной функции ошибок ($\operatorname{erfc} x = 1 - \operatorname{erf} x$) используется `erfc(x)`.

16.5 Функции двух переменных

16.5.1 Задание функций с помощью формул

Так как в языке FreeFem++ `x` и `y` являются зарезервированными словами (см. п. 16.1.2), то функция $z = f(x, y)$ определяется как функция без параметров

```
func f = sin(x) * cos(y);
```

Тип функции определяется типом входящих в нее выражений.

В FreeFem++ *элементарной функцией двух переменных* называется функция, составленная из элементарных функций $f(x)$ и/или $g(y)$ с помощью четырех арифметических операций (+, -, *, /) и операции суперпозиции. Например, элементарную функцию h можно определить следующим образом

```
func f = sin(x) * cos(y);
func g = exp(x) + 1;
func h = max(-0.5, 0.1 * log(f^2 + g^2));
```

16.5.2 Конечноэлементные функции

Конечноэлементная функции (FE-функция) — есть элемент пространства конечных элементов (вещественных или комплексных) (см. гл. 18). Задание формулы-функции f на FE-пространстве может иметь вид:

```
1  func f = x^2 + y^2;
2  mesh Th = square(20,20,[2*x,2*y]);  //  квадрат [0,2]x[0,2]
3  fespace Vh(Th, P1);
4  Vh fh = f;            //  fh -- проекция f в Vh (веществ. значение)
5  plot(fh);
6  func zf = f * exp(x+1i*y);
7  Vh<complex> zh = zf;  //  zh -- проекция zf (в компл. FE-простр. Vh)
```

Заметим, что команда `plot` применима только для вещественных FE-функций, т. е. использование вызова `plot(zh)` приведет к ошибке.

О конструкции `fh` ($= f_h$) см. в гл. 18.

Приведем пример создания комплексной функции и FE-пространств:

```
1  mesh Th = square(20,20);
2  fespace Vh(Th,P2);       // определение пространства конечных элементов
3  func z = x+y*1i;         // z = x + iy
4  func f = imag(sqrt(z));  // f = Im(sqrt(z))
5  func g = real(sqrt(z));  // g = Re(sqrt(z))
6  Vh fh = f;
7  plot(fh, wait=1);        // линии уровня функции f
8  Vh gh = g;
9  plot(gh, wait=1);        // линии уровня функции g
10 plot(fh, gh, wait=1);    // линии уровня функций f и g
```

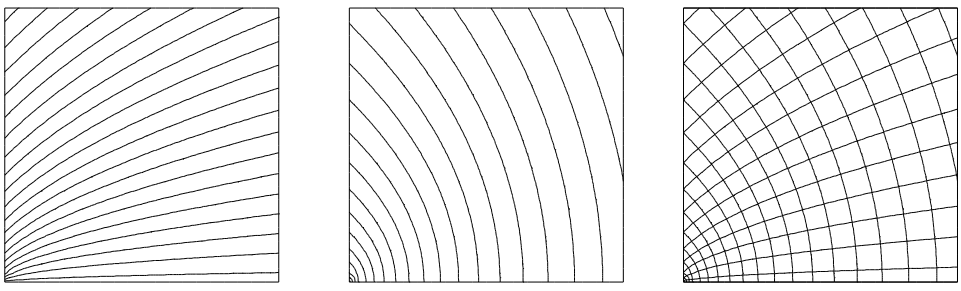

Рис. 16.2. Линии уровня функций $f = \operatorname{Im}\sqrt{z}$, $g = \operatorname{Re}\sqrt{z}$, $z \in \mathbb{C}$

16.6 Оператор условного перехода

В языке FreeFem++ имеется оператор условного перехода `if`, предназначенный для выполнения тех или иных действий в зависимости от результата проверки некоторого условия.

Синтаксис условного оператора

```
if (IFCondition)
  { Block1; }
else
  { Block2; }
```

Здесь `IFCondition` — условие; если условие выполнено, то выполняются операторы из `Block1`, иначе — операторы из `Block2`.

Ветвь `else { Block2; }` может отсутствовать

```
if (IFCondition)
  { Block1; }
```

В этом случае, если условие `IFCondition` не выполняется, то управление передается следующему оператору программы.

Условие представляет собой логическое выражение, состоящее из операндов и знаков операций сравнения (`==`, `!=`, `>`, `<`, `>=`, `<=`). Составные условия конструируются с помощью логических операций (`!` — отрицание, `||` — дизъюнкция, `&&` — конъюнкция). В следующем коде при выполнении равенства $a = 3$ значение переменной b увеличивается в два раза, в противном случае — в три раза.

```
int a=3, b=-7;
if (a==3)
  { b=b*2; }
else
  { b=b*3; }
cout << "b=" << b << endl;
```

Проверка принадлежности точки x интервалу $(a, b]$:

```
int a=-3, b=3, x=-3;
if ((x>a) && (x<=b))
  { cout << "x in (a,b]" << endl; }
else
  { cout << "x not in (a,b]" << endl; }
```

16.7 Циклы

В языке FreeFem++ имеется два вида циклов: `for` — цикл с параметром и `while` — цикл с условием. В теле любого из указанных циклов могут быть использованы ключевые слова `break` и `continue`.

16.7.1 Цикл с параметром

Синтаксис цикла `for`

```
for (FORInitialization; FORCondition; FORChange)
  { BLOCK }
```

Здесь `FORInitialization` — инициализация параметра цикла;
`FORCondition` — условие продолжения работы цикла;
`FORChange` — изменение параметра цикла;
`BLOCK` содержит операторы тела цикла.

Пока условие `FORCondition` выполняется цикл продолжает работать.

Например, вычисление суммы первых десяти целых чисел можно записать в виде

```
int sum = 0;
for (int i=1; i<=10; i++)
  { sum += i; }  // или   { sum = sum+i; }
cout << "Sum = " << sum << endl;
```

16.7.2 Цикл с условием

Синтаксис цикла `while`

```
while (WhileCondition)
  { BLOCK }
```

Цикл выполняется до тех пор, пока условие `WhileCondition` не станет ложным. `BLOCK` содержит операторы тела цикла и оператор изменения параметра цикла.

Вычисление суммы первых десяти целых чисел с помощью цикла `while` запишется в виде

```
int i = 1, sum = 0;
while (i<=10)
  { sum += i; i++; }
cout << "Sum = " << sum << endl;
```

Если оператор в теле цикла один, то его можно не окружать скобками `{ }`. Телом цикла в приведенном ниже коде является только

```
cout << i << "  ";
```

Оператор вывода `cout << "\n";`, переводящий курсор на новую строку выполнится только один раз после окончания работы цикла `for`

```
for (int i=0; i<10; i=i+1)
  cout << i << "  ";
cout << "\n";
```

Выход из любого цикла можно также осуществить с помощью зарезервированного слова `break`. Например, если программу вычисления первых десяти целых чисел изменить следующим образом

```
int i = 1, sum = 0;
 while (i<=10)
   { sum += i; i++;
     if ( sum > 30) break; }
 cout << "Sum = " << sum << endl;
```

то вычисление суммы прекратится, как только сумма превысит число 30.

Зарезервированное слово `continue` позволяет пропустить выполнение операторов, входящих в тело цикла, от слова `continue` до конца цикла. В нижеприведенном коде программы оператор вывода `cout` начнет выполняться только после того, как параметр цикла `i` станет больше 10

```
for (int i=0; i<20; i++)
  { if (i<10) continue;
    cout << "i = " << i << endl; }
```

16.8 Операторы ввода/вывода. Файлы

Синтаксически ввод/вывод различных значений подобен соответствующим операторам языка C++. При вводе/выводе используются операторы `cout` (для вывода информации на экран дисплея) и `cin` (для ввода данных с клавиатуры, при использовании FreeFem++ в Windows не работает). Операции `<<` — операция вставки (вставляет символы в выходной поток) и `>>` — операция извлечения (извлекает данных из входного потока, присваивая значение указанной переменной).

В операторе вывода `cout` часто используется символ конца строки — `endl` (переводит курсор в начало следующей строки). Можно также использовать специальный символ `\n`. Данный символ должен быть заключен в двойные кавычки, если он используется сам по себе и без кавычек, если он используется внутри других двойных кавычек:

```
cout << " Rostov \n University" << "\n";
```

Для записи данных в файл (чтения данных из файла) надо объявить новые переменные

```
ofstream ofile("имя файла");  ( ifstream ifile("имя файла"); )
```

и использовать `ofile`/`ifile` вместо `cout`/`cin`.

Для добавления в существующий файл указывается параметр `append`:

```
ofstream ofile("имя файла", append);
```

Следует учитывать, что при выходе из блока `{...}` файл закрывается.

Пример 16.3. В результате работы следующей программы организуется текстовый файл `DFile.txt`, содержащий две строки с числами 100 и 200.

```
int i = 100, j;
{ ofstream MyFile("DFile.txt");
  MyFile << i << "\n"; // запись в файл DFile.txt числа 100
};
{ ifstream MyFile("DFile.txt");
  MyFile >> j;         // чтение из файла DFile.txt в переменную j
  cout << "j = " << j << endl;
}
i = 2*j;
cout << "i = " << i << endl;
{ ofstream MyFile("DFile.txt", append);
  MyFile << i << "\n"; // добавление в конец DFile.txt числа 200
};
```

Приведем некоторые функции для управления форматом вывода (`f` — имя файловой переменной или оператор `cout`):

`int nold = f.precision(n)` — задает число цифр в вещественном числе, имеющем формат вывода с фиксированной точкой, и число цифр после запятой в вещественном числе, имеющем научный формат вывода. Относится к числам, участвующим в выходных потоках `f`. Например, в результате работы программы

```
real i = 10.157;
cout << "i = " << i << endl;
int nold = cout.precision(4);
cout << "i = " << i << endl;
```

на экране появятся две строки: `i = 10.157` и `i = 10.16`;

`f.scientific` — устанавливает экспоненциальный (научный) формат вывода чисел с плавающей точкой (±`d.dddE±dd`, где `d` — цифра);

`f.fixed` — устанавливает вывод вещественных чисел в виде с фиксированной точкой (±`d.ddd`);

`f.showpos` — добавляет знак плюс (+) перед числом;

`f.noshowpos` — отменяет опцию `showpos`;

`f.default` — восстанавливает все значения по умолчанию.

16.9 Массивы

16.9.1 Одномерные массивы

В массиве может храниться набор однотипных данных. В FreeFem++ существует два типа массива: *вектор* с целочисленными индексами и *вектор* со строковыми индексами. В случае массивов с целочисленными индексами необходимо заранее задать его размер (т. е. количество элементов

в массиве; нумерация индексов всегда начинается с нуля). Например,

```
real[int] A(5);  // массив из 5 вещ. чисел (A[0],..., A[4])
```

Неверное описание массива (не указан размер массива)

```
real [int] A;
```

Пример 16.4. Приведем различные способы инициализации массивов.

```
1 real [int] F = [0.5, 1, 0, 4];     // инициализация при описании
2 real [int] A(5), B(5), C(5), D(4); // описание массивов из вещ. чисел
3 A = 1.2;                           // все элементы массива A равны 1.2
4 A[0] = 2;                          // первый элемент массива A равен 2
5 B = C = -1.3;            // все элементы массивов B и C равны -1.3
6 C(0:2) = 5;              // эл-ты массива C с номерами 0, 1, 2 равны 5
7 D = [5, -4, 0.5, 35];
```

Обратим внимание, что способ инициализации массива при его описании не требует указания количества элементов массива (см. строку 1).

Операции, позволяющие получать информацию о массиве:
`A.n` — количество элементов массива;
`A[0]`, `A[A.n-1]` — первый и последний элементы массива;
`A.min`, `A.max` — минимальный и максимальный элементы массива;
`A.sum` — сумма элементов массива;
`A.l1` — $\|A\|_1 = \sum_i |a_i|$; `A.l2` — $\|A\|_2 = \sqrt{\sum_i a_i^2}$;
`A.linfty` — $\|A\|_\infty = \max_i |a_i|$; `A'*A` — (A^T, A).

Пример 16.5. Приведем пример, демонстрирующий работу с элементами массива и массивом в целом.

```
1 real [int] A(5);    // описание массива из 5 веществ. чисел
2 A = 1.5;            // инициализация массива -- все элементы равны 1.5
3 A[0] = A[0]*2;      // первый элемент равен 3
4 A[2] = 0.0;         // третий элемент равен 0
5 cout << "Size of array : " << A.n  << endl;
6 cout << "First : " << A[0] << endl;
7 cout << "Last  : " << A[A.n-1] << endl;
8 cout << "(Min,Max) = ("  << A.min << "," << A.max << ")" << endl;
9 cout << "Sum = " << A.sum << endl;
```

Результат работы программного кода:

```
Size of array : 5
First : 3
Last  : 1.5
(Min,Max) = (0,3)
Sum = 7.5
```

Размер массива можно изменять уже после его описания при помощи `resize`.

```
real [int] A(5);
A(0:2) = 1.5;     A(3:4) = 0.5;
cout << "Array : " << A  << endl;
cout << "First : " << A[0] << "  Last : " << A[A.n-1] << endl;
A.resize(8);   // изменение размера массива
A(5:7) = -0.5; // инициализация новых элементов, старые элементы те же
cout << "New array : " << A << endl;
cout << "First : " << A[0] << "  Last : " << A[A.n-1] << endl;
```

Результат работы:

```
Array : 5
        1.5     1.5     1.5     0.5     0.5
First : 1.5  Last : 0.5
New array : 8
        1.5     1.5     1.5     0.5     0.5
        -0.5    -0.5    -0.5
First : 1.5  Last : -0.5
```

Пример описания и использования массива с индексами типа `string`:

```
int [string] B;
B["a"] = 5;
cout << B["a"] << " " << B["+"] << endl;
```

Пример 16.6 (**Работа с массивами**).

```
int m = 5;
int[int] a(m), b(m), c(m), d(m);
a = 1;  a(2:3) = 0;  b = 2;  c = 3;
d = ( a ? b : c );
// функция для печати элементов массива
func string PrintAr(int[int] & Ar)
{  for (int i=0; i<Ar.n; i++)  cout << Ar[i] << "  ";
   return endl;
}
cout << "Array a : " << PrintAr(a);
cout << "Array b : " << PrintAr(b);
cout << "Array c : " << PrintAr(c);
cout << "Array d : " << PrintAr(d);
```

Результат работы программы:

```
Array a : 1  1  0  0  1
Array b : 2  2  2  2  2
Array c : 3  3  3  3  3
Array d : 2  2  3  3  2
```

Пример 16.7 (**Арифметические операции над массивами**). Здесь `PrintAr` — функция для печати массива; см. пример 16.6, строки 6–9.

```
int m = 5;
real[int] a = [1,2,3,4,5], b(m), c = [10,20,30,40,50];
cout << "Array a : " << PrintAr(a);
b = -a;              cout << "Array : " << PrintAr(b);
// b = a-2*a;        cout << "Array : " << PrintAr(b);
// b = -4*a + 2*c;   cout << "Array : " << PrintAr(b);
// c += a;           cout << "Array : " << PrintAr(c);
// c += 2*a;         cout << "Array : " << PrintAr(c);
// c *= 2;           cout << "Array : " << PrintAr(c);
// c *= a;           cout << "Array : " << PrintAr(c);
// c /= 2;           cout << "Array : " << PrintAr(c);
// c /= a;           cout << "Array : " << PrintAr(c);
// c = c .* a;       cout << "Array : " << PrintAr(c);
// c = c ./ a;       cout << "Array : " << PrintAr(c);
```

Пример	Результат работы	Равносильное выражение
`b = -a;`	`-1 -2 -3 -4 -5`	
`b = a - 2 * a;`	`-1 -2 -3 -4 -5`	
`b = -4 * a + 2 * c;`	`16 32 48 64 80`	
`c += a;`	`11 22 33 44 55`	`c = c + a;`
`c += 2 * a;`	`12 23 36 48 60`	`c = c + 2 * a;`
`c *= 2;`	`20 40 60 80 100`	`c = 2 * c;`
`c *= a;`	`10 40 90 160 250`	`c = c .* a;`
`c /= 2;`	`5 10 15 20 25`	
`c /= a;`	`10 10 10 10 10`	`c = c ./ a;`
`c = c .* a;`	`10 40 90 160 250`	`c *= a;`
`c = c ./ a;`	`10 10 10 10 10`	`c /= a;`

Пример 16.8 (**Дополнительные операции над векторами**). Минимальное и максимальное значения (`A.min`, `A.max`); сумма элементов массива (`A.sum`); норма вектора (`A.l1`, `A.l2`, `A.linfty`); скалярное умножение (A^T, A) (`A'*A`) (см. также с. 176).

```
real[int] a = [2, 1, -3, 5, 4];
cout << " ||a||_1 = " << a.l1 << endl;          //  результат: 15
cout << " ||a||_2 = " << a.l2 << endl;          //  результат:  7.4162
cout << " ||a||_infty = " << a.linfty << endl;  //  результат:  5
cout << " sum a_i = " << a.sum << endl;         //  результат:  9
cout << " max a_i = " << a.max << endl;         //  результат:  5
cout << " min a_i = " << a.min << endl;         //  результат: -3
cout << " a'*a = " << (a'*a) << endl;           //  результат: 55
```

16.9.2 Двумерные массивы с целочисленными индексами

Во FreeFem++ можно использовать массивы с двумя индексами — аналог матрицы.

Пример 16.9. Приведем различные способы инициализации массивов.

```
int N = 4, M =5;
real[int,int] A(N,M);
```

```
A = 1;              // все элементы матрицы A = 1
A[0,0] = 0;          // первый элемент массива A = 0
A(2,:) = 3;          // все элементы второй строки матрицы = 3
A(:,1) = 2;          // все элементы I столбца матрицы = 2
A(0:N-1, M-1) = 7;  // все элементы последнего столбца матрицы = 7
real[int,int] B = [ [0.5,2], [3,7], [0,5] ]; // иниц. при описании
int[int,int] E(3,3);
E = [ [1,0,0], [0,1,0], [0,0,1] ];
```

Вывести на экран матрицу можно, например, с помощью оператора

```
cout << " A = " << A << endl;
```

Инициализация матрицы с помощью циклов

```
int N = 5;
real [int,int] A(N,N);
A = 0;
for (int i=0; i<N; i++)
  {
    A(i,i) = 1+i;
    if (i+1<N)
      A(i,i+1) = -i;
  }
```

Инициализация матрицы с помощью двух векторов

```
int N = 4, M = 5;
real[int,int] A(N,M);
real[int] b(N), c(M);
b = [1, 3, 5, 7];
c = [0, 2, 4, 6, 8];
A = b * c';                  // A[i,j] = b[i] * c[j]
A = 2 * b * c';
A = 1;    A += 3 * b * c';
A = 1;    A -= b * c';
```

Оператор	`A = b * c';`	`A = 2 * b * c';`
Результат	$\begin{pmatrix} 0 & 2 & 4 & 6 & 8 \\ 0 & 6 & 12 & 18 & 24 \\ 0 & 10 & 20 & 30 & 40 \\ 0 & 14 & 28 & 42 & 56 \end{pmatrix}$	$\begin{pmatrix} & 4 & 8 & 12 & 16 \\ 0 & 12 & 24 & 36 & 48 \\ 0 & 20 & 40 & 60 & 80 \\ 0 & 28 & 56 & 84 & 112 \end{pmatrix}$

Оператор	`A = 1;  A += 3 * b * c';`	`A = 1;  A -= b * c';`
Результат	$\begin{pmatrix} 1 & 7 & 13 & 19 & 25 \\ 1 & 19 & 37 & 55 & 73 \\ 1 & 31 & 61 & 91 & 121 \\ 1 & 43 & 85 & 127 & 169 \end{pmatrix}$	$\begin{pmatrix} 1 & -1 & -3 & -5 & -7 \\ 1 & -5 & -11 & -17 & -23 \\ 1 & -9 & -19 & -29 & -39 \\ 1 & -13 & -27 & -41 & -55 \end{pmatrix}$

16.10 Разреженные матрицы

В языке FreeFem++ вводится понятие разреженных матриц, для описания которых используется служебное слово `matrix`. Например,

```
matrix A;
```

Собственно говоря, именно разреженные матрицы и конечноэлементные функции (FE-функции, см. 16.5.2 и гл. 18) являются основными конструкциями языка FreeFem++, в то время как обычные матрицы и векторы сохранились для вспомогательных целей (а также по причине того, что ядром языка является C++).

Более подробные сведения об использовании разреженных матриц для решения задач имеются в гл. 19. Здесь описаны лишь операции над такими матрицами (и векторами).

16.10.1 Создание разреженной матрицы

Напомним, что заключительным этапом решения задачи методом конечных элементов является решение системы линейных уравнений

$$\sum_{j=1}^{n} A_{ij}u_j = b_i, \quad i = 1, \ldots, n, \quad Au = b. \tag{16.1}$$

При этом матрица A_{ij} определяется на основе некоторой билинейной формы (см. гл. 1, 2 и формулы (1.7), (1.8), (2.18), (2.19)), например,

$$A(u, v) = \iint\limits_D \nabla u \cdot \nabla v \, dx \, dy, \quad A_{ij} = A(\varphi_i, \varphi_j), \quad i, j = 1, \ldots, n, \tag{16.2}$$

где φ_k — базисные функции.

Векторы u, b в (16.1) это фактически FE-функции, точнее координаты в векторном представлении таких функций

$$u(x, y) = \sum_{k=1}^{n} u_k \varphi_k(x, y), \quad b(x, y) = \sum_{k=1}^{n} b_k \varphi_k(x, y). \tag{16.3}$$

Очевидно, что размерность матрицы A_{ij} определяется размерностью пространства конечных элементов (т. е. количеством базисных функций) и в силу выбора базисных функций φ_k матрица является сильно разреженной. В частности, она может быть трехдиагональной, блочно-диагональной или ленточной матрицей (см. гл. 1, 2).

Для экономии памяти элементы сильно разреженной матрицы целесообразно представлять в виде троек

$$(i, j, A_{ij} \neq 0),$$

иными словами, указывать индексы и значения ненулевых элементов, считая все остальные элементы матрицы равными нулю по умолчанию.

Для этой цели в FreeFem++ задаются три массива `I`, `J`, `C` (имена могут быть произвольными), размерность которых равна количеству ненулевых элементов матрицы. Матрице A ставится в соответствие тройка

```
[I,J,C]=A.
```
(16.4)

Заметим, что некоторые значения массива `C` все-таки могут равняться нулю. Это связано с тем, что в некоторых случаях целесообразно сохранять структуру сильно разреженной матрицы. Например, если матрица A является ленточной (трехдиагональной, пятидиагональной и т. д.), то можно представлять матрицы в виде лент. При этом некоторые элементы на диагоналях могут равняться нулю.

Формула (16.2) показывает основной способ конструирования разреженной матрицы при помощи языка FreeFem++ — необходимо определить какую-либо билинейную форму $A(u, v)$ и вычислить $A(\varphi_i, \varphi_j)$.

Пример 16.10 (**Разреженная матрица на основе билинейной формы**).

```
1  mesh Th = square(1,1);                    // сетка
2  fespace Vh(Th,P1);                        // пространство конечных элем.
3  varf mat(u,v) =                           // varf --- служебное слово
4    int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v));     // билинейная (вариац.) форма
5  matrix A = mat(Vh,Vh);                    // создание разреженной матрицы
6  cout << A << endl;
```

Результатом работы кода будет

```
-- square mesh : nb vertices  =4 ,  nb triangles = 2 ,  nb boundary edges 4
 Nb Of Nodes = 4
 Nb of DF = 4
# Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:
  i j a_ij where (i,j)\in {1,...,n}x{1,...,m}
4 4 0  14
        1         1 1
        1         2 -0.5
        1         3 -0.5
        1         4 0
        2         1 -0.5
        2         2 1
        2         4 -0.5
        3         1 -0.5
        3         3 1
        3         4 -0.5
        4         1 0
        4         2 -0.5
        4         3 -0.5
        4         4 1
```

В привычном представлении эта матрица имеет вид

$$A = \begin{pmatrix} 1{,}0 & -0{,}5 & -0{,}5 & 0{,}0 \\ -0{,}5 & 1{,}0 & 0{,}0 & -0{,}5 \\ -0{,}5 & 0{,}0 & 1{,}0 & -0{,}5 \\ 0{,}0 & -0{,}5 & -0{,}5 & 1{,}0 \end{pmatrix}.$$

Существует и иной способ создания разреженной матрицы. Сначала формируется обычная матрица `A`, которая затем преобразуется в разреженную матрицу `sparseA`: `sparseA=A`. При этом нулевые элементы матрицы `A`, естественно, отсутствуют в представлении разреженной матрицы.

Пример 16.11 (**Разреженная матрица на основе двумерного массива**).

```
int N=4;
real [int,int] A(N,N);  // заполненная матрица
A =0;
for (int i=0; i<N; i++)
  { A(i,i) = 1;
    if (i+1 <  N)  A(i,i+1) = 2;
    if (i-1 >= 0)  A(i,i-1) = -2;
  }
matrix sparseA = A;     // разреженная матрица
cout << sparseA << endl;
```

Результатом работы кода будет

```
# Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:
i j a_ij where (i,j) \in  {1,...,n}x{1,...,m}
4 4 0  10
        1         1 1
        1         2 2
        2         1 -2
        2         2 1
        2         3 2
        3         2 -2
        3         3 1
        3         4 2
        4         3 -2
        4         4 1
```

Здесь текст `# first line: n m (is symmetic) nbcoef` и соответствующая ему строка `4 4 0  10` указывают размерность матрицы `n=4`, `m=4` и количество ненулевых элементов `nbcoef=10`.

В привычном представлении эта матрица имеет вид

$$\texttt{sparseA} = \begin{pmatrix} 1{,}0 & 2{,}0 & 0{,}0 & 0{,}0 \\ 2{,}0 & 1{,}0 & 2{,}0 & 0{,}0 \\ 0{,}0 & 2{,}0 & 1{,}0 & 2{,}0 \\ 0{,}0 & 0{,}0 & 2{,}0 & 1{,}0 \end{pmatrix}.$$

Ввиду важности понятия разреженной матрицы в языке FreeFem++, более подробно опишем способ доступа к ее элементам. Для того, чтобы задать или получить все индексы и элементы разреженной матрицы A, следует описать массивы I, J (типа `int[int]`) и C (`real[int]`). Эти три массива определяют матрицу следующим образом

$$A_{ij} = \sum_k C[k] M_{iI[k],jJ[k]}, \quad M_{iI[k],jJ[k]} = \delta_{iI[k]}\delta_{jJ[k]}, \tag{16.5}$$

где M — базисная матрица, δ_{ik} — символ Кронекера.

Иными словами, для каждого k значение элемента A_{ij} будет $C[k]$. При этом индекс $i = I[k]$, а индекс $j = J[k]$. Приведенную формулу следует рассматривать как некоторую формальную запись матрицы A_{ij}. При численной реализации суммирование, конечно же, не требуется и четырехиндексная матрица M не создается — достаточно просто реализовать перебор всех значений индексов.

Выражение `[I,J,C] = A` позволит получить все элементы матрицы A (массивы автоматически изменяют размеры, подстраиваясь под размер матрицы); `A = [I,J,C]` — задать все элементы матрицы. Заметим, что размер матрицы можно получить, используя соотношения `n = I.max` и `m = J.max`. Другой способ получения информации о матрице это использование выражений `n = A.n` и `m = A.m`. Количество ненулевых элементов разреженной матрицы определяется при помощи величины `C.n`.

Приведем код, демонстрирующий получение информации о разреженной матрице.

Пример 16.12 (**Информация о разреженной матрице**).

```
mesh Th = square(2,1);
fespace Vh(Th,P1);
varf mat(u,v) =
     int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v)); // билинейная форма
matrix A = mat(Vh,Vh);              // создание разреженной матрицы
cout << "A = " << A << endl;
int[int] I(1),J(1); real[int] C(1); // описание массивов для получения
                 // информации о матрице (их размер будет автоматически
                 // изменен при присвоении)
[I,J,C]=A; // получение разреженной матрицы (т.е. только значений <> 0)
cout << " I= " << I << endl;
cout << " J= " << J << endl;
cout << " C= " << C << endl;
```

Результатом выполнения кода будет

```
 -- square mesh : nb vertices  =6 ,  nb triangles = 4 ,  nb boundary edges 6
 Nb Of Nodes = 6
 Nb of DF = 6
A = # Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:   i j a_ij where (i,j)
\in  {1,...,n}x{1,...,m}
6 6 0  24
        1         1 1.25
        1         2 -1
        1         4 -0.25
        1         5 0
        2         1 -1
        2         2 2.5
        2         3 -1
        2         5 -0.5
        2         6 0
```

```
       3         2 -1
       3         3 1.25
       3         6 -0.25
       4         1 -0.25
       4         4 1.25
       4         5 -1
       5         1 0
       5         2 -0.5
       5         4 -1
       5         5 2.5
       5         6 -1
       6         2 0
       6         3 -0.25
       6         5 -1
       6         6 1.25
I= 20
         0       0       0       1       1
         1       1       2       2       2
         3       3       3       4       4
         4       4       5       5       5
J= 20
         0       1       3       0       1
         2       4       1       2       5
         0       3       4       1       3
         4       5       2       4       5
C= 20
       1.25     -1     -0.25    -1     2.5
        -1     -0.5     -1     1.25   -0.25
       -0.25   1.25     -1     -0.5    -1
       2.5      -1     -0.25    -1     1.25
```

Для наглядности приведем привычный вид матрицы A (добавлены нули, которые отсутствуют в разреженной матрице)

$$\begin{pmatrix} 1{,}25 & -1{,}00 & 0{,}00 & -0{,}25 & 0{,}00 & 0{,}00 \\ -1{,}00 & 2{,}50 & -1{,}00 & 0{,}00 & -0{,}50 & 0{,}00 \\ 0{,}00 & -1{,}00 & 1{,}25 & 0{,}00 & 0{,}00 & -0{,}25 \\ -0{,}25 & 0{,}00 & 0{,}00 & 1{,}25 & -1{,}00 & 0{,}00 \\ 0{,}00 & -0{,}50 & 0{,}00 & -1{,}00 & 2{,}50 & -1{,}00 \\ 0{,}00 & 0{,}00 & -0{,}25 & 0{,}00 & -1{,}00 & 1{,}25 \end{pmatrix}.$$

Для разреженной матрицы определена операция выделения диагонали.

Пример 16.13 (**Выделение и задание диагонали разреженной матрицы**).

```
1  mesh Th = square(2,1);
2  fespace Vh(Th, P1);
3  varf mat(u,v) =
4  int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v)); // билинейная форма
5  matrix A = mat(Vh,Vh);              // создание разреженной матрицы
6  cout << "A = " << A << endl;
7  real[int] diagofA(A.n);
8  diagofA = A.diag;       // получение диагонали матрицы
9  A.diag = diagofA ;      // задание диагонали матрицы
10 matrix D = [diagofA];   // задание диагонали разреженной матрицы
```

```
11                              // обычный массив превращается в разреженный
12  cout << " D = " << D << endl;
13  cout << " diag = " << diagofA << endl;
```

Результатом выполнения кода будет

```
-- square mesh : nb vertices  =4 ,  nb triangles = 2 ,  nb boundary edges 4
Nb Of Nodes = 4
Nb of DF = 4
A = # Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:
  i j a_ij where (i,j)\in {1,...,n}x{1,...,m}
4 4 0  14
       1         1 1
       1         2 -0.5
       1         3 -0.5
       1         4 0
       2         1 -0.5
       2         2 1
       2         4 -0.5
       3         1 -0.5
       3         3 1
       3         4 -0.5
       4         1 0
       4         2 -0.5
       4         3 -0.5
       4         4 1
-- Raw Matrix    nxm  =4x4 nb  none zero coef. 4
D = # Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:
  i j a_ij where (i,j)\in  {1,...,n}x{1,...,m}
4 4 1  4
        1         1 1
        2         2 1
        3         3 1
        4         4 1
diag = 4
         1       1       1       1
```

Приведем также пример функции (процедуры), при помощи которой можно генерировать текст для использования в языке LaTeX.

Пример 16.14 (**Запись привычного представления матрицы на языке LaTeX**).

```
1  func int PrintMat(matrix  Mat, string & s)
2  {
3    int[int] I(1),J(1); real[int] C(1);
4    [I,J,C]=Mat;
5    real a;
6    {  ofstream ff(s);
7       int nold=ff.precision(3);  // три знака после запятой
8       ff << "\\" << "[" << endl;
9       ff << "\\" << "left(" << endl;
10      ff << "\\" << "begin{array}{";
```

```
11      for (int j=0; j<Mat.m; j++)
12      ff << "r";
13      ff << "}" << endl;
14      for (int i=0; i<Mat.n; i++)
15        {
16          for (int j=0; j<Mat.m; j++)
17            {
18              a=0;
19              for (int k=0; k<C.n; k++)
20                {
21                  if ((i==I[k])*(j==J[k]))  { a=C[k]; };
22                }
23              if (j!=(Mat.m-1))   { ff.fixed << a << "  &  " ; }
24              else
25                { ff.fixed << a << "\\" << "\\" ; }
26              ff << endl;
27            }
28        }
29      ff << "\\" << "end{array}" << endl;
30      ff << "\\" << "right)" << endl;
31      ff << "\\" << "]" << endl;
32    }
33    return 1;
34  }
```

Вызов функции для разреженной матрицы A осуществляется следующим образом

```
string ss="latex.tex";  PrintMat(A,ss);
```

Результатом является файл `latex.tex`, в котором содержится LaTeX текст.

Наконец, заметим, что одна из причин преобразования обычной матрицы в разреженную это уменьшение памяти для хранения матрицы. Другая (и основная) причина — для ***разреженных матриц*** в языке FreeFem++ реализованы операции матричной алгебры (транспонирование, умножение матриц и др.) В частности, можно использовать операцию `^-1` (см. с. 190), предназначенную для решения систем линейных уравнений $Au = b$, но не для получения обратной матрицы (!):

```
u[]=A^-1*b[]
```

16.10.2 Операции над разреженными матрицами

В языке FreeFem++ для работы с разреженными матрицами имеются операции умножения (`*`), деления (`/`) и деления нацело (`%`). Кроме этого, имеются дополнительные операции транспонирования, поэлементного умножения и т. д. Перечислим некоторые из них.

Символ «`'`» означает (унарную) операцию транспонирования матрицы, т. е. `A'` $= A^T$. В комплексном случае это эрмитова сопряженная матрица, т. е. транспонированная и комплексно сопряженная (`A'` $= A^* = \bar{A}^T$).

Пример 16.15 (**Транспонирование матрицы**).

```
mesh Th = square(1,1);
fespace Vh(Th,P1);
varf mat(u,v) =
   int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v)); // билинейная форма
matrix A = mat(Vh,Vh);                 // создание разреженной матрицы
cout << "A = " << A << endl;
matrix B;
B=A';
cout << "B = " << B << endl;
```

Заметим, что матрица A в данном примере симметрична, т. е. $A^* = A$. Это означает, что результатом работы программы будет матрица B, равная матрице A.

Символ * означает операцию (бинарную) умножения матриц. Операция умножения определена для любых матриц, в том числе и для прямоугольных. Если имеются матрицы $A_{m\times n}$ и $B_{n\times m}$, то возможно получить два различных произведения: матрицу $m \times m$, выполнив AB или матрицу $n \times n$, выполнив BA.

Пример 16.16 (**Умножение матриц**). Формирование двух прямоугольных матриц на основе двумерных массивов

```
include "LATEX.edp"; // строка для подключ. ф-ции PrintMat
int N=2, M=3;
real [int,int] B(N,M), A(M,N); // // заполненная матрица
for (int i=0; i<N; i++)
 {
   for (int j=0; j<M; j++)
    { A(j,i)=i+j;
      B(i,j)=i-j;
    }
 }
```

Создание разреженных матриц `AA`, `BB`

```
 matrix AA=A, BB=B;
```

Описание матриц `CC`, `DD` и выполнение операции умножения

```
matrix CC, DD;
CC=BB*AA; DD=AA*BB;
```

Вывод результата на экране дисплея

```
cout << "AA= " << AA << endl;
cout << "BB= " << BB << endl;
cout << "CC= " << CC << endl;
cout << "DD= " << DD << endl;
```

Создание файлов, содержащих команды языка LaTeX, которые позволяют распечатывать матрицы так, как это принято в математических текстах

```
19  string sss;
20  sss="latexA.txt";  PrintMat(AA, sss);
21  sss="latexB.txt";  PrintMat(BB, sss);
22  sss="latexC.txt";  PrintMat(CC, sss);
23  sss="latexD.txt";  PrintMat(DD, sss);
```

Как уже говорилось, разреженные матрицы являются основной конструкцией, с которой работает язык FreeFem++. Наиболее естественно разреженные матрицы возникают при вычислении значений билинейной формы (см. (16.2) и гл. 19). Конструирование разреженных матриц на основе двумерных массивов следует рассматривать лишь как некоторую дополнительную возможность языка FreeFem++.

Еще одной основной конструкцией языка являются конечноэлементные функции (FE-функции), которые можно рассматривать как функции координат x, y и как вектор конечноэлементного пространства (см. (16.3) и гл. 19). В последнем случае для FE-функции (например, `f`) используется обозначение `f[]`. Иными словами, если имеется FE-функция

$$f(x,y) = \sum_{k=1}^{n} f_k \varphi_k(x,y),$$

то `f[]` — вектор-столбец, который будем называть FE-вектор (см. также замечание на с. 207)

$$\texttt{f[]} = \{f_1, f_2, \ldots, f_n\}.$$

Для разреженных матриц и FE-векторов определена операция умножения. В частности, это означает, что размерности разреженных матриц, построенных на основе билинейных форм, и размерности FE-векторов согласованы между собой — если размерность разреженной матрицы A будет $n \times n$, то размерность FE-вектора также будет n.

Символ `*` означает операцию (бинарную) умножения матрицы на вектор. Естественно, что один и тот же символ используется как для обозначения операции умножения матриц, так и для обозначения операции умножения матрицы на вектор. При этом разреженную матрицу `A` можно умножать на FE-вектор `f` только лишь справа, т. е. `A*f`. Операция умножения слева, т. е. `f*A`, не определена.

Пример 16.17 (**Умножение матриц на вектор**).

```
1  mesh Th = square(1,1);
2  fespace Vh(Th,P1);
3  Vh f;                                  // описание FE-функции
4  f = x*y;                               // задание FE-функции
5  varf mat(u,v) =
6      int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v)); // билинейная форма
7  matrix A = mat(Vh,Vh);                 // создание разреженной матрицы
```

```
Vh m0; m0[] = A*f[];                  // умножение матрицы на вектор
cout << "f =  " << f[]  << endl;
cout << "A =  " << A    << endl;
cout << "m0 = " << m0[] << endl;
```

Результатом работы кода будет

```
-- square mesh : nb vertices  =4 ,  nb triangles = 2 ,  nb boundary edges 4
 Nb Of Nodes = 4
 Nb of DF = 4
f =  4
          0       0       0       1
A =  # Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:
  i j a_ij where (i,j)\in {1,...,n}x{1,...,m}
4 4 0  14
        1         1 1
        1         2 -0.5
        1         3 -0.5
        1         4 0
        2         1 -0.5
        2         2 1
        2         4 -0.5
        3         1 -0.5
        3         3 1
        3         4 -0.5
        4         1 0
        4         2 -0.5
        4         3 -0.5
        4         4 1
m0 = 4
          0     -0.5    -0.5      1
```

В привычной записи имеем

$$\begin{pmatrix} 1{,}0 & -0{,}5 & -0{,}5 & 0{,}0 \\ -0{,}5 & 1{,}0 & 0{,}0 & -0{,}5 \\ -0{,}5 & 0{,}0 & 1{,}0 & -0{,}5 \\ 0{,}0 & -0{,}5 & -0{,}5 & 1{,}0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0{,}0 \\ -0{,}5 \\ -0{,}5 \\ 1{,}0 \end{pmatrix}$$

Для FE-векторов предусмотрены следующие операции:
`.*` оператор поэлементного умножения.
`./` оператор поэлементного деления.
`'*` совокупность транспонирования и матричного произведения.

Пример 16.18 (**Действия с FE-векторами**).

```
mesh Th = square(1,1);
fespace Vh(Th,P1);
Vh f,g;                               // описание FE-функции
f = x*y;                              // задание FE-функции
g = x+y;                              // задание FE-функции
Vh m0;
m0[] = f[].*g[];                      // поэлементное умножение векторов
```

```
8  cout << "f.*g =  " << m0[]  << endl;
9  m0[] = f[]./g[];                    // поэлементное деление векторов
10 cout << "f./g =  " << m0[]  << endl;
11 real a = f[]'*g[];                  // скалярное произведение векторов
12 cout << "a  =  " << a  << endl;
```

Результатом работы кода будет

```
f =  4
          0       0       0       1
g =  4
          0       1       1       2
f.*g =  4
          0       0       0       2
f./g =  4
         -1.#IND   0       0     0.5
a  =  2
```

Здесь `-1.#IND` означает результат деления $0/0$.

Язык FreeFem++ предоставляет возможность решать системы линейных уравнений $Au = f$. Для этих целей предусмотрена операция `^-1` и следующая конструкция

```
u[] = A^-1*f[].
```

Особенно обратим внимание, что хотя здесь и записана обратная матрица `A^-1`, непосредственное вычисление обратной матрицы в языке FreeFem++ не предусмотрено. Выражение `u[] = A^-1*f[]` есть символическая запись для получения решения системы линейных уравнений. На самом деле, более правильно говорить, что имеется операция `^-1*`, т. к. символы `^-1` используются только в комбинации с символом `*`.

✓. *Операцию* `^-1` *нельзя использовать для создания обратной матрицы; следующий оператор вызовет ошибку*

```
matrix AAA = A^-1;
```

✓. *Решение системы линейных уравнений возможно только для разреженных матриц, созданных на основе билинейной формы. Для матриц, созданных на основе двумерных массивов, использование операции* `^-1` *приведет к ошибке.*

Более подробное описание методов решения систем линейных уравнений имеется в гл. 19.

Глава 17

Генерация сеток

В языке FreeFem++ имеется много конструкций, позволяющих управлять построением дискретизации исходной области D. В основе всех алгоритмов лежит триангуляция Делоне–Вороного. Далее описаны лишь наиболее употребительные возможности FreeFem++ (см. также [1]).

17.1 Простейшая область. Ключевое слово `square`

Простейший способ построения сетки предоставляет ключевое слово `square`. Конструкция

```
mesh Th = square(4,5);
```

позволяет сгенерировать сетку 4×5 в области $\overline{D} = [0,1]\times[0,1]$. Добавление параметра `flags` позволяет генерировать сетку, носящую название «Union Jack» (см. рис. 17.1).

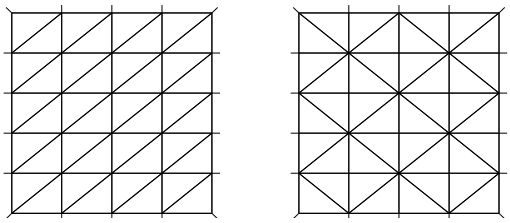

Рис. 17.1. Обычная триангуляция и триангуляция типа «Union Jack» (справа)

Более общая конструкция, позволяющая строить сетку в прямоугольной области $[x_0, x_1] \times [y_0, y_1]$ с количеством точек сетки $(n+1)$ на верхней и нижней границах, и $(m+1)$ на левой и правой границах, имеет вид (n, m — число отрезков на границах)

```
1 real x0=0,  x1=3;
2 real y0=1,  y1=2.1;
3 int  n=20,  m=5;
4 mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
5 // plot(Th, wait=1);  // для визуализации сетки на дисплее
```

Заметим, что в данной конструкции зарезервированные переменные x, y пробегают значения от 0 до 1 каждая.

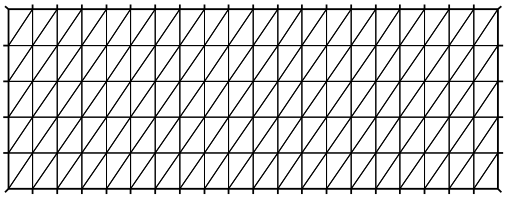

Рис. 17.2. Триангуляция прямоугольной области

17.2 Ключевое слово border

Для задания области D произвольной формы используется ключевое слово border. По существу, область D задается при помощи ее границы Γ, уравнения которой должны быть записаны в параметрическом виде

$$\Gamma_k = \big\{(x,y)\colon x = \varphi_k(t),\ y = \psi_k(t),\ a_k \leqslant t \leqslant b_k\big\}, \quad \Gamma = \bigcup_i \Gamma_i. \qquad (17.1)$$

Легко проверить, что (17.1) задает ориентированную линию

$$t \mapsto (\varphi_k(t), \psi_k(t)), \quad a_k \leqslant t \leqslant b_k.$$

Заметим, что границы Γ_k могут пересекаться только лишь в их конечных точках.

Конструкция, позволяющая задавать сетку в области D, имеет вид (buildmesh — служебное слово для генерации сетки)

$$\texttt{mesh Mesh\_Name = buildmesh}(\Gamma_1(m_1) + \ldots + \Gamma_n(m_n));$$

где m_k положительное или отрицательное число, указывающее какое количество точек $(m_k + 1)$ должно быть на участке границы Γ_k при триангуляции. Положительные значения $m_k > 0$ указывают на то, что при движении вдоль линии Γ_k граница области D остается слева. Напротив, отрицательные значения $m_k < 0$ указывают на то, что граница области D остается справа.

Приведем пример использования border и buildmesh.

```
1 border Gamma1(t=0,2*pi){ x=cos(t); y=sin(t); }
2 border Gamma2(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t); }
3 plot(Gamma1(50) + Gamma2(+30), wait=1) ;  // для визуализации границы
4 mesh ThWithoutHole = buildmesh(Gamma1(50) + Gamma2(+150));
5 mesh ThWithHole = buildmesh(Gamma1(50) + Gamma2(-30));
6 plot(ThWithoutHole, wait=1);
7 plot(ThWithHole, wait=1);
```

Результат работы приведенного кода показан на рис. 17.3.

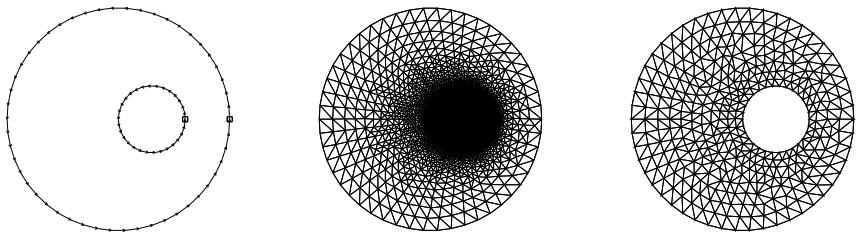

Рис. 17.3. Триангуляция для круга с отверстием (I вариант)

Наиболее общий вид команды `border` может быть следующим

```
border Gamma1(t=0,2*pi){ x=cos(t); y=sin(t); label=1;}
```

Здесь `Gamma1` — идентификатор границы; `(t=0,2*pi)` указывает область изменения параметра t: от 0 до 2π; `label=1` является меткой границы.

Допустимый вариант записи, когда метка границы отсутствует

```
border Gamma1(t=0,2*pi){ x=cos(t); y=sin(t); }
```

Генератор сеток будет генерировать сетку лишь в случае, когда область лежит слева от границы (в противном случае произойдет ошибка выполнения программы). Различие в сетках `ThWithoutHole` и `ThWithHole` заключается в следующем. Сетка `ThWithHole` содержит отверстие в области, т. к. сетка генерируется слева от границ (см. рис. 17.3, на котором показано направление обхода границ). Напротив, сетка `ThWithoutHole` не будет содержать отверстия, т. к. сначала будет сгенерирована внешняя сетка, а затем внутренняя. Иными словами, область D представляет собой бóльший круг, на котором созданы две различных (по размеру треугольников) сетки.

Заметим, что создать сетку без дырки `ThWithoutHole` можно и при помощи следующего кода, изменив направление на кривой Γ_2 (t изменяется от 2π до 0) и записав `Gamma2(+30)` (а не `Gamma2(-30)`) в `buildmesh`.

```
border Gamma1(t=0,2*pi){ x=cos(t); y=sin(t); }
border Gamma2(t=2*pi,0){ x=0.3+0.3*cos(t); y=0.3*sin(t); }
mesh ThWithHole = buildmesh(Gamma1(50)+Gamma2(+30));
```

После изменения ориентации одной из границ оператор `buildmesh` приведет к построению сеток, показанных на рис. 17.4.

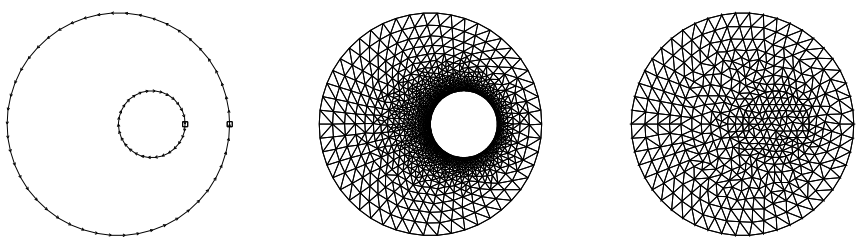

Рис. 17.4. Триангуляция для круга с отверстием (II вариант)

Читателю настоятельно рекомендуется, прежде чем приступать к решению сложных задач, построить различные сетки, проконтролировать расположение границ и проч. Приведем пример, показывающий, что это крайне необходимо.

Две одинаковые на первый взгляд программы генерируют разные сетки. В обоих случаях на границе круга задано десять точек, но в первом случае обход контура осуществляется против часовой стрелки, во втором — по часовой.

```
border Circle0(t=0,2*pi){ x=cos(t); y=sin(t); };
mesh th = buildmesh(Circle0(10));
plot(th, wait=1);
```

```
border Circle0(t=2*pi,0){ x=cos(t); y=sin(t); };
mesh th = buildmesh(Circle0(-10));
plot(th, wait=1);
```

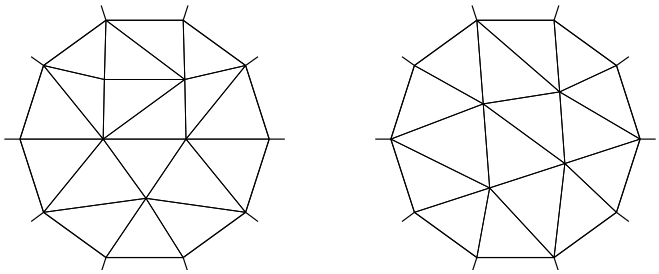

Рис. 17.5. Триангуляция круга

✓. *Заметим, что сетки в FreeFem++ генерируются при помощи программы bamg (bidimensional anisotropic mesh generator), содержащейся в каталоге с FreeFem++, которую можно использовать независимо (о ее возможностях можно узнать, запустив файл* `bamg.exe` *на выполнение с заведомо неверным ключом, например,* `bamg -help`*, см. также [27]).*

17.3 Запись/чтение сгенерированных сеток

Пользователю FreeFem++ предоставляется возможность сохранять данные о сгенерированных сетках в файлы, которые в дальнейшем могут использоваться в какой-либо другой программе. Для этих целей служат ключевые слова `savemesh` и `readmesh`.

```
1 border Circle0(t=0,2*pi){ x=cos(t); y=sin(t); };
2 mesh th = buildmesh(Circle0(100));
3 savemesh(th, "toto.msh");          // сохранение сетки freefem-формата
4 mesh th2 = readmesh("toto.msh");   // чтение сетки
```

Подробно о структуре файлов, содержащих данные о триангуляции области: количестве треугольников, координатах вершин, метках границ и областей, можно прочитать в [1, гл. 5].

17.4 Ключевое слово «triangulate»

FreeFem++ предоставляет возможность построить триангуляцию по заданному множеству точек. Триангуляция строится по алгоритму Делоне-Вороного (см., например, [27]). Информация о точках должна быть задана в виде текстового файла, состоящего из трех колонок, *разделенных пробелами*: первые две колонки — координаты точки x и y, а третья — значения функции $f(x, y)$. Третья колонка не используется при построении триангуляции, но, тем не менее, должна присутствовать в файле.

Пусть имя файла с данными имеет вид `xyf.txt`. Тогда сетка, например, с именем `ThXY` может быть построена при помощи кода (инициализация при описании сетки)

```
mesh ThXY = triangulate("xyf.txt");
```

или

```
mesh ThXY;                          // описание сетки
ThXY = triangulate("xyf.txt"); // построение сетки из файла
```

Третья колонка может быть использована для визуализации изолиний функции $f(x, y)$. Для этого необходимо определить пространство конечных элементов, например, с именем `VhXY`

```
fespace VhXY(ThXY,P1);
```

определить функцию на этом пространстве, например, с именем `fxy`

```
VhXY fxy;
```

прочитать данные из файла в `xx`, `yy`, `fxy[][i]`

```
{ ifstream file("xyf.txt");
  real xx, yy;
  for(int i=0; i<fxy.n; i++)
    file >> xx >> yy >> fxy[][i];
 }
```

и визуализировать изолинии `plot(fxy, wait=1);`

Переменные `xx` и `yy` являются вспомогательными и служат для считывания из файла двух первых колонок данных. Фактически из файла считывается FE-функция в виде элементов массива (см. формулу (18.23)).

Следует обратить внимание на то, что правильный результат будет получен лишь при использовании конечных элементов `P1`. При этом качество построения (и изображения) изолиний может быть недостаточным. Можно попытаться улучшить ситуацию, применяя алгоритм интерполяции.

Для прямоугольной области можно предложить следующий алгоритм:

1. Построить еще одну прямоугольную сетку с бóльшим числом узлов;

2. Задать новое пространство конечных элементов, например, используя элементы `P2`;

3. Задать на этом пространстве новую функцию, например, Fxy;

4. Выполнить операцию присвоения Fxy=fxy (такое присвоение означает процедуру интерполяции, предусмотренную в FreeFem++, см. [1] и пп. 18.1.2–18.1.5).

```
mesh ThXY = triangulate("xyf.txt");
fespace VhXY(ThXY, P1);
VhXY fxy, xx, yy;                          // строки изменены
{ ifstream file("xyf.txt");
  for(int i=0; i<fxy.n; i++)
    file >> xx[][i] >>yy[][i] >> fxy[][i]; // строки изменены
}
plot(fxy, wait=1);
plot(ThXY, wait=1);
real x0=xx[].min,  x1=xx[].max;  // определение размера прямоугольника
real y0=yy[].min,  y1=yy[].max;  // определение размера прямоугольника
int n=15, m=15;
mesh Th = square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]); // новая сетка
plot(Th, wait=1);
fespace Vh(Th, P2);      // задание пространства конечных элементов P2
Vh Fxy;                  // определение функции на новом пространстве
Fxy = fxy;               // интерполяция
plot(Fxy, wait=1);
```

17.5 Ключевое слово «movemesh»

Построенная сетка может быть деформирована достаточно произвольным образом. Для этих целей предусмотрено ключевое слово movemesh

```
Th = movemesh(Th, [F1,F2]);
```

Здесь $(F_1(x,y), F_2(x,y))$ — вектор-функция, задающая деформацию сетки.

Деформированную сетку можно использовать для создания другой сетки, например,

```
mesh Th0 = movemesh(Th, [F1,F2]);
```

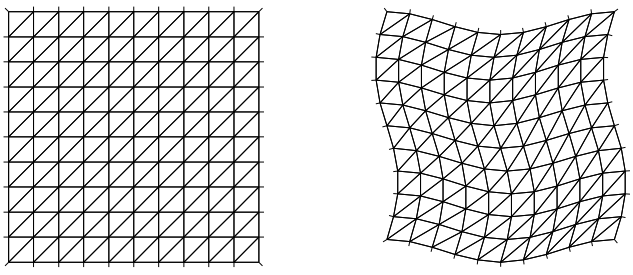

Рис. 17.6. Деформация сетки

```
real x0=0,  x1=2;
real y0=0,  y1=2;
```

```
int  n=10,  m=10;
mesh Th = square(n, m, [x0+(x1-x0)*x, y0+(y1-y0)*y]);
fespace Vh(Th, P2);
Vh F1 = x+sin(pi*y)/10,  F2=y+cos(pi*x)/10;
plot(Th, wait=1);  // для визуализации сетки на дисплее
Th = movemesh(Th, [F1, F2]);
plot(Th, wait=1);  // для визуализации сетки на дисплее
```

✓. *Заметим, что если на исходной сетке была определена некоторая функция f, то после деформации сетки эта функция остается заданной на старой сетке. Для нахождения значений функции f на новой сетке необходимо выполнить операцию* `f=f`.

Иногда деформированная сетка не может быть создана, т. к. в результате деформации некоторые треугольники «выворачиваются» и их площадь становится отрицательной. Для предотвращения этого язык FreeFem++ позволяет выполнить предварительную проверку при помощи ключевого слова `checkmovemesh` и лишь затем производить реальную деформацию. Например,

```
real minarea = checkmovemesh(Th, [F1,F2]);
if (minarea >0)
  { Th = movemesh(Th, [F1,F2]); }
```

Ключевые слова `movemesh`, `checkmovemesh`, в частности, использовались в гл. 6, с. 84 и в гл. 14, с. 149.

17.6 Ключевое слово «adaptmesh»

Функция, заданная на сетке, может достаточно сильно изменяться в некоторой области. Если размеры треугольников сетки велики, то может оказаться, что масштаб изменений функции не соразмерен треугольникам сетки. Например, при изображении линий уровня функции $f(x, y)$ может произойти их резкое сгущение в какой-то области, сравнимой по размерам с треугольником сетки. Естественно, в этом случае сгущение линий невозможно будет увидеть при визуализации и при расчетах. Язык FreeFem++ предлагает функцию `adaptmesh`, позволяющую управлять параметрами сетки при триангуляции. Простейший способ использования

```
Th = adaptmesh(Th, f);
```

где `f` — некоторая сеточная функция.

В результате сетка `Th` автоматически перестроится в соответствии со скоростью изменения функции `f`. Проиллюстрируем сказанное примером.

Пример 17.1. Пусть в квадрате $[-1, 1] \times [-1, 1]$ задана функция

$$f(x, y) = \operatorname{th}\left(-\frac{x^2 + y^2 - r^2}{\varepsilon}\right).$$

Ясно, что при $\varepsilon \to +0$ внутри круга радиуса r $f(x,y) \to -1$, в вне круга $f(x,y) \to 1$. Наибольшие изменения функции будут происходить в окрестности окружности радиуса r. Ширина области, в которой будет происходить «переход» значений функции от 1 до -1, зависит от величины ε. На рис. 17.7, 17.8 хорошо видно, что действие `adaptmesh` привело к резкому увеличению количества треугольников в окрестности окружности радиуса r. Заметим, что изолинии функции после действия `adaptmesh` выглядят как окружности (рис. 17.8 справа), на первоначальной же сетке изолинии (окружности) деформированы.

```
real eps = 0.1;
func f = tanh( -(x^2+y^2-0.8^2)/eps);
mesh Th = square(15,15,[-1+2*x,-1+2*y]);
fespace Vh(Th, P1);
Vh fh = f;
plot(Th, fh, wait=1);
for (int i=0; i<2; i++)
{ Th = adaptmesh(Th, fh);
  fh = f;              // функция на новой сетке
  plot(Th, fh, wait=1);
}
```

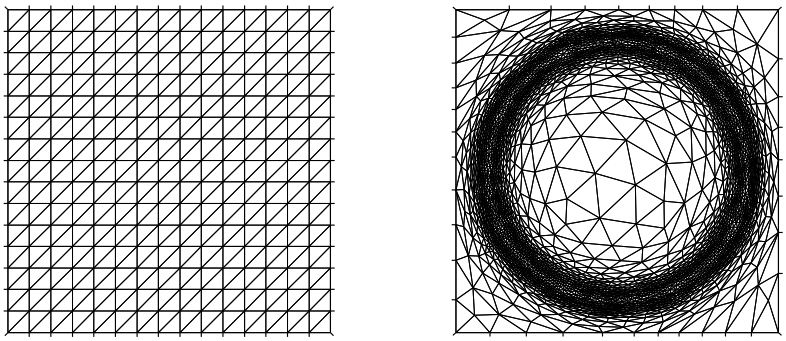

Рис. 17.7. Результат действия `adaptmesh`

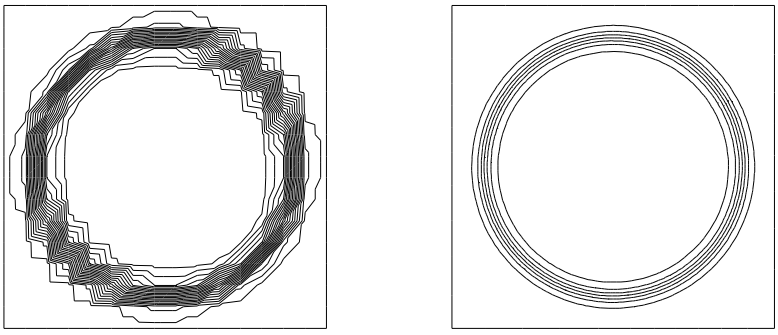

Рис. 17.8. Изолинии до и после действия `adaptmesh`

Функция `adaptmesh` дает широкие возможности управления процессом триангуляции области за счет изменения различных параметров. Приведем список этих параметров, давая лишь краткие пояснения (подробно об использовании параметров `adaptmesh` см. в [1]).

`hmin` = минимальный размер ребра (вещественное значение, которое по умолчанию связано с размером области, подлежащей триангуляции, и точностью генерации сеток);

`hmax` = максимальный размер ребра (вещественное значение, которое по умолчанию связано с размером области, подлежащей триангуляции);

`err` = уровень ошибки интерполяции элементами `P1` (= 0,01)[1];

`errg` = относительная геометрическая ошибка. По умолчанию ошибка равна 0,01 и в других случаях не должна превышать $1/\sqrt{2}$. Сетка, создаваемая с этой опцией, в результате геометрических построений может иметь размер некоторых ребер меньше, чем `hmin`;

`nbvx` = максимальное количество вершин при генерации сетки (= 9000);

`nbsmooth` = количество итераций процедуры сглаживания (= 5);

`nbjacoby` = количество итераций процедуры сглаживания при построении метрики (= 6, 0 — сглаживание отсутствует);

`ratio` = коэффициент для заданного сглаживания на метрике (= 1,8). Если его значение 0 или меньше, чем 1,1, сглаживание на метрике будет выполнено. Если коэффициент больше, чем 1,1, то скорость изменения размеров сетки ограничена величиной $\log($`ratio`$)$. Заметим, что при стремлении `ratio` к единице количество сгенерированных вершин растет. Это можно использовать для контроля за улучшением «плотности» триангуляции области вблизи разрывов или пограничных слоев;

`omega` = параметр релаксации для процедуры сглаживания (= 1,0);

`iso` = если `true`, то метрика насильно делается изотропной (= `false`);

`abserror` = если `false`, то метрика вычисляется с использованием критерия равнораспределения относительной ошибки (= `false`);

`cutoff` = нижний предел относительной ошибки вычислений ($= 10^{-6}$);

`verbosity` = уровень информационных сообщений (может быть выбран от 0 до «бесконечности»);

`inquire` = требовать запроса о действиях с сеткой — на дисплее высвечивается

```
Enter ? for help
```

(= `false`);

`splitpbedge` = если `true`, расщепляет все внутренние ребра на половинки с двумя граничными вершинами (= `true`);

`maxsubdiv` = изменяет метрику так, что максимальное подразделение фоновых ребер ограничено значением `maxsubdiv` (= 10, всегда $\leqslant 10$);

`rescaling` = если `true`, то функция в соответствии с тем, как сетка адаптируется, перемасштабируется между 0 и 1 (= `true`);

`keepbackvertices` = если `true`, то при перестройке сетки делается попытка сохранения как можно большего количества вершин исходной сетки (= `true`);

`IsMetric` = если `true`, то метрика определяется явно (= `false`).

`power` = степень гессиана, используемого при расчете метрики (=1);

`thetamax` = минимальное значение угла раствора (в градусах) для каждого треугольника сетки (= 0);

[1] Здесь и далее в скобках указаны значения по умолчанию.

splitin2 = если true, то расщепляет все треугольники на окончательной сетке на четыре подтреугольника;

metric = массив, состоящий из трех вещественных массивов и позволяющий определять (или получать информацию) о метрике; размер этого массива должен совпадать с количеством вершин треугольников в триангуляции (достаточно полный пример имеется в файле convect-apt.edp, входящем в комплект установки FreeFem++);

nomeshgeneration = если true, то перестроенная сетка не генерируется (используется для вычисления только лишь метрики);

periodic = позволяет строить периодические сетки и задавать периодические краевые условия; формат использования подробно описан в [1], см. также с. 97;

Функция adaptmesh позволяет построить регулярную сетку с заданными начальными размерами треугольников. Для этого можно использовать следующий код

```
real hinit = 0.03; // начальный размер ячейки сетки
Vh hh = hinit;     // определение конечноэлементной функции
                   // для построения сетки данного размера
Th = adaptmesh(Th, hh, IsMetric=1, nbvx=10000);
```

Заметим, что, прежде чем использовать вышеуказанные параметры для проведения расчетов, желательно написать ряд тестовых программ, позволяющих детально понять, к чему приводит изменение того или иного параметра. Наиболее востребованными, скорее всего, могут быть параметры hmin, hmax, nbvx, iso, IsMetric. Задание параметра hmin позволяет избежать ненужного измельчения сетки, а hmax — ее укрупнения (такое укрупнение может произойти, если при использовании adaptmesh(Th, f) в некоторой области функция f сильно изменяется, а в остальной части области почти постоянна).

Параметр nbvx можно использовать с той же целью, что и параметры hmin, hmax, т. е. для избежания укрупнения или измельчения сетки.

Наконец, параметры iso и IsMetric позволяют делать сетку изотропной, что бывает полезным, например, в случае, когда заранее известно, что изучаемый процесс не обладает ярко выраженной анизотропией.

О других возможностях генерации сетки, в частности, о ключевых словах splitmesh и trunc см. в [1].

Глава 18

Конечные элементы

В настоящей главе изложены основные принципы представления в языке FreeFem++ функций, используемых для построения приближенных решений в методе конечных элементов. Подробно описано большое количество базисных функций, имеющихся в настоящее время в словаре языка FreeFem++. Кроме этого, перечислены квадратурные формулы, используемые при вычислениях матриц для метода конечных элементов и указаны способы выбора квадратурных формул при решении конкретных задач. Более подробная информация содержится в [1].

18.1 Интерполяция кусочно-линейными функциями

Приведем формальное определение триангуляции области $D \subset \mathbb{R}^2$.

Определение 18.1.1. *Множество T_h, состоящее из конечного числа треугольников T_k, $k = 1, \ldots, N$, называется* **триангуляцией** *области D, если*

$$T_h = \bigcup_{k=1}^{N} T_k \approx D, \tag{18.1}$$

$$\mathrm{mes}(T_j \cap T_i) = 0, \quad j \neq i. \tag{18.2}$$

Определение 18.1.2. *Точки (x_k, y_k), являющиеся вершинами треугольников, принадлежащих T_h, называются узлами триангуляции. Точки, не принадлежащие границе области D, называются внутренними узлами, а принадлежащие границе — граничными узлами.*

✓. *При триангуляции предполагается, что ни одна из вершин треугольников не лежит на стороне (исключая вершины) другого треугольника.*

Знак $\approx$ в условии (18.1) означает, что треугольники должны ***почти полностью*** покрывать область D. В случае, когда область D является многоугольником (полигоном), всегда можно считать, что $T_h = D$. В случае областей, отличающихся от многоугольника (например, круга), всегда следует уточнять, что понимается под приближенным равенством $T_h \approx D$.

Интуитивно понятно, что проблемы могут возникнуть лишь в окрестности границ области D. Понятно также, что криволинейные участки границы следует заменять прямыми ломанными линиями, превращая тем самым произвольную область в многоугольник. Естественно, при этом считается, что граница области D достаточно гладкая. Точность такой замены области D многоугольником будет зависеть от количества отрезков ломанной, которыми аппроксимируется граница (или от длины звеньев ломанной линии). Заметим, что в FreeFem++ при выполнении процедуры триангуляции следует задавать именно количество отрезков, на которые разбивается граница.

Условие (18.2) означает, что пересечение треугольников из множества T_h либо пусто, либо треугольники могут иметь пересечения в точках, соответствующих их вершинам или сторонам. Иными словами, не должно быть «перекрывающихся» треугольников.

18.1.1 Барицентрические координаты

Предположим, что произведена триангуляция области D, и рассмотрим отдельный треугольник $T \in T_h$. Пусть треугольник задан вершинами $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, $\boldsymbol{q}_3$ (см. рис. 18.1)

$$\boldsymbol{q}_1 = (x_1, y_1), \quad \boldsymbol{q}_2 = (x_2, y_2), \quad \boldsymbol{q}_3 = (x_3, y_3), \tag{18.3}$$

где x_i, y_i — координаты вершин.

Для треугольника будем также использовать обозначение $(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)$, непосредственно указывая его вершины. Так, например, $(\boldsymbol{p}, \boldsymbol{q}_2, \boldsymbol{q}_3)$ означает треугольник с вершинами $\boldsymbol{p}$, $\boldsymbol{q}_2$, $\boldsymbol{q}_3$ (см. рис. 18.1).

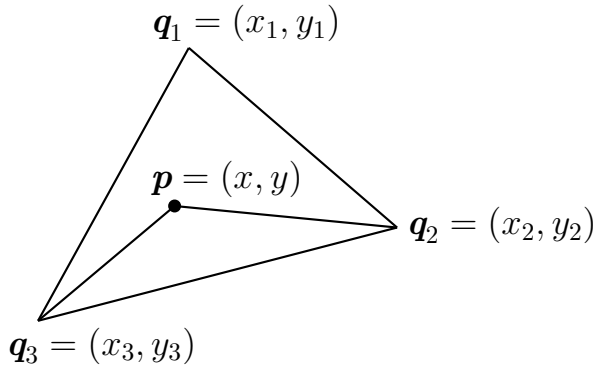

Рис. 18.1. Барицентрические координаты

Точку $\boldsymbol{p} = (x, y)$, лежащую внутри треугольника, можно однозначно определять при помощи так называемых ***барицентрических координат*** λ_1, λ_2, λ_3, задаваемых соотношениями

$$\boldsymbol{p} = \sum_{k=1,2,3} \lambda_k \boldsymbol{q}_k, \qquad \sum_{k=1,2,3} \lambda_k = 1. \tag{18.4}$$

или

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3, \quad y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3, \quad \lambda_1 + \lambda_2 + \lambda_3 = 1. \tag{18.5}$$

Напомним некоторые важные свойства барицентрических координат. Обозначим через $S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)$ площадь соответствующего треугольника. Удвоенная площадь треугольника равна определителю линейной системы уравнений относительно неизвестных λ_1, λ_2, λ_3

$$S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3) = \frac{1}{2} \det \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}. \tag{18.6}$$

Подчеркнем, что имеется ввиду площадь треугольника с точностью до знака (S может быть отрицательной). Очевидно также, что порядок аргументов в выражении $S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)$ важен — он соответствует порядку следования столбцов в определителе. Например, $S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3) = -S(\boldsymbol{q}_2, \boldsymbol{q}_1, \boldsymbol{q}_3)$.

Аналогично, площадь треугольника $(\boldsymbol{p}, \boldsymbol{q}_2, \boldsymbol{q}_3)$ будет

$$S(\boldsymbol{p}, \boldsymbol{q}_2, \boldsymbol{q}_3) = \frac{1}{2} \det \begin{pmatrix} x & x_2 & x_3 \\ y & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}. \tag{18.7}$$

С учетом соотношений (18.6), (18.7) (и аналогичных им) решение системы (18.5) записывается при помощи правила Крамера в виде

$$\lambda_1 = \frac{S(\boldsymbol{p}, \boldsymbol{q}_2, \boldsymbol{q}_3)}{S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)}, \quad \lambda_2 = \frac{S(\boldsymbol{q}_1, \boldsymbol{p}, \boldsymbol{q}_3)}{S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)}, \quad \lambda_3 = \frac{S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{p})}{S(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)}. \tag{18.8}$$

В частности, это означает, что λ_1, λ_2, λ_3 являются ***относительными площадями треугольников***, на которые можно разбить исходный треугольник, выбирая внутреннюю точку $\boldsymbol{p}$ в качестве вершины (см. рис. 18.2). Заметим также, что $\lambda_k = \lambda_k(x, y)$ являются ***линейными*** функциями.

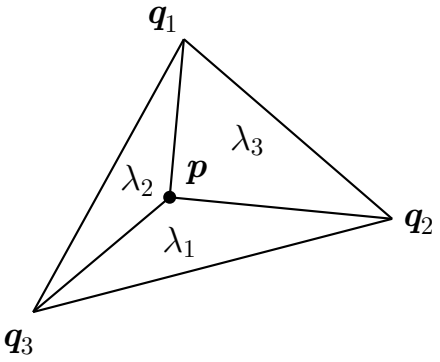

Рис. 18.2. Барицентрические координаты

Следующее свойство барицентрических координат заключается в том, что с их помощью вершины и стороны треугольника представляются

в очень простой форме. Так, например, вершине $\boldsymbol{q}_1$ соответствуют $\lambda_1 = 1$, $\lambda_2 = 0$, $\lambda_3 = 0$ и т. д.

$$\begin{aligned}
\boldsymbol{q}_1 = (x_1, y_1) &\iff \lambda_1 = 1, \quad \lambda_2 = 0, \quad \lambda_3 = 0, \quad \boldsymbol{\lambda} = (1, 0, 0), \qquad (18.9)\\
\boldsymbol{q}_2 = (x_2, y_2) &\iff \lambda_1 = 0, \quad \lambda_2 = 1, \quad \lambda_3 = 0, \quad \boldsymbol{\lambda} = (0, 1, 0),\\
\boldsymbol{q}_3 = (x_3, y_3) &\iff \lambda_1 = 0, \quad \lambda_2 = 0, \quad \lambda_3 = 1, \quad \boldsymbol{\lambda} = (0, 0, 1),\\
[\boldsymbol{q}_2, \boldsymbol{q}_3] &\iff \lambda_1 = 0, \quad \lambda_2 + \lambda_3 = 1, \qquad (18.10)\\
[\boldsymbol{q}_1, \boldsymbol{q}_3] &\iff \lambda_2 = 0, \quad \lambda_1 + \lambda_3 = 1,\\
[\boldsymbol{q}_1, \boldsymbol{q}_2] &\iff \lambda_3 = 0, \quad \lambda_1 + \lambda_2 = 1.
\end{aligned}$$

Здесь $[\boldsymbol{q}_i, \boldsymbol{q}_k]$ — сторона треугольника, соединяющая вершины $\boldsymbol{q}_i$, $\boldsymbol{q}_k$.

Заметим, что для получения соотношений (18.9), (18.10) можно воспользоваться рис. 18.2, последовательно помещая точку $\boldsymbol{p}$ в вершины и на стороны треугольника и вспоминая, что λ_k это относительные площади треугольников.

В частности, с учетом (18.10), например, для точки $\boldsymbol{p}$ лежащей на стороне $[\boldsymbol{q}_i, \boldsymbol{q}_k]$, легко получить параметрическое представление точек отрезка, соединяющего точки $\boldsymbol{q}_i$, $\boldsymbol{q}_k$

$$\begin{gathered}
\boldsymbol{p} \in [\boldsymbol{q}_i, \boldsymbol{q}_k] \implies \boldsymbol{p} = t\boldsymbol{q}_i + (1-t)\boldsymbol{q}_k, \qquad (18.11)\\
(\lambda_i = t, \quad \lambda_k = 1 - t), \quad 0 \leqslant t \leqslant 1
\end{gathered}$$

или

$$\boldsymbol{p} \in [\boldsymbol{q}_i, \boldsymbol{q}_k] \implies x = tx_i + (1-t)x_k, \quad y = ty_i + (1-t)y_k, \quad 0 \leqslant t \leqslant 1. \qquad (18.12)$$

18.1.2 Интерполяция на треугольнике

Предположим, что имеется функция $u(x, y)$, значения которой в вершинах треугольника $(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)$ заданы, т. е. известны значения $u_k = u(x_k, y_k)$. Линейная интерполяция $u_h(x, y)$ во внутренних точках $\boldsymbol{p} = (x, y)$ треугольника может быть записана при помощи барицентрических координат

$$u_h(\boldsymbol{p}) = \sum_{k=1,2,3} \lambda_k u(\boldsymbol{q}_k), \quad \left(u_h(x, y) = \sum_{k=1,2,3} \lambda_k(x, y) u_k, \quad u_k = u(x_k, y_k) \right). \qquad (18.13)$$

Действительно, $u_h(x, y)$ в силу (18.8) является линейной функцией от x, y. Далее, если $\boldsymbol{p} = \boldsymbol{q}_1$, то с учетом (18.9) имеем $\lambda_1 = 1$, $\lambda_2 = 0$, $\lambda_3 = 0$ и, следовательно, $u_h(\boldsymbol{p})$ совпадает с $u_h(\boldsymbol{q}_1)$, и т. д.

На сторонах треугольника функция $u_h(x, y)$ с учетом (18.10), (18.11), (18.12) представима в виде

$$\begin{gathered}
\boldsymbol{p} \in [\boldsymbol{q}_i, \boldsymbol{q}_k] \Rightarrow u_h(\boldsymbol{p}) = tu(\boldsymbol{q}_i) + (1-t)u(\boldsymbol{q}_k), \qquad (18.14)\\
\boldsymbol{p} = t\boldsymbol{q}_i + (1-t)\boldsymbol{q}_k, \quad 0 \leqslant t \leqslant 1
\end{gathered}$$

или

$$u_h(x,y) = tu(x_i,y_i) + (1-t)u(x_k,y_k),$$

$$x = tx_i + (1-t)x_k, \quad y = ty_i + (1-t)y_k.$$

Обратим внимание, что при линейной интерполяции (18.13) барицентрические координаты $\lambda_k = \lambda_k(x,y)$ играют роль базисных функций.

18.1.3 Линии уровня

С помощью (18.14) можно строить линии уровня (изолинии) функции $u(x,y)$. Конечно, имеется ввиду интерполяция линий уровня.

Линия уровня функции $u(\boldsymbol{p})$ для заданного значения μ это множество точек вида

$$L_\mu = \{\boldsymbol{p}\colon u(\boldsymbol{p}) = \mu\}\,. \tag{18.15}$$

Пусть на треугольнике $(\boldsymbol{q}_1,\boldsymbol{q}_2,\boldsymbol{q}_3)$ задана линейная интерполяция функции u_h (см. (18.13)). Пересечение точки $\boldsymbol{p} \in L_\mu$ со стороной треугольника $[\boldsymbol{q}_i,\boldsymbol{q}_k]$ (если, конечно, такое пересечение имеется для заданного μ) определяется с помощью (18.14)

$$u_h(\boldsymbol{p}) = tu(\boldsymbol{q}_i) + (1-t)u(\boldsymbol{q}_k) = \mu, \tag{18.16}$$

$$\boldsymbol{p} = t\boldsymbol{q}_i + (1-t)\boldsymbol{q}_k, \quad 0 \leqslant t \leqslant 1.$$

Исключая t $(0 \leqslant t \leqslant 1)$, получим

$$t = \frac{\mu - u(\boldsymbol{q}_k)}{u(\boldsymbol{q}_i) - u(\boldsymbol{q}_k)}, \quad \boldsymbol{p} = \frac{\mu - u(\boldsymbol{q}_k)}{u(\boldsymbol{q}_i) - u(\boldsymbol{q}_k)}\boldsymbol{q}_i + \frac{u(\boldsymbol{q}_i) - \mu}{u(\boldsymbol{q}_i) - u(\boldsymbol{q}_k)}\boldsymbol{q}_k, \tag{18.17}$$

или

$$t = \frac{\mu - u_k}{u_i - u_k}, \quad x = \frac{\mu - u_k}{u_i - u_k}x_i + \frac{u_i - \mu}{u_i - u_k}x_k, \quad y = \frac{\mu - u_k}{u_i - u_k}y_i + \frac{u_i - \mu}{u_i - u_k}y_k.$$

Заметим, что при численной реализации алгоритма необходимо осуществлять проверку условия $(u_i - u_k) \neq 0$ и условия $t \in [0,1]$ — это как раз и будет проверкой возможности пересечения данной линии уровня со стороной треугольника. В случае, когда $u_i = u_k = \mu$, линия уровня совпадает со стороной треугольника.

Подчеркнем, что если (в случае использования линейной интерполяции) данная линия уровня проходит через выбранный треугольник, то она представляет собой отрезок, концы которого либо лежат на двух сторонах треугольника, либо на стороне и вершине треугольника (правда, возможен еще вариант, когда $u(\boldsymbol{q}_1) = u(\boldsymbol{q}_2) = u(\boldsymbol{q}_3) = \mu$). В частности, это означает, что линия уровня попадет и в соседний треугольник (см. рис. 18.3).

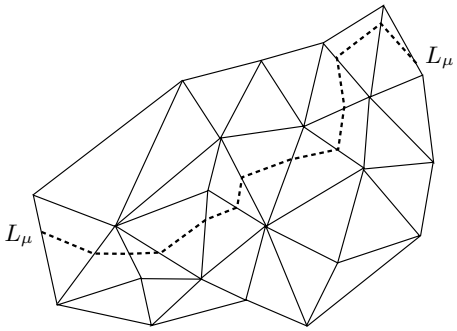

Рис. 18.3. Схема построения линии уровня (пунктир)

18.1.4 FE-функции

При использовании метода конечных элементов функция $u(x, y)$ аппроксимируется при помощи формулы

$$u(x, y) \approx u^h(x, y) = \sum_{k=1}^{n} u_k \varphi_k(x, y), \tag{18.18}$$

где $\varphi_k(x, y)$ — базисные функции, u_k — коэффициенты (если $u(x, y)$ комплексная функция, то u_k также будут комплексными).

По существу, это означает, что задается некоторое ***конечномерное*** пространство V_h, которое называется пространством конечно-элементных функций (далее часто используется сокращение — FE-***функция***)

$$V_h = \{w : \ w_1\varphi_1 + w_2\varphi_2 + \ldots + w_n\varphi_n, \quad w_k \in \mathbb{R}^n\}, \tag{18.19}$$

где $\varphi_k(x, y)$ — базисные функции.

Величину u^h можно считать как функцией, т. е. $u^h = u^h(x, y)$, так и вектором из $\mathbb{R}^n$

$$u^h = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n. \tag{18.20}$$

При использовании формулы (18.18) полагается, что базисные функции удовлетворяют условиям

$$\varphi_k(x_k, y_k) = 1, \quad \varphi_k(x_i, y_i) = 0, \quad i \neq k, \tag{18.21}$$

где (x_k, y_k) — узлы триангуляции T_h.

В этом случае коэффициенты u_k будут совпадать со значениями интерполируемой функции $u(x, y)$ в узлах сетки

$$u_k = u(x_k, y_k). \tag{18.22}$$

Формулу (18.18) при выполнении условий (18.21), (18.22) называется ***интерполяционной формулой Лагранжа***.

✓. *FE-функция $u(x, y)$ связана с массивом чисел u_k и к ней можно обращаться и как к функции двух переменных, и как к массиву чисел.*

*В представлении языка FreeFem++ формула (*18.18*) записывается в виде*

$$u(x,y) \to \sum_{k=0}^{n-1} \texttt{u[][k]}\,\varphi_k(x,y), \tag{18.23}$$

где $n =$ `Vh.ndof`*. Несоответствие в индексах формул (*18.18*) и (*18.23*) вызвано тем, что в FreeFem++ элементы массивов нумеруются с нуля.*

Приведем простейший (и бессодержательный) пример трех способов использования FE-функции. Подробнее см. гл. 19*.*

```
1  Vh u, v;
2  u = x * y;
3  // I способ
4  v = u;
5  // II способ
6  v[] = u[];
7  // III способ
8  for (int m=0; m<=Vh.ndof-1; m++)
9   { v[][m] = u[][m]; }
```

18.1.5 Кусочно-линейная интерполяция

Рассмотрим случай линейной интерполяции. Пусть имеется триангуляция T_h области D и треугольник $T_i \in T_h$ имеет вершины $\boldsymbol{q}_{i_1}$, $\boldsymbol{q}_{i_2}$, $\boldsymbol{q}_{i_3}$ (или в других обозначениях $T_i = (\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})$). Базисную функцию, связанную с вершиной $\boldsymbol{q}_{i_1}$, ***для точек, принадлежащих треугольнику*** T_i, зададим как барицентрическую координату

$$\varphi_{i_1} = \lambda_1^i(x,y), \quad (x,y) \in T_i. \tag{18.24}$$

Индекс i у величины λ_1^i означает, что барицентрические координаты определяются для треугольника T_i.

С учетом (18.8) имеем

$$\lambda_1^i = \frac{S(\boldsymbol{p}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})}{S(\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})}, \quad \boldsymbol{p} = (x,y) \in T_i. \tag{18.25}$$

Может показаться, что базисная функция, заданная формулой (18.24), определяется не единственным образом, т. к. вершина $\boldsymbol{q}_{i_1}$, помимо треугольника T_i, является еще и вершиной для смежных треугольников (по крайней мере, одного в случае, когда вершина принадлежит границе). На самом деле, формула (18.24) определяет φ_{i_1} лишь для треугольника T_i, а для смежных треугольников, естественно, будет иная формула.

Поясним сказанное примером (см. рис. 18.4, на котором показана триангуляция некоторой области D и числами в кружках нумеруются треугольники). Пусть два треугольника $T_i = (\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})$ и $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ имеют общую вершину $\boldsymbol{q}_{i_1} = \boldsymbol{q}_{k_3}$. Тогда для точек треугольника T_i и точек

треугольника T_k (для определенности, скажем, треугольники $i = 1$ и $k = 2$ с общей вершиной 8 на рис. 18.4) имеем

$$\varphi_{i_1} = \lambda_1^i = \frac{S(\boldsymbol{p}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})}{S(\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})}, \quad \boldsymbol{p} = (x, y) \in T_i, \tag{18.26}$$

$$\varphi_{i_1} = \lambda_3^k = \frac{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{p},)}{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})}, \quad \boldsymbol{p} = (x, y) \in T_k.$$

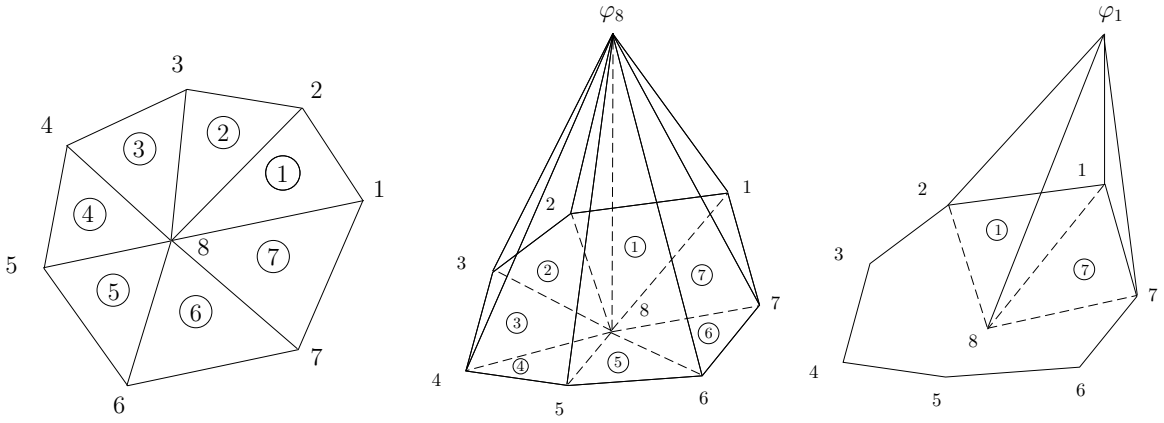

Рис. 18.4. Сетка T_h и базисные функции φ_1, φ_8

После выбора базисных функций в виде (18.24) очевидно, что функция $u^h(x, y)$ является кусочно-линейной, т. к. $\varphi_k(x, y)$ линейны. Из формул (18.11) следует, что значения $u^h(x, y)$ на стороне любого треугольника определяются лишь координатами вершин, принадлежащих данной стороне. Это означает, что $u^h(x, y)$ является непрерывной функцией — при переходе от одного треугольника к другому на соприкасающихся сторонах треугольников значения функции $u^h(x, y)$ одинаковы.

Более того, для каждого треугольника, например, $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$, вместо формулы (18.18) можно использовать формулу (ср. с (18.13))

$$u_h(\boldsymbol{p}) = \varphi_{k_1} u_{k_1} + \varphi_{k_2} u_{k_2} + \varphi_{k_3} u_{k_3} = \lambda_k^1 u_{k_1} + \lambda_k^2 u_{k_2} + \lambda_k^3 u_{k_3}, \tag{18.27}$$

$$u_{k_i} = u(\boldsymbol{q}_{k_i}), \quad \boldsymbol{p} \in T_k.$$

Изложенный подход к построению базисных функций, конечно, является более общим, чем способ, приведенный в гл. 2. В качестве упражнения полезно убедиться, что в частном случае, рассмотренном в гл. 2, базисные функции (2.14) (см. также рис. 2.3) будут совпадать с (18.24).

В заключение рассмотрим пример конкретной сетки, показанной на рис. 18.4. На этом же рисунке изображены базисные функции φ_1 и φ_8. Носителями этих базисных функций будут

$$\operatorname{supp} \varphi_1 = T_1 \cup T_7, \quad \operatorname{supp} \varphi_8 = T_1 \cup T_2 \cup T_3 \cup T_4 \cup T_5 \cup T_6 \cup T_7,$$

$$\operatorname{supp}\varphi_1 \cap \operatorname{supp}\varphi_8 = T_1 \cup T_7.$$

При вычислении интегралов, в частности, имеем

$$\iint\limits_D \varphi_1\varphi_8\,dx\,dy = \iint\limits_{T_1\cup T_7} \varphi_1\varphi_8\,dx\,dy.$$

18.2 Базисные функции в FreeFem++

Как уже говорилось, в FreeFem++ для представления функций применяется формула (18.18)

$$u(x,y) \approx u^h(x,y) = \sum_{k=1}^{N} u_k\varphi_k(x,y), \tag{18.28}$$

где $\varphi_k(x,y)$ — базисные функции, u_k — коэффициенты, N — количество базисных функций (размерность пространства V_h).

Для задания ***конечномерного*** пространства V_h

$$V_h = \left\{w:\ w_1\varphi_1 + w_2\varphi_2 + \ldots + w_N\varphi_N,\quad w_k \in \mathbb{R}^N\right\} \tag{18.29}$$

используется ключевое слово `fespace`

```
fespace IDspace(IDmesh,<IDFE>);
```

где `IDspace` — идентификатор пространста V_h (например, `Vh`), `IDmesh` — идентификатор триангуляции (например, `Th`), `<IDFE>` — идентификатор типа базисных функций (конечных элементов).

✓. *Строго говоря, термин* ***конечный элемент*** *означает* ***носитель базисной функции****. Однако очень часто базисные функции также называют конечными элементами. Во всяком случае, такая терминология принята в FreeFem++ и в дальнейшем, как и во FreeFem++, термин* ***конечный элемент*** *будет часто использоваться для названия базисных функций.*

В настоящее время для указания типов конечных элементов FreeFem++ позволяет использовать в качестве идентификатора `<IDFE>` следующие обозначения: `P0`, `P1`, `P1dc`, `P1b`, `P2`, `P2b`, `P2dc`, `RT0`, `P1nc`.

`P0` — кусочно-постоянные разрывные конечные элементы;
`P1` — кусочно-линейные непрерывные конечные элементы;
`P1dc` — кусочно-линейные разрывные конечные элементы;
`P1b` — кусочно-линейные непрерывные конечные элементы + bubble[1];
`P2` — кусочно-квадратичные непрерывные конечные элементы;
`P2b` — кусочно-квадратичные непрерывные конечные элементы + bubble;
`P2dc` — кусочно-квадратичные разрывные конечные элементы;
`RT0` — конечные элементы Равьяра–Томаса;
`P1nc` — кусочно-линейные конечные элементы, непрерывные лишь в середине стороны треугольника (несогласованные конечные элементы).

[1] В русскоязычной литературе отсутствует устоявшийся перевод термина bubble.

✓. *Далее будем использовать следующие обозначения для параметров триангуляции T_h: n_v — количество вершин треугольников, n_t — количество треугольников, n_m — количество точек, являющихся серединами сторон треугольников.*

✓. *В языке FreeFem++ для параметров триангуляции* `Th` *используются обозначения:*

$$\texttt{Th.nt} = n_t, \quad \texttt{Th.nv} = n_v, \quad n_m = n_t + n_v - 1.$$

✓. *Возможен доступ к координатам вершин треугольника*

$$T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_1}), \boldsymbol{q}_{k_j} = (x_{k_j}, y_{k_j}),$$

$$\texttt{Th[k-1][j-1].x} = x_{k_j}, \quad \texttt{Th[k-1][j-1].y} = y_{k_j}.$$

При этом предполагается, что обход вершин осуществляется против часовой стрелки.

18.2.1 P0 элементы

`P0` элементы являются кусочно-постоянными конечными элементами. Базисные функции φ_k, определяющие пространство V_h на триангуляции T_h (в терминах FreeFem++ — `Vh(Th,P0)`), заданы в форме

$$\varphi_k(x,y) = 1, \ \ (x,y) \in T_k; \quad \varphi_k(x,y) = 0, \ \ (x,y) \notin T_k, \ \ T_k \in T_h. \tag{18.30}$$

Формула (18.28) записывается в виде

$$u^h(x,y) = \sum_{k=1}^{n_v} \left(\frac{u(\boldsymbol{q}_{k_1}) + u(\boldsymbol{q}_{k_2}) + u(\boldsymbol{q}_{k_3})}{3} \right) \varphi_k(x,y), \tag{18.31}$$

где n — количество узлов (вершин треугольников) в триангуляции T_h.

В частности, для отдельного треугольника $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ имеем

$$u^h(x,y) = \frac{u(\boldsymbol{q}_{k_1}) + u(\boldsymbol{q}_{k_2}) + u(\boldsymbol{q}_{k_3})}{3} \varphi_k(x,y), \quad (x,y) \in T_k.$$

18.2.2 P1 элементы

`P1` элементы являются кусочно-линейными непрерывными конечными элементами. Базисные функции φ_{i_1}, определяющие пространство V_h на триангуляции T_h (в терминах FreeFem++ — `Vh(Th,P1)`), были подробно описаны в п. 18.1.5 (см. (18.24)) и имеют вид

$$\varphi_{i_1}(x,y) = \lambda_1^i(x,y) = \frac{S(\boldsymbol{p}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})}{S(\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3})}, \quad \boldsymbol{p} = (x,y) \in T_i. \tag{18.32}$$

Напомним, что такие базисные функции линейны

$$\varphi_{i_1}(x,y) = a_{i_1} x + b_{i_1} y + c_{i_1}, \quad (x,y) \in T_i,$$

равны нулю во всех вершинах треугольников за исключением вершины $\boldsymbol{q}_{i_1}$

$$\varphi_{i_1}(\boldsymbol{q}_{i_1}) = 1, \quad \varphi_{i_1}(\boldsymbol{q}_j) = 0, \quad j \neq i_1$$

и для каждого треугольника $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ имеем (см. (18.27))

$$u_h(\boldsymbol{p}) = \varphi_{k_1} u_{k_1} + \varphi_{k_2} uk_2 + \varphi_{k_3} u_{k_3} = \lambda_k^1 u_{k_1} + \lambda_k^2 u_{k_2} + \lambda_k^3 u_{k_3},$$
$$u_{k_i} = u(\boldsymbol{q}_{k_i}), \quad \boldsymbol{p} \in T_k.$$

Формула (18.28) записывается в виде

$$u^h(x, y) = \sum_{k=1}^{n_v} u(\boldsymbol{q}_k)\varphi_k(x, y),$$

где n_v — количество узлов (вершин треугольников) в триангуляции T_h.

18.2.3 P2 элементы

`P2` элементы являются кусочно-квадратичными непрерывными конечными элементами. Базисные функции φ_i, определяющие пространство V_h на триангуляции T_h (в терминах языка FreeFem++ — `Vh(Th,P2)`), задаются в виде квадратичного полинома

$$\varphi_i(x, y) = a_i x + b_i y + c_i + d_i x^2 + e_i xy + f_i y^2, \quad (x, y) \in T_i,$$
$$\varphi_i(\boldsymbol{q}_i) = 1, \quad \varphi_i(\boldsymbol{q}_j) = 0, \quad j \neq i.$$

Функции φ_i определяются шестью параметрами. В этом случае говорят, что φ_i имеют шесть степеней свободы. Заметим, что для `P0` элементов степень свободы равна 1, а в случае `P1` элементов — равна 3.

Для `P2` элементов базисные функции задаются не только в вершинах треугольников, но и дополнительно в точках, являющихся серединами сторон треугольников. Рассмотрим треугольник $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ (см. рис. 18.5) и обозначим $\boldsymbol{m}_{k_{12}}$, $\boldsymbol{m}_{k_{13}}$, $\boldsymbol{m}_{k_{23}}$ середины его сторон

$$\boldsymbol{m}_{k_{12}} = \frac{1}{2}(\boldsymbol{q}_{k_1} + \boldsymbol{q}_{k_2}), \quad \boldsymbol{m}_{k_{13}} = \frac{1}{2}(\boldsymbol{q}_{k_1} + \boldsymbol{q}_{k_3}), \quad \boldsymbol{m}_{k_{23}} = \frac{1}{2}(\boldsymbol{q}_{k_2} + \boldsymbol{q}_{k_3}). \quad (18.33)$$

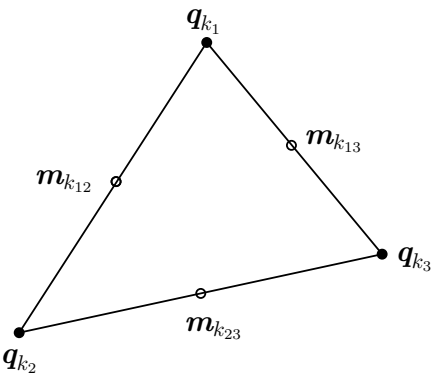

Рис. 18.5. Вершины и середины сторон треугольника T_k

Для вершины треугольника $\boldsymbol{q}_{k_1}$ базисная функция задается в виде

$$\varphi_{k_1}(x,y) = \lambda_1^k(x,y)(2\lambda_1^k(x,y) - 1), \tag{18.34}$$

$$\lambda_1^k(x,y) = \frac{S(\boldsymbol{p}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})}{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})}, \quad \boldsymbol{p} = (x,y) \in T_k.$$

Для точки $\boldsymbol{m}_{k_{12}}$, являющейся серединой стороны, базисная функция задается в виде

$$\varphi_{k_{12}}(x,y) = 4\lambda_1^k(x,y)\lambda_2^k(x,y), \tag{18.35}$$

$$\lambda_2^k(x,y) = \frac{S(\boldsymbol{q}_{k_1}, \boldsymbol{p}, \boldsymbol{q}_{k_3})}{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})}, \quad \boldsymbol{p} = (x,y) \in T_k.$$

Формально, соотношение (18.28) для определения $u^h(x,y)$ остается прежним, однако лучше использовать следующую форму записи

$$u(x,y) \approx u^h(x,y) = \sum_{k=1}^{n_v} u(\boldsymbol{q}_k)\varphi_k^v(x,y) + \sum_{k=1}^{n_m} u(\boldsymbol{m}_k)\varphi_k^m(x,y), \tag{18.36}$$

где $\varphi_k^v(x,y)$, $\varphi_k^m(x,y)$ — соответственно, базисные функции для вершин и для середин сторон.

18.2.4 P1nc элементы

`P1nc` элементы являются кусочно-линейными конечными элементами, которые непрерывны лишь в середине стороны треугольника (другое название — несогласованные конечные элементы). Базисные функции φ_i, определяющие пространство V_h на триангуляции T_h (`Vh(Th,P1nc)` в терминах FreeFem++), задаются в виде линейной функции

$$\varphi_i(x,y) = a_i x + b_i y + c_i, \quad (x,y) \in T_i,$$

$$\varphi_i(\boldsymbol{m}_i) = 1, \quad \varphi_i(\boldsymbol{m}_j) = 0, \quad j \neq i,$$

где $\boldsymbol{m}_i$ — середины сторон треугольников (см. (18.33)).

Заметим, что φ_i в случае `P1nc` элементов не являются непрерывными и при дифференцировании возникают δ-функции Дирака. Однако при вычислении интегралов наличие δ-функций игнорируется.

Соотношение (18.28) для определения $u^h(x,y)$ записывается в форме

$$u(x,y) \approx u^h(x,y) = \sum_{k=1}^{n_m} u(\boldsymbol{m}_k)\varphi_k(x,y). \tag{18.37}$$

Способ построения `P1nc` элементов детально изложен в [10]. Здесь, как и в [1], перечислим лишь наиболее важные свойства `P1nc`.

1. Пусть $A(u,v)$ билинейная форма

$$A(u,v) = \int_D \nabla u \cdot \nabla v \, dxdy.$$

Справедливы соотношения

$$A(\varphi_i, \varphi_i) > 0, \quad A(\varphi_i, \varphi_j) \leqslant 0, \quad i \neq j, \quad \sum_{k=1}^{n_m} A(\varphi_i, \varphi_k) \geqslant 0.$$

2. Пусть имеется уравнение $\Delta u = -f$. Тогда $f \geqslant 0 \Rightarrow u^h \geqslant 0$.

3. L^2-ортогональность

$$\int\limits_D \varphi_i, \varphi_j \, dxdy = 0, \quad i \neq j.$$

Заметим, что для обычных P1 элементов условие L^2-ортогональности не выполняется.

18.2.5 Элементы P1b и P2b

Для каждого треугольника $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3}) \in T_h$ определим функцию, которая в англоязычной литературе по методу конечных элементов называется ***bubble function***, что связано с характерной «пузырчатой» формой этой функции

$$\beta_k(x, y) = 27\lambda_1^k(x, y)\lambda_2^k(x, y)\lambda_3^k(x, y), \quad (x, y) \in T_k. \tag{18.38}$$

Функция $\beta_k(x, y)$ обладает следующими свойствами:

1. Обращается в нуль на границе треугольника ∂T_k (на сторонах треугольника)

$$\beta_k(x, y) = 0, \quad (x, y) \in \partial T_k. \tag{18.39}$$

2. Равна 1 в барицентре $\boldsymbol{q}_{k_b}$ треугольника T_k

$$\beta_k(\boldsymbol{q}_{k_b}) = 1, \quad \boldsymbol{q}_{k_b} = \frac{1}{3}\left(\boldsymbol{q}_{k_1} + \boldsymbol{q}_{k_2} + \boldsymbol{q}_{k_3}\right). \tag{18.40}$$

`P1b` элементы строятся на основе `P1` элементов с добавлением bubble function. Соотношение (18.28) для $u^h(x, y)$ записывается в форме

$$u(x, y) \approx u^h(x, y) = \sum_{k=1}^{n_v} u(\boldsymbol{q}_k)\varphi_k(x, y) + \sum_{k=1}^{n_t} u(\boldsymbol{q}_{k_b})\beta_k(x, y), \tag{18.41}$$

где $\varphi_k(x, y)$ — базисные функции, такие же как для `P1` (см. (18.32)).

В терминах FreeFem++ пространство V_h задается кодом — `Vh(Th,P1b)`.

`P2b` элементы строятся на основе `P2` элементов с добавлением bubble function. Соотношение (18.28) для определения $u^h(x, y)$ записывается в форме (см. (18.36))

$$\begin{aligned} u(x, y) \approx u^h(x, y) \;\; = \;\; & \sum_{k=1}^{n_v} u(\boldsymbol{q}_k)\varphi_k^v(x, y) + \sum_{k=1}^{n_m} u(\boldsymbol{m}_k)\varphi_k^m(x, y) + \\ & + \sum_{k=1}^{n_t} u(\boldsymbol{q}_{k_b})\beta_k(x, y), \end{aligned} \tag{18.42}$$

где $\varphi_k^v(x,y)$, $\varphi_k^m(x,y)$ — соответственно, базисные функции для вершин и для середин сторон (см. (18.34), (18.35)).

В терминах FreeFem++ пространство V_h задается кодом — `Vh(Th,P2b)`.

18.3 Векторнозначные FE-функции

Все ранее описанные конечные элементы являлись скалярными функциями $\mathbb{R}^2 \to \mathbb{R}$. В FreeFem++ можно задавать векторные конечноэлементные пространства и векторные FE-функции $\mathbb{R}^2 \to \mathbb{R}^N$, $N \geqslant 2$. В этом случае можно использовать, например, следующий код

```
fespace Vh(Th,[P1,P1]);
```

Размерность векторного пространства может быть любой, т. е. имеется возможность записывать

```
fespace Vh(Th,[P1,P1,P1]);
```

Более того, конечные элементы, на основе которых конструируется векторное пространство, могут быть различными. Например,

```
fespace Vh(Th,[P0,P1]);
```

Приведем код программы, демонстрирующий некоторые особенности работы с векторными пространствами.

Пример 18.1 (**Векторные пространства**).

```
1 mesh Th = square(5,5,[1-2*x,1-2*y]);
2 fespace Vh(Th, [P2,P2]);
3 Vh [Ux, Uy];
4 [Ux, Uy] = [x, -y];          // векторная FE функция
5 plot([Ux,Uy], wait=1);
```

Результат работы кода показан на рис. 18.6.

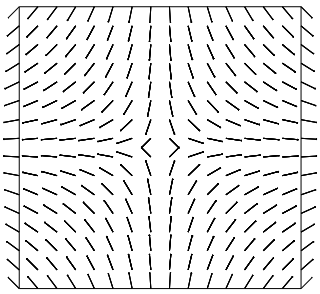

Рис. 18.6. Векторное поле (x,y)

✓. *Обратим внимание, что компоненты векторного поля можно присваивать функциям (см. строку 7), но задавать отдельно компоненты векторного поля нельзя (см. строку 8).*

```
mesh Th = square(5,5);
fespace Vh(Th, [P2,P2]);
fespace Xh(Th,P2);
Vh [Ux, Uy];
Xh p;
[Ux,Uy] = [x,-y]; // векторная FE функция
p = Ux;
// Ux = p;     // Ошибка: невозможно присвоить только одну компоненту
```

18.3.1 RT0 элементы

В языке FreeFem++ имеются специальные конечные элементы для работы с векторными пространствами. Это так называемые кусочно-постоянные конечные элементы Равьяра-Томаса — `RT0`.

Для каждого треугольника $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ определим векторы $\boldsymbol{e}_{k_1}$, $\boldsymbol{e}_{k_2}$, $\boldsymbol{e}_{k_3}$ (стороны или ребра треугольника), нормальные к ним векторы $\boldsymbol{n}_{k_1}$, $\boldsymbol{n}_{k_2}$, $\boldsymbol{n}_{k_3}$ и середины сторон треугольника (см. рис. 18.7) $\boldsymbol{m}_{k_1}$, $\boldsymbol{m}_{k_2}$, $\boldsymbol{m}_{k_3}$

$$\boldsymbol{e}_{k_1} = \operatorname{sgn}(k_2 - k_3)(\boldsymbol{q}_{k_2} - \boldsymbol{q}_{k_3}), \quad \boldsymbol{n}_{k_1} = \frac{\boldsymbol{k} \wedge \boldsymbol{e}_{k_1}}{|\boldsymbol{e}_{k_1}|}, \quad \boldsymbol{m}_{k_1} = \frac{1}{2}(\boldsymbol{q}_{k_2} + \boldsymbol{q}_{k_3}),$$

$$\boldsymbol{e}_{k_2} = \operatorname{sgn}(k_3 - k_1)(\boldsymbol{q}_{k_3} - \boldsymbol{q}_{k_1}), \quad \boldsymbol{n}_{k_2} = \frac{\boldsymbol{k} \wedge \boldsymbol{e}_{k_2}}{|\boldsymbol{e}_{k_2}|}, \quad \boldsymbol{m}_{k_2} = \frac{1}{2}(\boldsymbol{q}_{k_3} + \boldsymbol{q}_{k_1}),$$

$$\boldsymbol{e}_{k_3} = \operatorname{sgn}(k_1 - k_2)(\boldsymbol{q}_{k_1} - \boldsymbol{q}_{k_2}), \quad \boldsymbol{n}_{k_3} = \frac{\boldsymbol{k} \wedge \boldsymbol{e}_{k_3}}{|\boldsymbol{e}_{k_3}|} \quad \boldsymbol{m}_{k_3} = \frac{1}{2}(\boldsymbol{q}_{k_1} + \boldsymbol{q}_{k_2}).$$

Здесь $\boldsymbol{k}$ — единичный вектор, нормальный плоскости (x, y), $|\boldsymbol{e}_{k_1}|$, $|\boldsymbol{e}_{k_2}|$, $|\boldsymbol{e}_{k_3}|$ — длины сторон треугольника.

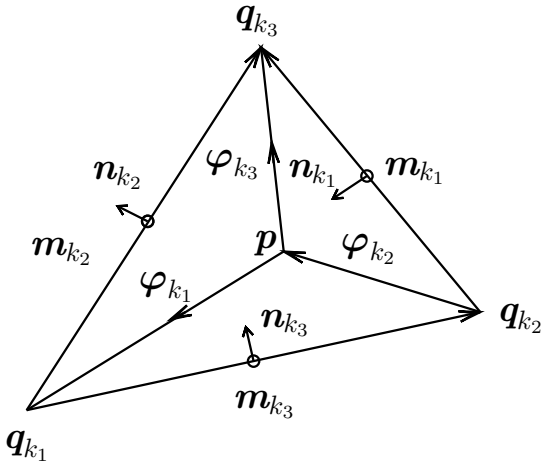

Рис. 18.7. Векторное поле (x, y)

Три базисные функции, отвечающие треугольнику T_k, зададим в виде

$$\boldsymbol{\varphi}_{k_1}(x,y) = \frac{\operatorname{sgn}(k_2-k_3)}{2S_k}(\boldsymbol{p}-\boldsymbol{q}_{k_1}),$$
$$\boldsymbol{\varphi}_{k_2}(x,y) = \frac{\operatorname{sgn}(k_3-k_1)}{2S_k}(\boldsymbol{p}-\boldsymbol{q}_{k_2}),$$
$$\boldsymbol{\varphi}_{k_3}(x,y) = \frac{\operatorname{sgn}(k_1-k_2)}{2S_k}(\boldsymbol{p}-\boldsymbol{q}_{k_3}),$$

где

$$S_k = |S(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{q}_{k_3})|, \quad S(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{q}_{k_3}) = \frac{1}{2}\det\begin{pmatrix} x_{k_1} & x_{k_2} & x_{k_3} \\ y_{k_1} & y_{k_2} & y_{k_3} \\ 1 & 1 & 1 \end{pmatrix},$$

$$\boldsymbol{q}_{k_1} = (x_{k_1}, y_{k_1}), \quad \boldsymbol{q}_{k_2} = (x_{k_2}, y_{k_2}), \quad \boldsymbol{q}_{k_3} = (x_{k_3}, y_{k_3}), \quad \boldsymbol{p} = (x,y) \in T_k.$$

Соотношение для определения векторнозначной функции $\boldsymbol{u}(\boldsymbol{p}) = \boldsymbol{u}(x,y)$ имеет следующий вид

$$\boldsymbol{u}(x,y) \approx \boldsymbol{u}^h(x,y) = \sum_{k=1}^{n_t}\sum_{s=1}^{3}(\boldsymbol{n}_{k_s}\cdot\boldsymbol{u}(\boldsymbol{m}_{k_s}))|\boldsymbol{e}_{k_s}|\boldsymbol{\varphi}_{k_s}(x,y). \tag{18.43}$$

Непосредственной подстановкой легко проверить, что

$$\boldsymbol{n}_{k_1}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_1}) = 1, \quad \boldsymbol{n}_{k_2}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_1}) = 0, \quad \boldsymbol{n}_{k_3}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_1}) = 0,$$
$$\boldsymbol{n}_{k_1}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_2}) = 0, \quad \boldsymbol{n}_{k_2}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_2}) = 1, \quad \boldsymbol{n}_{k_3}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_2}) = 0,$$
$$\boldsymbol{n}_{k_1}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_3}) = 0, \quad \boldsymbol{n}_{k_2}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_2}) = 0, \quad \boldsymbol{n}_{k_3}\cdot\boldsymbol{\varphi}(\boldsymbol{m}_{k_3}) = 1.$$

Вычисляя интеграл по стороне треугольника $\boldsymbol{e}_{k_s}$ (поток векторного поля $\boldsymbol{u}$), получим (простейшая квадратурная формула с узлом в середине стороны)

$$\int\limits_{\boldsymbol{e}_{k_s}} \boldsymbol{n}\cdot\boldsymbol{u}(x,y)\,ds \approx (\boldsymbol{n}_{k_s}\cdot\boldsymbol{u}(\boldsymbol{m}_{k_s}))|\boldsymbol{e}_{k_s}|.$$

Это, в частности, означает, что степенями свободы конечного элемента (коэффициенты перед базисными функциями (18.43)) будут потоки векторного поля $\boldsymbol{u}$ через границы треугольника.

Приведем код программы, демонстрирующий использование векторнозначных конечных элементов `RT0`.

Пример 18.2 (**Векторные пространства. Элементы** `RT0`).

```
mesh Th = square(5, 5, [1-2*x,1-2*y]);
fespace Vh(Th, RT0);
Vh [Ux, Uy];
[Ux, Uy] = [x, -y];          // векторная FE функция
plot([Ux,Uy], wait=1);
```

18.4 Загружаемые конечные элементы

Язык FreeFem++ постоянно обновляется и новые версии могут содержать новые конечные элементы, помимо перечисленных на с. 209. Кроме этого, имеется возможность создавать собственные конечные элементы. Подробно способ создания новых конечных элементов описан в [1]. Для этого требуется знание языка C++ и, в случае OS Windows, наличие специального эмулятора `cygwin` для компиляции кодов языка C++, на основе которых создаются библиотеки `*.dll`.

Некоторые такие библиотеки уже созданы разработчиками FreeFem++ и содержатся в каталоге, в котором установлен FreeFem++. К сожалению, новые конечные элементы не документированы, как и элементы `P1dc`, `P2dc`, `RTmodif`, имеющиеся в стандартной версии. Впрочем, часто по названию элементов нетрудно понять к какому именно виду они относятся.

Перечислим некоторые конечные элементы, которые можно подключать с помощью `dll` библиотек.

`P3`	— `Element_P3.dll`	;	скалярный ($\mathbb{R}^2 \to \mathbb{R}$);
`P3dc`	— `Element_P3dc.dll`	;	скалярный ($\mathbb{R}^2 \to \mathbb{R}$);
`P4`	— `Element_P4.dll`	;	скалярный ($\mathbb{R}^2 \to \mathbb{R}$);
`P4dc`	— `Element_P4dc.dll`	;	скалярный ($\mathbb{R}^2 \to \mathbb{R}$);
`P2BR`	— `BernadiRaugel.dll`	;	векторный ($\mathbb{R}^2 \to \mathbb{R}^2$);
`P2Morley`	— `Morley.dll`	;	векторный ($\mathbb{R}^2 \to \mathbb{R}^3$).

Подключение библиотек осуществляется при помощи ключевого слова `load` с указанием имении файла (без расширения).

Пример 18.3 (**Векторные пространства. Элементы** `P2RB`)**.**

```
load "BernadiRaugel";  // загрузка библиотеки
mesh Th = square(5, 5, [1-2*x, 1-2*y]);
fespace Vh(Th, P2BR);
Vh [Ux, Uy];
[Ux, Uy] = [x, -y]; //
plot([Ux, Uy], wait=1);
```

Результат работы кода показан на рис. 18.8.

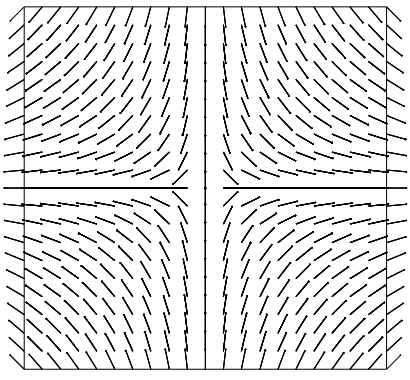

Рис. 18.8. Векторное поле (x, y)

Отметим, что конечный элемент `P2Morley` существенно отличается от всех остальных, имеющихся в языке FreeFem++. При использовании этого конечного элемента следует иметь ввиду, что *одновременно осуществляется аппроксимация функции и ее первых производных.*

```
load "Morley";  // загрузка библиотеки
mesh Th = square(5, 5, [1-2*x, 1-2*y]);
fespace Vh(Th, P2Morley);
Vh [U, Ux, Uy];
```

В приведенном фрагменте вектором `[U,Ux,Uy]` будет аппроксимироваться функция u и частные производные u_x, u_y. Пример решения бигармонического уравнения с использованием элементов `P2Morley` имеется в файле `bilapMorley.edp`, входящем в комплект установки FreeFem++ (см. также пример 19.7 на с. 231).

18.5 Численное интегрирование

Для вычисления интегралов по области и по границе области в языке FreeFem++ используется численное интегрирование на основе квадратурных формул Гаусса различного порядка точности.

18.5.1 Интегрирование по границе области

Пусть задана ограниченная область D с границей Γ ($\Gamma = \partial D$) и имеется триагуляция области

$$T_h = \bigcup_{k=1}^{N} T_k \approx D, \quad \mathrm{mes}(T_j \cap T_i) = 0, \quad j \neq i.$$

Для границы триангуляции T_h используем обозначение

$$\Gamma_h = \partial T_h.$$

Пусть $[\boldsymbol{q}_i, \boldsymbol{q}_k]$ некоторый сегмент (отрезок), принадлежащий границе Γ_h. Понятно, что $[\boldsymbol{q}_i, \boldsymbol{q}_k]$ является одной из сторон треугольника T_k, которая принадлежит границе. Точка (x, y), лежащая на сегменте $[\boldsymbol{q}_i, \boldsymbol{q}_k]$, определяется параметром t, $0 \leqslant t \leqslant 1$ (см. (18.12))

$$x(t) = (1-t)x_i + tx_j, \quad y(t) = (1-t)y_i + ty_j,$$

$$\boldsymbol{q}_i = (x_i, y_i), \quad \boldsymbol{q}_j = (x_j, y_j).$$

Интеграл на каждом сегменте $\Gamma_{ij} = [\boldsymbol{q}_i, \boldsymbol{q}_k]$ определяется квадратурой

$$\int_{\Gamma_{ij}} f(x,y)\, ds = |\boldsymbol{q}_i \boldsymbol{q}_k| \sum_{m=1}^{M} \omega_m f(x(t_m), y(t_m)), \tag{18.44}$$

где t_m — узлы квадратурной формулы, ω_m — коэффициенты квадратурной формулы, $|\boldsymbol{q}_i \boldsymbol{q}_k|$ — длина сегмента, M — количество узлов.

В табл. 18.1 приведены значения t_m и ω_m. В колонках таблицы указаны также наименование формулы (`qfe`) и ее порядок (`qforder`). Последняя колонка таблицы указывает для полиномов какой степени (P_k) квадратурная формула является точной.

Для вычисления интеграла по границе Γ_h можно использовать одно из следующих выражений

$$\begin{aligned}\int\limits_{\Gamma_h} f(x,y)\,ds &= \texttt{int1d(Th)(f)}\\ &= \texttt{int1d(Th, qfe= *)(f)}\\ &= \texttt{int1d(Th, qforder = *)(f)}\end{aligned} \tag{18.45}$$

Здесь вместо звездочки следует использовать наименование формулы или ее порядок из табл. 18.1.

✓. *По умолчанию выбрана квадратурная формула, точная для полиномов степени 5, т. е.* `qfe=qf3pE` *или* `qforder=6`.

M	имя (`qfe=`)	порядок (`qforder=`)	t_m	ω_m	точна для $P_k, k=$
1	qf1pE	2	0	1	1
2	qf2pE	3	$\frac{1}{2}\left(1 \mp \sqrt{\frac{1}{3}}\right)$	$\frac{1}{2}$	3
3	**qf3pE**	6	$\frac{1}{2}\left(1 \mp \sqrt{\frac{3}{5}}\right)$ $\frac{1}{2}$	$\frac{5}{18}$ $\frac{8}{18}$	5
4	qf4pE	8	$\frac{1}{2}\left(1 \mp \frac{\sqrt{525+70\sqrt{30}}}{35}\right)$ $\frac{1}{2}\left(1 \mp \frac{\sqrt{525-70\sqrt{30}}}{35}\right)$	$\frac{18-\sqrt{30}}{72}$ $\frac{18+\sqrt{30}}{72}$	7
5	qf5pE	10	$\frac{1}{2}\left(1 \mp \frac{\sqrt{245+14\sqrt{70}}}{21}\right)$ $\frac{1}{2}$ $\frac{1}{2}\left(1 \mp \frac{\sqrt{245-14\sqrt{70}}}{21}\right)$	$\frac{322-13\sqrt{70}}{1800}$ $\frac{64}{225}$ $\frac{322+13\sqrt{70}}{1800}$	9
2	qf1pElump	2	∓ 1	$\frac{1}{2}$	1

Таблица 18.1. Параметры квадратурных формул для интегралов по границе, см. [1]

В случае, когда необходимо вычислить интеграл по участку (фрагменту) Γ_1 границы Γ_h, можно использовать одно из следующих выражений

$$\begin{aligned}\int\limits_{\Gamma_1} f(x,y)\,ds &= \texttt{int1d(Th, 1)(f)}\\ &= \texttt{int1d(Th, 1, qfe= *)(f)}\\ &= \texttt{int1d(Th, 1, qforder=*)(f)}\end{aligned} \tag{18.46}$$

Здесь полагается, что участок границы Γ_1 помечен меткой 1 (`label=1`)

```
border Gamma1(t=0,1){ x=t; y=0; label=1; };
```

Пример 18.4 (**Вычисление длины полуокружности**).

```
real Length,HulfLength;
border C1(t=0,pi)   { x=cos(t); y=sin(t); label=1; };
border C2(t=pi,2*pi){ x=cos(t); y=sin(t); label=2; };
mesh Th = buildmesh(C1(50) + C2(50));
fespace Vh(Th, P2);
Length = int1d(Th)(1.0);
HulfLength = int1d(Th,1)(1.0);
cout << "Length      =  " << Length  << endl;
cout << "HulfLength  =  " << HulfLength  << endl;
```

Результатом работы кода будет

```
Length      =  6.28215
HulfLength  =  3.14108
```

Для вычисления интеграла по нескольким (возможно несвязным) участкам границы, например, Γ_1, Γ_3, используется одно из следующих выражений (границы помечены метками 1, 3)

$$\begin{aligned}\int\limits_{\Gamma_1\cup\Gamma_3} f(x,y)\,ds &= \texttt{int1d(Th, 1, 3)(f)}\\ &= \texttt{int1d(Th, 1, 3, qfe= *)(f)}\\ &= \texttt{int1d(Th, 1, 3, qforder= *)(f)}\end{aligned} \tag{18.47}$$

18.5.2 Интегрирование по области

Пусть задана ограниченная область D с границей Γ ($\Gamma = \partial D$) и имеется триангуляция области

$$T_h = \bigcup_{k=1}^{N} T_k \approx D, \quad \operatorname{mes}(T_j \cap T_i) = 0, \quad j \neq i.$$

Для каждого треугольника $T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ введем барицентрические координаты (п. 18.1.1 и формулы (18.3)–(18.8))

$$\xi = \lambda_1^k = \frac{S(\boldsymbol{p}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})}{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})}, \quad \eta = \lambda_2^k = \frac{S(\boldsymbol{q}_{k_1}, \boldsymbol{p}, \boldsymbol{q}_{k_3})}{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})},$$

$$\lambda_3^k = \frac{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{p})}{S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})} = 1 - \xi - \eta, \tag{18.48}$$

$$S(\boldsymbol{p}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3}) = \frac{1}{2}\det\begin{pmatrix} x & x_{k_2} & x_{k_3} \\ y & y_{k_2} & y_{k_3} \\ 1 & 1 & 1 \end{pmatrix}, \quad S(\boldsymbol{q}_{k_1}, \boldsymbol{p}, \boldsymbol{q}_{k_3}) = \frac{1}{2}\det\begin{pmatrix} x_{k_1} & x & x_{k_3} \\ y_{k_1} & y & y_{k_3} \\ 1 & 1 & 1 \end{pmatrix},$$

$$S(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{p})=\frac{1}{2}\det\begin{pmatrix}x_{k_1}&x_{k_2}&x\\y_{k_1}&y_{k_2}&y\\1&1&1\end{pmatrix},\quad S(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{q}_{k_3})=\frac{1}{2}\det\begin{pmatrix}x_{k_1}&x_{k_2}&x_{k_3}\\y_{k_1}&y_{k_2}&y_{k_3}\\1&1&1\end{pmatrix},$$

$$\boldsymbol{q}_{k_1}=(x_{k_1},y_{k_1}),\quad \boldsymbol{q}_{k_2}=(x_{k_2},y_{k_2}),\quad \boldsymbol{q}_{k_3}=(x_{k_3},y_{k_3}),\quad \boldsymbol{p}=(x,y)\in T_k.$$

Интеграл на каждом треугольнике $T_k=(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{q}_{k_3})$ определяется квадратурной формулой

$$\int\limits_{T_k} f(x,y)\,dx\,dy=S(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{q}_{k_3})\sum_{m=1}^{M}\omega_m f(x_m,y_m),\tag{18.49}$$

$$\begin{aligned}x_m&=\xi_m x_{k_1}+\eta_m x_{k_2}+(1-\xi_m-\eta_m)x_{k_3},\\ y_m&=\xi_m y_{k_1}+\eta_m y_{k_2}+(1-\xi_m-\eta_m)y_{k_3},\end{aligned}\tag{18.50}$$

где (ξ_m,η_m) — узлы квадратурной формулы (в барицентрических координатах), ω_m — коэффициенты квадратурной формулы, $S(\boldsymbol{q}_{k_1},\boldsymbol{q}_{k_2},\boldsymbol{q}_{k_3})$ — площадь треугольника, M — количество узлов.

В табл. 18.2 приведены значения (ξ_m,η_m) и ω_m. В колонках таблицы указаны также наименование формулы (`qft`) и ее порядок (`qforder`). Последняя колонка таблицы указывает для полиномов какой степени (P_k) квадратурная формула является точной.

M	имя (`qft=`)	порядок (`qforder=`)	(ξ,η)	ω_m	точна для P_k, $k=$
1	qf1pT	2	$\left(\frac{1}{3},\frac{1}{3}\right)$	1	1
3	qf2pT	3	$\left(\frac{1}{2},\frac{1}{2}\right)$, $\left(\frac{1}{2},0\right)$, $\left(0,\frac{1}{2}\right)$	$\frac{1}{3}$	2
7	**qf5pT**	6	$\left(\frac{1}{3},\frac{1}{3}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{9+2\sqrt{15}}{21}\right)$ $\left(\frac{9+2\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{9-2\sqrt{15}}{21}\right)$ $\left(\frac{9-2\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$	0.225 $\frac{(155-\sqrt{15})}{1200}$ $\frac{(155-\sqrt{15})}{1200}$ $\frac{(155-\sqrt{15})}{1200}$ $\frac{(155+\sqrt{15})}{1200}$ $\frac{(155+\sqrt{15})}{1200}$ $\frac{(155+\sqrt{15})}{1200}$	5
3	qf1pTlump		$(0,0)$, $(1,0)$, $(0,1)$	$\frac{1}{3}$	1
9	qf2pT4P1		$\left(\frac{1}{4},\frac{3}{4}\right)$, $\left(\frac{3}{4},\frac{1}{4}\right)$, $\left(0,\frac{1}{4}\right)$ $\left(0,\frac{3}{4}\right)$, $\left(\frac{1}{4},0\right)$, $\left(\frac{3}{4},0\right)$ $\left(\frac{1}{4},\frac{1}{4}\right)$, $\left(\frac{1}{4},\frac{1}{2}\right)$, $\left(\frac{1}{2},\frac{1}{4}\right)$	$\frac{1}{12}$ $\frac{1}{12}$ $\frac{1}{6}$	1

Таблица 18.2. Параметры квадратурных формул для интегралов по области, см. [1].

Для вычисления интеграла по области T_h используется одно из следующих выражений

$$\begin{aligned}\int\limits_{T_h} f(x,y)\,dx\,dy &= \texttt{int2d(Th)(f)}\\ &= \texttt{int2d(Th, qft= *)(f)}\\ &= \texttt{int2d(Th, qforder= *)(f)}\end{aligned} \qquad (18.51)$$

Здесь вместо звездочки следует использовать наименование формулы или ее порядок из табл. 18.2.

✓. *По умолчанию выбрана квадратурная формула, которая точна для полиномов степени 5, т. е.* `qfe=qf5pT` *или* `qforder=6`.

✓. *Для вычисления (*18.49*) достаточно использовать (*18.50*), формулу для площади треугольника* $S(\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ *и табл.* 18.2. *Бóльшая часть формул (*18.48*) приведена лишь для справки.*

Пример 18.5 (**Вычисление площади круга**).

```
1 real S;
2 border C0(t=0,2*pi)   { x=cos(t); y=sin(t); };
3 mesh Th = buildmesh(C0(50));
4 fespace Vh(Th, P2);
5 S = int2d(Th)(1.0);
6 cout << "S  =  " << S  << endl;
7 cout << "S  =  " << Th.area  << endl;
```

Результатом работы кода будет (точность расчета можно повышать, увеличивая количество узлов триангуляции)

```
S  =  3.13333
S  =  3.13333
```

Заметим, что значение площади можно получить, используя `Th.area`, как это сделано в строке 7.

Глава 19

Способы записи и решения задач в языке FreeFem++

Почти во всех примерах, приведенных в гл. 3–15, использовались наиболее простые конструкции языка FreeFem++. На самом деле FreeFem++ предоставляет пользователю много различных возможностей при выборе способов записи решаемых задач и методов их решения.

Напомним, что FreeFem++ ориентирован на решение двумерных краевых (и начальных) задач для уравнений в частных производных, которые должны быть записаны в слабой (вариационной) формулировке. В общем случае, при использовании операторной формы записи это означает следующее.

Пусть дано операторное уравнение

$$Lu = f, \quad L : \mathbb{X} \to \mathbb{Y}, \quad u \in \mathbb{X}, \quad f \in \mathbb{Y}, \quad (x, y) \in D,$$

где L — линейный (дифференциальный) оператор, $\mathbb{X}$, $\mathbb{Y}$ — для определенности гильбертовы пространства (при выборе пространства $\mathbb{X}$ предполагается, что краевые условия для u выполнены автоматически), $D \subset \mathbb{R}^2$ — область, в которой рассматривается задача.

Вычисляя скалярное произведение исходного уравнения на тестовую функцию v, получим слабую формулировку задачи

$$(Lu, v) = (f, v), \quad \forall\, v$$

или

$$A(u, v) - b(v) = 0, \tag{19.1}$$

$$A(u, v) \stackrel{\text{def}}{=} (Lu, v), \quad b(v) \stackrel{\text{def}}{=} (f, v),$$

где $A(u, v)$ — билинейная форма, $b(v)$ — линейная форма.

Именно задача в виде (19.1) является основной конструкцией, с которой работает язык FreeFem++. Билинейные и линейные формы задаются при помощи интегрирования. Иными словами, скалярное произведение определяется следующим образом

$$(u, v) = \int\limits_D (u(x, y)v(x, y)\, dx\, dy.$$

19.1 Ключевые слова problem и solve

Если имеется слабая формулировка задачи в виде (19.1), то для ее решения используются следующие конструкции

```
1 problem IDProblem(u,v) =
2                            A(u,v) - b(v)
3                            + (boundary condition);
```

или

```
1 solve IDProblem(u,v) =
2                          A(u,v) - b(v)
3                          + (boundary condition);
```

Здесь имеется ввиду, что строки 2, 3, конечно же, записаны в терминах языка FreeFem++, `IDProblem` — идентификатор задачи.

Различие в ключевых словах `problem` и `solve` заключается в том, что использование `solve` приведет к непосредственному решению задачи, а использование `problem` означает, что задача просто определена и для ее решения необходимо записать строку кода (только лишь идентификатор без параметров `(u,v)`) в той части программы, где требуется решение

```
1 problem IDProblem(u,v) =
2                            A(u,v) - b(v)
3                            + (boundary condition);
4 ...
5 IDProblem;
6 ...
```

19.1.1 Слабая форма задачи и краевые условия

Для того, чтобы проиллюстрировать конкретный способ записи задачи в слабой форме, рассмотрим решение задачи для уравнения Лапласа (см. также п. 3.1). Пусть имеется задача

$$-\Delta u = f, \quad u\big|_{\Gamma_1} = g_1, \quad \left.\frac{\partial u}{\partial n}\right|_{\Gamma_2} = g_2, \quad \left.\left(\frac{\partial u}{\partial n} + \beta u\right)\right|_{\Gamma_3} = g_3. \tag{19.2}$$

Здесь $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 = \partial D$.

Умножая на тестовую функцию v и интегрируя, с учетом формулы Грина получим

$$\iint\limits_D \nabla v \cdot \nabla u \, dx\, dy - \int\limits_{\Gamma_1} v \frac{\partial u}{\partial n}\, ds - \int\limits_{\Gamma_2} v \frac{\partial u}{\partial n}\, ds - \int\limits_{\Gamma_3} v \frac{\partial u}{\partial n}\, ds - \iint\limits_D f v \, dx\, dy = 0. \tag{19.3}$$

Исключая при помощи краевых условий производную $\partial u/\partial n$ и накладывая на функцию v дополнительное ограничение $v|_{\Gamma_1} = 0$, запишем

$$\iint\limits_D \nabla u \cdot \nabla v \, dx \, dy - \int\limits_{\Gamma_2} g_2 v \, ds - \int\limits_{\Gamma_3} (g_3 - \beta u) \, v \, ds - \iint\limits_D f v \, dx \, dy = 0. \quad (19.4)$$

Имеем следующие линейные и билинейные формы (после символа $\to$ приведена форма записи на языке FreeFem++).

Билинейная форма для уравнения Лапласа

$$A_e(u,v) = \iint\limits_D \nabla u \cdot \nabla v \, dx \, dy \to \texttt{int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v));}$$

Линейная форма для правой части уравнения

$$b_f(v) = \iint\limits_D f v \, dx \, dy \to \texttt{int2d(Th)(f*v);}$$

Линейная форма для краевых условий Неймана на границе Γ_2

$$b_2(v) = \int\limits_{\Gamma_2} g_2 v \, ds \to \texttt{int1d(Th,Gamma2)(g2*v);}$$

Линейная форма для краевых условий третьего рода на границе Γ_3

$$b_{31}(v) = \int\limits_{\Gamma_3} g_3 v \, ds \to \texttt{int1d(Th,Gamma3)(g3*v);}$$

Билинейная форма для краевых условий третьего рода на границе Γ_3

$$b_{32}(u,v) = \int\limits_{\Gamma_3} \beta u v \, ds \to \texttt{int1d(Th,Gamma3)(beta*u*v);}$$

Объединяя эти выражения в соответствии с (19.4), получим (19.1)

$$A(u,v) - b(v) = 0,$$

где

$$A(u,v) = A_e(u,v) + b_{32}(u,v), \quad b(v) = b_2(v) + b_{31}(v) + b_f(v).$$

Код на языке FreeFem++ имеет вид

```
problem P(u,v) = int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v) )
                 + int1d(Th,Gamma3)( beta*u*v )
                 - int1d(Th,Gamma2)( g2 * v )
                 - int1d(Th,Gamma3)( g3 * v )
                 - int2d(Th)( f * v )
                 + on( Gamma1, u = g1 ) ;
```

Здесь предполагается, что `Th` — идентификатор сетки, `Gamma1`, `Gamma2`, `Gamma3` — идентификаторы границ. Краевое условие Дирихле для контура Γ_1 задается при помощи ключевого слова `on`.

19.1.2 Параметры, влияющие на решение задачи

Ключевые слова `solve` и `problem` можно использовать с дополнительными параметрами, которые определяют метод решения системы линейных уравнений.

Перечислим некоторые из этих параметров (подробнее см. [1]).

`solver=` — определяет метод решения системы линейных уравнений. Допустимые значения — имена методов:

`LU, CG, Crout, Cholesky, GMRES, UMFPACK ...`

Более точно, имя метода предопределяет тип матрицы $A_{ij} = A(\varphi_i, \varphi_j)$, которая будет вычисляться по билинейной форме $A(u, v)$ и базисным функциям φ_i.

`LU` — матрица является несимметричной и ленточной, для решения используется метод LU-разложения;

`Cholesky` — матрица является симметричной, ленточной и положительно определенной, для решения используется метод Холецкого;

`Crout` — матрица является симметричной и ленточной, для решения используется метод Краута;

`CG` — матрица является разреженной и симметричной, для решения используется итерационный метод сопряженных градиентов;

`GMRES` — матрица является разреженной, для решения используется метод минимальных невязок;

`UMFPACK` — матрица является разреженной, метод решения — несимметричный мультифронтальный метод для разреженных систем линейных уравнений[1];

✓. *Метод* `UMFPACK` *используется в FreeFem++ по умолчанию.*

`eps=` точность вычислений (`real`) в итеративных методах типа `CG` и `GMRES`. Можно задавать абсолютную и относительную погрешность вычислений.

При $\varepsilon < 0$ имеется ввиду абсолютная точность и остановка вычислений происходит при выполнении условий

$$\|Ax - b\| < |\varepsilon|.$$

При $\varepsilon > 0$ имеется ввиду относительная точность и остановка вычислений происходит при выполнении условий

$$\|Ax - b\| < \varepsilon \|Ax_0 - b\|.$$

`init=` (`boolean`). Если `init= false` или 0 матрица при вычислениях восстанавливается, т. е. перерасчитывается заново. Заметим, что восстановление матрицы происходит и в случае изменения размеров сетки;

`tgv=` некоторая большая величина (по умолчанию 10^{30}), используемая для реализации краевых условий Дирихле.

`strategy=` (`integer`) устанавливает стратегию решения уравнений в методе `UMFPACK` (по умолчанию параметр равен 0).

[1] Подробнее см. `http://www.cise.ufl.edu/research/sparse/umfpack/`

✓. *Имеются и другие параметры, в частности,* `tolpivot`, `tolpivotsym`, `precon` *(см. [1]).*

Пример 19.1 (**Выбор для решения задачи** `LU` **метода**).

```
problem P(u,v, solver=LU) = int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v) )
                            + int1d(Th,Gamma3)( beta*u*v )
                            - int1d(Th,Gamma2)( g2 * v )
                            - int1d(Th,Gamma3)( g3 * v )
                            - int2d(Th)( f * v )
                            + on( Gamma1, u = g1 ) ;
```

✓. *Указанные параметры* `eps`, `tgv`, `init` ... *перечисляются в* `problem` *или* `solve` *через запятую.*

Пример 19.2 (**Выбор для решения задачи** `GMRES` **метода**).

```
problem Laplace(u,v, solver=GMRES, eps=1.0e-10, tgv=1e30 )=
.............
```

19.2 Вариационные формы и разреженные матрицы

В языке FreeFem++ имеется возможность непосредственной записи линейных и билинейных форм и вычисления на их основе соответствующих матриц. Напомним схему метода конечных элементов (см. гл. 2, 3).

Пусть имеется слабая форма записи исходной задачи

$$A(u,v) - b(v) = 0, \quad \forall\, v\big|_\Gamma = 0. \tag{19.5}$$

Ищем решение в виде

$$u(x,y) = \sum_{k=1}^{n} u_k \varphi_k(x,y), \quad \varphi_i\big|_\Gamma = 0, \quad i = 1,\ldots,n, \tag{19.6}$$

где φ_i — базисные функции.

Подставляя (19.6) в (19.5) и выбирая в качестве v базисные функции, получим систему алгебраических уравнений для определения u_k

$$\sum_{k=1} A_{ik} u_k = b_i, \quad i = 1,\ldots,n, \quad A_{ik} = A(\varphi_i, \varphi_k), \quad b_i = b(\varphi_i). \tag{19.7}$$

Решаем систему линейных уравнений

$$\sum_{k=1} A_{ik} u_k = b_i, \quad Au = b, \quad u = A^{-1} b. \tag{19.8}$$

Подчеркнем, что при численной реализации система линейных уравнений решается при помощи какого-либо алгоритма и обратная матрица A^{-1} не определяется.

Язык FreeFem++ позволяет почти дословно воспроизвести указанные действия.

Пример 19.3 (**Решение задачи Дирихле для уравнения Лапласа**). Пусть, для определенности, поставлена задача

$$-\Delta u = f, \quad (x,y) \in D = [0,1]\times[0,1], \quad u\big|_{\Gamma} = 0. \tag{19.9}$$

В этом случае билинейная и линейная формы будут

$$A(u,v) = \iint\limits_D \nabla u \cdot \nabla v \, dx\, dy \to \texttt{int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v));}$$

$$b(v) = \iint\limits_D f v \, dx\, dy \to \texttt{int2d(Th)(f*v);}$$

Определяем область D, создаем триангуляцию `Th` области D и задаем пространство конечных элементов `Vh`

```
mesh Th = square(25,25,[-1+2*x,-1+2*y]);
fespace Vh(Th, P2);
```

Создание билинейной и линейной (вариационных) форм осуществляется при помощи ключевого слова `varf`

```
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
              + on(1, 2, 3, 4, u=0);
varf b(unused,v) = int2d(Th)(f*v);
```

✓. *Заметим, что величины* `u`, `v` *являются формальными параметрами и их можно предварительно не описывать. Напротив, величина* `f` *должна быть предварительно описана.*

Фрагмент кода является правильным (`f` заранее описана)

```
Vh f;
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
              + on(1, 2, 3, 4, u=0);
varf b(unused,v) = int2d(Th)(f*v);
Vh u;
```

Фрагмент кода является неправильным (`f` описана после того как использована)

```
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
              + on(1, 2, 3, 4, u=0);
varf b(unused,v) = int2d(Th)(f*v);
Vh u,f;
```

✓. *Обратим внимание на использование ключевого слова* `unused` *в качестве параметра в выражении* `varf b(unused,v)`. *Дело в том, что синтаксис языка формально позволяет задавать лишь формы с двумя аргументами.*

Вычисление матрицы `A` и вектора правых частей `F` системы линейных уравнений осуществляется при помощи кода

```
matrix A = a(Vh, Vh);
F[] = b(0,Vh);
```

Наконец, решение системы уравнений $Au = F$ записывается в виде

```
u[] = A^-1*F[];
```

Приведем полный код для решения задачи (19.9)

```
1  mesh Th = square(25,25,[-1+2*x,-1+2*y]);
2  fespace Vh(Th, P2);
3  Vh u, F, f = sin(pi*x)*sin(pi*y);
4  varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
5                + on(1, 2, 3, 4, u=0);
6  varf b(unused,v) = int2d(Th)(f*v);
7  matrix A = a(Vh,Vh);
8  F[] = b(0,Vh);
9  u[] = A^-1*F[];
10 plot(u, wait=1);
```

✓. *Формальный параметр* `v` *в выражениях* `a(u,v)` *и* `b(unused,v)` *не требует описания, т. к. нигде явно не используется.*

Результат работы показан на рис. 19.1.

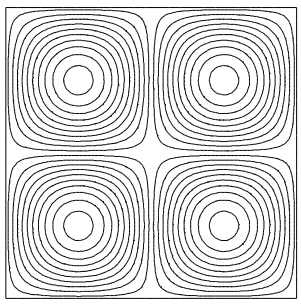

Рис. 19.1. Решение задачи Дирихле для уравнения Лапласа

Точно такой же результат будет получен и в случае традиционного способа решения

```
1  mesh Th = square(25, 25, [-1+2*x,-1+2*y]);
2  fespace Vh(Th, P2);
3  Vh u, v, f = sin(pi*x)*sin(pi*y);
4  solve P(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))-int2d(Th)(f*v)
5                 + on(1, 2, 3, 4, u=0);
6  plot(u, wait=1);
```

✓. *Параметры* `solver`, `eps`, `init`, `tgv`, `eps`,..., *применяемые с ключевыми словами* `solve` *и* `problem`, *можно использовать при записи вариационных форм (с ключевым словом* `varf`*) и создании матриц (с ключевым словом* `matrix`*).*

Пример 19.4 (**Использование параметров со словом** `varf` **или** `matrix`).

```
varf a(u, v, solver=CG) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
                          + on(1, 2, 3, 4, u=0);
varf b(unused, v) = int2d(Th)(f*v);
matrix A = a(Vh, Vh);
```

```
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
              + on(1, 2, 3, 4, u=0);
varf b(unused, v) = int2d(Th)(f*v);
matrix A = a(Vh, Vh, solver=GMRES);
```

19.3 Ключевое слово macro

Имеется еще одна возможность записи решаемых задач — с использованием ключевого слова `macro`.

Пример 19.5 (**Использование ключевого слова** `macro`).

```
mesh Th = square(25, 25, [-1+2*x,-1+2*y]);
fespace Vh(Th, P2);
Vh uh, vh, f = sin(pi*x)*sin(pi*y);
macro Laplace(u,v) ( dx(u)*dx(v) + dy(u)*dy(v) ) // конец записи
macro RightHand(u,v) ( u*v )                     // конец записи
solve P(uh,vh) = int2d(Th)( Laplace(uh,vh))-int2d(Th)(RightHand(f,vh))
                 + on(1, 2, 3, 4, uh=0);
plot(uh, wait=1);
```

Очевидно, что ключевое слово `macro` можно использовать для создания часто повторяющихся фрагментов программы. Фактически, это аналог процедуры с параметрами, например, для языка `Pascal`.

✓. *Обратим внимание, что запись* `macro` *должна заканчиваться символами* `//`*, а не как обычно символом* `;`.

В конструкции `macro` в качестве параметров можно использовать и имена других конструкций языка, например, имя билинейной формы.

Пример 19.6 (**Использование имени билинейной формы в** `macro`).

```
mesh Th = square(25, 25, [-1+2*x,-1+2*y]);
fespace Vh(Th,P2);
Vh uh, F, vh, f = sin(pi*x)*sin(pi*y);
varf a(u,v) = int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))
              + on(1, 2, 3, 4, u=0);
```

```
varf b(unused, v) = int2d(Th)(f*v);
macro Laplace(L,u,v) (L(u,v)) //
matrix A = Laplace(a,Vh,Vh);
F[] = b(0,Vh);
uh[] = A^-1*F[];
plot(uh, wait=1);
```

Приведем еще один пример использования ключевого слова `macro` и одновременно продемонстрируем возможности использования конечного элемента `P2Morley`.

Пример 19.7 (**Решение задачи для бигармонического уравнения**). Пусть дана задача

$$\Delta^2 u = f, \quad (x,y) \in \overline{D} = [0,1]\times[0,1], \quad u\big|_\Gamma = 0, \quad \left.\frac{\partial u}{\partial n}\right|_\Gamma = 0. \qquad (19.10)$$

В этом случае итегрирование по частям с учетом краевых условий дает билинейную форму

$$\iint\limits_D v\Delta^2 u\,dx\,dy = -\iint\limits_D \frac{\partial v}{\partial x_i}\frac{\partial^3 u}{\partial x_i \partial^2 x_k}\,dx\,dy = \iint\limits_D \frac{\partial^2 v}{\partial x_i \partial x_k}\frac{\partial^2 u}{\partial x_i \partial x_k}\,dx\,dy,$$

$$A(u,v) = \iint\limits_D (u_{xx}v_{xx} + 2u_{xy}v_{xy} + u_{yy}v_{yy})\,dx\,dy.$$

```
load "Morley"
mesh Th = square(40, 40);
fespace Vh(Th, P2Morley);      // пространство конечных элементов
macro bilaplacien(u,v)(dxx(u)*dxx(v)+dyy(u)*dyy(v)+2.*dxy(u)*dxy(v)) //
real f = 1;
Vh [u,ux,uy],[v,vx,vy];
solve bilap([u,ux,uy],[v,vx,vy]) =
  int2d(Th)(  bilaplacien(u,v) )
 - int2d(Th)(f*v)
 + on(1,2,3,4,u=0,ux=0,uy=0); // следует задавать все величины u,ux,uy
plot(u, wait=1, cmm="u");
plot(ux, wait=1, cmm="u_x");
plot(uy, wait=1, cmm="u_y");
```

Результаты расчетов представлены на рис. 19.2

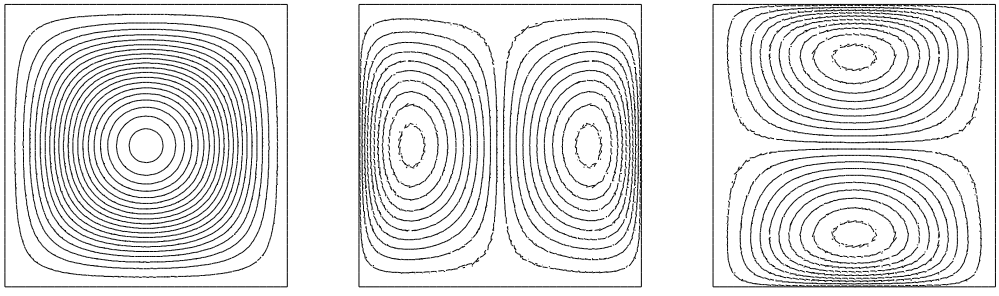

Рис. 19.2. Изолинии функций u, u_x и u_y, соответственно

✓. *В языке FreeFem++ можно использовать ключевые слова* `dxx`, `dyy`, `dxy`, *соответствующие производным* $(\ )_{xx}$, $(\ )_{yy}$. *В описании языка [1] эти ключевые слова не указаны, но их использование имеется в примере* `bilap.edp` *(каталог* `..FreeFem++\examples++`*).*

Наличие в языке FreeFem++ ключевого слова `macro` открывает возможность упрощения записи многих задач.

Следующий фрагмент кода показывает как завести оператор `Grad`, соответствующий оператору ∇ в исходных задачах и, в частности, записать интеграл по области

$$\int_D \nabla u \cdot \nabla v \, dx \, dy$$

почти в привычной математической форме

```
1 macro Grad(u) [dx(u),dy(u)] // конец макро
2 problem A(u,v) = int2d(Th)( Grad(u)' * Grad(v) );
```

В FreeFem++ имеется также ключевое слово `include`, позволяющее вставлять в текст программы тексты из любых файлов, что позволяет в частности, создавать библиотеки обозначений.

Например, строка 1 предыдущего кода могла быть записана в файле с именем `MyLibrary.txt` и вставлена в программу при помощи `include`

```
1 include "MyLibrary.txt";
2 problem A(u,v) = int2d(Th)( Grad(u)' * Grad(v) );
```

Глава 20

Задачи на собственные значения

Задачи об определении собственных значений и собственных функций для краевых задач имеют важное значение в математической физике. FreeFem++ предоставляет пользователю возможность исследования таких задач. Встроенные функции языка базируются на `arpack++`[1]— объектно-ориентированной версии пакета `ARPACK`, предназначенного для нахождения собственных значений и собственных векторов.

В этой главе средствами FreeFem++ решается задача определения собственных значений оператора Лапласа в случае краевых условий Дирихле.

20.1 Функция EigenValue

Функция `EigenValue` предназначена для определения собственных значений λ обобщенной задачи на собственные значения для системы линейных алгебраических уравнений

$$Au = \lambda Bu, \tag{20.1}$$

где A, B — матрицы.

Вызов функции для вычисления набора приближенных собственных значений λ осуществляется, например, следующим образом

```
int k = EigenValue(OP, B, nev= , sigma= );
```

Здесь

`OP = A - sigma B` — матрица (`matrix`);

`B` — матрица (`matrix`);

`sigma` — начальное приближение для какого-либо λ;

`nev` — количество вычисляемых λ, ближайших к `sigma`.

Матрицы `OP`, `B` должны быть разреженными матрицами (т. е. не двумерными массивами). Способ создания таких матриц, связанных с краевыми задачами для уравнений в частных производных, описан в п. 17.2 и может быть, например, следующим

[1] http://www.caam.rice.edu/software/ARPACK/

```
varf op(u1,u2) = int2d(Th)( dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma*u1*u2 )
                 + on(1,2,3,4,u1=0) ;     // краевые условия
varf b([u1],[u2]) = int2d(Th)( u1*u2 ) ;  // краевые условия отсутствуют
matrix OP = op(Vh, Vh, solver=Crout, factorize=1);
// можно выбирать solver и параметры
matrix B = b(Vh, Vh, solver=CG, eps=1e-20);
```

Функция `EigenValue` может использоваться с дополнительными параметрами, которые перечислены ниже.

`sym=` задача на собственные значения является симметричной, т. е. все собственные значения вещественны;

`value=` массив для хранения вещественной части собственных значений;

`ivalue=` массив для хранения мнимой части собственных значений;

`vector=` массив для хранения собственных векторов. Должен быть описан как массив `FE`-функции, т. е. `Vh[int] F(nev);`

`rawvector=` двумерный массив (`real[int,int]`) для хранения собственных векторов в виде колонок матрицы. Для вещественных несимметричных задач комплексные собственные векторы представляются в виде двух последовательных векторов. Если имеется k-ое и $(k+1)$-ое комплексно сопряженные собственные значения, то k-ый вектор будет содержать вещественную часть, а $(k+1)$-ый — мнимую часть соответствующего комплексно сопряженного собственного вектора;

`tol=` относительная погрешность вычисления собственных значений;

`maxit=` максимальное количество итераций при вычислении (уточнении) собственных значений.

20.2 Собственные значения оператора Лапласа

Рассмотрим задачу на собственные значения для оператора Лапласа в случае краевых условий Дирихле (краевые условия первого рода)

$$-\lambda u = \Delta u, \quad (x,y) \in D, \quad u\big|_\Gamma = 0. \tag{20.2}$$

Слабая форма задачи (20.2) получается, как обычно, путем умножения на тестовую функцию с последующим интегрированием по частям (заметим, что из этого соотношения вытекает неравенство $\lambda > 0$)

$$\lambda \iint\limits_D uv\,dx\,dy = \iint\limits_D \nabla u \cdot \nabla v\,dx\,dy, \quad \forall\, v\big|_\Gamma = 0. \tag{20.3}$$

Введем обозначения для билинейных форм и перепишем (20.3) в следующем виде

$$A(u,v) = \lambda B(u,v), \tag{20.4}$$

где

$$A(u,v) = \iint\limits_D \nabla v \cdot \nabla \psi\,dx\,dy, \quad B(u,v) = \iint\limits_D \psi v\,dx\,dy. \tag{20.5}$$

Приближенное решение задачи (20.4) ищем в виде линейной комбинации базисных функций

$$u(x,y) = \sum_{k=1}^{N} u_k \varphi_i(x,y). \tag{20.6}$$

Подставляя (20.6) в (20.4) и выбирая в качестве v базисные функции, получим задачу на собственные значения для системы линейных алгебраических уравнений

$$\sum_{k=1}^{N} A_{ik} u_k = \lambda \sum_{k=1}^{N} B_{ik} u_k, \quad i = 1, \ldots, N, \tag{20.7}$$

где

$$A_{ik} = A(\varphi_i, \varphi_k), \quad B_{ik} = B(\varphi_i, \varphi_k). \tag{20.8}$$

20.2.1 Задача для прямоугольника

Рассмотрим случай, когда D является прямоугольником с несоизмеримыми сторонами $a\pi$ и $b\pi$

$$D = [0, a\pi] \times [0, b\pi].$$

Собственные значения и соответствующие собственные функции для задачи (20.2) имеют вид

$$u^{nm}(x,y) = c_{mn} \sin\frac{nx}{a} \sin\frac{my}{b}, \quad \lambda_{mn} = \frac{n^2}{a^2} + \frac{m^2}{b^2}, \tag{20.9}$$

где c_{mn} — произвольные константы.

20.2.2 Задача для квадрата

В случае, когда D является прямоугольником с соизмеримыми сторонами $a\pi$, $b\pi$, в частности, квадратом, имеются кратные собственные значения. Пусть

$$D = [0, a\pi] \times [0, a\pi].$$

Тогда собственные значения и соответствующие собственные функции для задачи (20.2) имеют вид

$$u^{nm}(x,y) = c^{(1)}_{mn} \sin\frac{nx}{a} \sin\frac{my}{a} + c^{(2)}_{mn} \sin\frac{mx}{a} \sin\frac{ny}{a},$$

$$\lambda_{mn} = \frac{n^2 + m^2}{a^2},$$

где $c^{(1)}_{mn}$, $c^{(2)}_{mn}$ — произвольные константы.

20.3 Вычислительный эксперимент

Приведем алгоритм решения задачи на языке FreeFem++

```
//verbosity = 10;
real a1 = 1*pi,  b1 = 2*pi;
mesh Th = square(25, 25, [a1*x,b1*y]);
fespace Vh(Th, P2);
Vh u1, u2;
real sigma = 0; // начальное приближение
// определение матрицы OP = A - sigma B ;
varf op(u1,u2) = int2d(Th)(dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma*u1*u2)
                 + on(1, 2, 3, 4, u1=0) ;
varf b(u1,u2) = int2d(Th)( u1*u2 );    // определение матрицы B
matrix OP = op(Vh, Vh, solver=Crout, factorize=1);    // м-д Crout,
  // т.к. матрица не положительно определенная при выборе элементов P2
matrix B = b(Vh, Vh, solver=CG, eps=1e-20);
int nev = 20; // кол-во вычисляемых собств. значений, ближайших к sigma
real[int] ev(nev); // для сохранения собств. векторов
Vh[int] eV(nev);   // для сохранения собств. векторов (и визуализации)
int k = EigenValue(OP, B, sym=true, sigma=sigma, value=ev, vector=eV,
                   tol=1e-10, maxit=0, ncv=0);
for (int i=0; i<k; i++)   // организация визуализации собств. векторов
{
  u1=eV[i];
  // для вычисления погрешности
  real gg = int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1));
  real mm = int2d(Th)(u1*u1) ;
  cout << "---- " << i << " " << ev[i] << " err= " << gg-ev[i]*mm
       << " --- " << endl;
  plot(eV[i], bw=1, ps="Ris23_"+i+"", wait=1);
}
```

Результатом работы будет

```
Real symmetric eigenvalue problem: A*x - B*x*lambda
Thanks to ARPACK++ class ARrcSymGenEig
Real symmetric eigenvalue problem: A*x - B*x*lambda
Shift and invert mode  sigma=0
Eigenvalues:
  lambda[1]: 1.25
  lambda[2]: 2.00003
  lambda[3]: 3.25015
  ..................
  lambda[25]: 20.0118
---- 0 1.25     err= 1.31006e-014 ---
---- 1 2.00003  err= 1.28786e-014 ---
---- 2 3.25015  err= 1.86517e-014 ---
  ...........................
---- 24 20.0118 err= -3.55271e-014 ---
```

В табл. 20.1 для сравнения приведены точные и приближенные величины собственных значений.

k	FreeFem++	n	m	λ_{nm}		k	FreeFem++	n	m	λ_{nm}
1	1,25	1	1	1,25		14	11,2521	3	3	11,25
2	2,00003	1	2	2,00		15	13,0047	3	4	13,00
3	3,25015	1	3	3,25		16	13,0093	3	6	13,00
4	4,25005	2	1	4,25		17	13,2619	1	7	13,25
5	5,00019	2	2	5,00		18	15,2597	3	5	15,25
6	5,0006	1	4	5,00		19	16,2518	4	1	16,25
7	6,25061	2	3	6,25		20	16,2692	2	7	16,25
8	7,25192	1	5	7,25		21	17,0033	4	2	17,00
9	8,00169	2	4	8,00		22	17,0251	1	8	17,00
10	9,25039	3	1	9,25		23	18,0187	3	6	18,00
11	10,0009	3	2	10,00		24	18,2563	4	3	18,25
12	10,0051	1	6	10,00		25	20,0118	4	4	20,00
13	10,2542	2	5	10,25						

Таблица 20.1. Собственные значения

На рис. 20.1 показаны собственные функции в случае прямоугольника $\overline{D} = [0, \pi] \times [0, 2\pi]$. Стороны прямоугольника соизмеримы и имеются кратные собственные значения, в частности, $\lambda_{42} = \lambda_{24}$ и $\lambda_{15} = \lambda_{51}$.

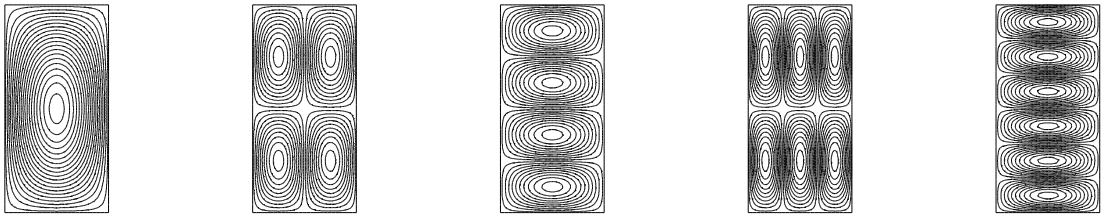

Рис. 20.1. Изолинии собственных функций для $\lambda_{42} = \lambda_{24}$ и $\lambda_{15} = \lambda_{51}$

На рис. 20.2 в случае квадрата $\overline{D} = [0, \pi] \times [0, \pi]$ приведены собственные функции. Имеются кратные собственные значения, в частности, $\lambda_{43} = \lambda_{34}$ и $\lambda_{52} = \lambda_{25}$.

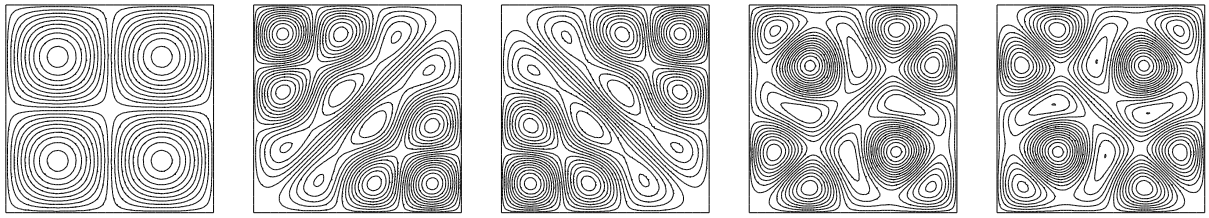

Рис. 20.2. Изолинии собственных функций для λ_{22}, $\lambda_{43} = \lambda_{34}$ и $\lambda_{52} = \lambda_{25}$

Заметим, что в случае прямоугольника собственные функции для кратных собственных значений, например, $\lambda_{42} = \lambda_{24}$, вычислены в виде

$$\sin 4x \sin 2y \quad \text{и} \quad \sin 2x \sin 4y.$$

В случае квадрата вычисления, например, для $\lambda_{43} = \lambda_{34}$, дают собственные функции в виде

$$\sin 4x \sin 3y + \sin 3x \sin 4y \quad \text{и} \quad \sin 4x \sin 3y - \sin 3x \sin 4y.$$

Глава 21

Визуализация результатов расчетов

21.1 Визуализация с помощью средств FreeFem++

Визуализация результатов расчетов на экране в виде сеток, изолиний конечно-элементных функций и векторных полей возможна с помощью команды FreeFem++ `plot`, которая, помимо простейших функций управления графическим выводом, позволяет также сохранять полученное изображение в postscript файлы.

Параметрами команды `plot` могут быть сетки, FE-функции, двумерные массивы FE-функций, трехмерные вещественные массивы, функции, векторные поля. Например, изображения триангуляции квадрата области и линий уровня FE-функции $f(x, y) = xy$ могут быть получены с помощью пары команд `plot` (строки 4 и 5).

```
1 mesh Th = square(10,10);
2 fespace Vh(Th, P2);
3 Vh u = x*y;
4 plot(Th);  // рисование сетки (параметр команды - имя сетки)
5 plot(u);   // рисование FE-функции (параметр команды - имя FE-функции)
```

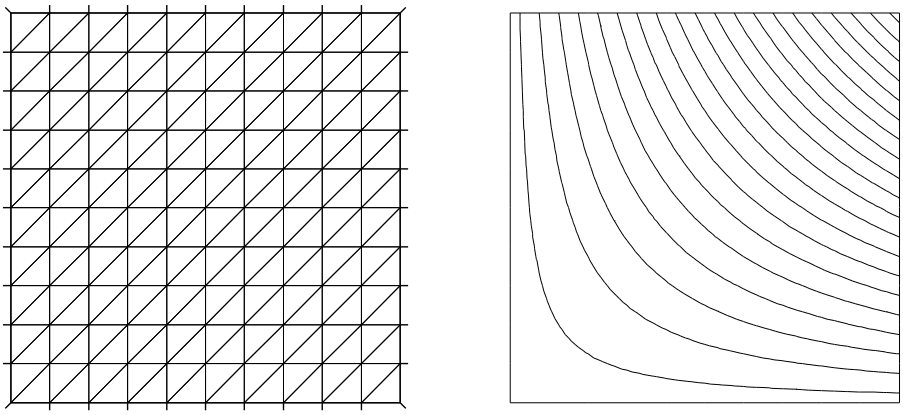

Рис. 21.1. Сетка 10×10 и линии уровня функции $f(x, y) = xy$

21.1.1 Параметры команды `plot`

`aspectratio=` (`bool`) выражение, отвечающее за растягивание области по всему графическому окну (`=false`, $= 0$[1]);

`bb=` массив от 2D массива (типа `[[0.1,0.2],[0.5,0.6]]`), задающий область окна графического вывода. Окно задается координатами угловых точек — `[0.1,0.2]` и `[0.5,0.6]` в нашем случае;

`boundary=` (`bool`) выражение, отвечающее за способ вывода границ области (`=true` — рисовать границы);

`bw=` (`bool`) выражение, отвечающее за способ вывода изображения: с использованием черно-белой (`true`) или цветной (`=false`) палитры;

`cmm=` (`string`) строковое выражение, выдаваемое в графическое окно;

`coef=` (`real`) коэффициент, равный отношению единицы длины стрелок векторов к единице длины области;

`fill=` (`bool`) заливка областей между изолиниями ($= 0$ — нет заливки);

`grey=` (`bool`) выражение, отвечающее за способ вывода изображения: с использованием оттенков серого (1) или нет ($= 0$);

`hsv=` (`real[int]`) вещественный массив, определяющий $3 \times n$ значений параметров цветовой модели HSV:

```
real[int] Colors = [h1,s1,v1, h2,s2,v2, ..., hn,sn,vn],
```

где `hi`, `si`, `vi` — параметры цветовой модели HSV (H — Hue (оттенок), S — Saturation (насыщенность), V — Value (значение яркости)). Каждый параметр — вещественное число из отрезка $[0, 1]$;

`nbarrow=` (`int`) количество цветов векторных значений ($= 20$);

`nbiso=` (`int`) количество выводимых изолиний ($= 20$);

`ps=` (`string`) выражение для сохранения рисунка в postscript-файл (с расширением `.ps`);

`value=` (`bool`) выражение, отвечающее за возможность вывода «легенды» (`value=1` — показывать легенду, отражающую значения изолиний и направления векторов);

`varrow=` (`real[int]`) множество номеров выводимых цветных стрелок;

`viso=` (`real[int]`) массив значений c_k, где $f(x, y) = c_k$. Служит для построения выбранных изолиний;

`wait=` (`boolean`) выражение ожидания команды (`=false`). Если `true`, то программа прервет выполнение до нажатия клавиши на клавиатуры или щелчка мыши, при этом при нажатии любой клавиши на клавиатуре, кроме нижеперечисленных, происходит продолжение работы программы. Специальными считаются символы:

`+` увеличить область вокруг указателя мыши,

`-` уменьшить область вокруг указателя мыши,

`=` восстановить начальный размер области вывода,

`c` увеличить коэффициент длины стрелок,

[1] Здесь и далее в скобках после знака равенства указаны значения по умолчанию.

C уменьшить коэффициент длины стрелок,
r восстановить (перерисовать) графическое окно,
f включить/выключить заливку между изолиниями,
b выбрать черно-белый цвет изображения,
g выбрать палитру: оттенки серого или цветная,
m показать/убрать сетку,
v показать/убрать легенду,
p сохранить изображение в postscript файл (с именем `freefem.ps` в каталог с текущим `edp`-файлом),
? вывести подсказку о ключах.

Пример 21.1 (**сохранение изображения в файл**). Обратим внимание, что ключи параметра `wait` дублируют действия некоторых параметров команды `plot`. Например, сохранить изображение в ps-файл с именем `freefem.ps` можно с помощью ключа `p` в случае использования команды

```
plot(u, wait=1);
```

и при использовании параметра `ps`

```
plot(u, ps="freefem.ps");
```

Существенное отличие состоит в том, что во втором случае можно задать другое имя для файла. Это особенно удобно, когда результаты расчетов сохраняются, например, в цикле:

```
1  func f = tanh(exp(x)^2-exp(y)^2);
2  mesh Th = square(5,5);
3  fespace Vh(Th, P1);
4  Vh fh = f;
5  plot(fh, ps="TestFuncInit.ps");
6  plot(Th, ps="Test0.ps");
7  for (int i=1; i<3; i++)
8  { Th = adaptmesh(Th, fh);
9    fh = f;              // функция на новой сетке
10   plot(Th, ps="Test" + i + ".ps");
11 }
12 plot(fh, ps="TestFunc.ps");
```

При работе программы сохраняются пять файлов: `TestFuncInit.ps`, `Test0.ps`, `Test1.ps`, `Test2.ps`, `TestFunc.ps` (см. рис. 21.2, 21.3).

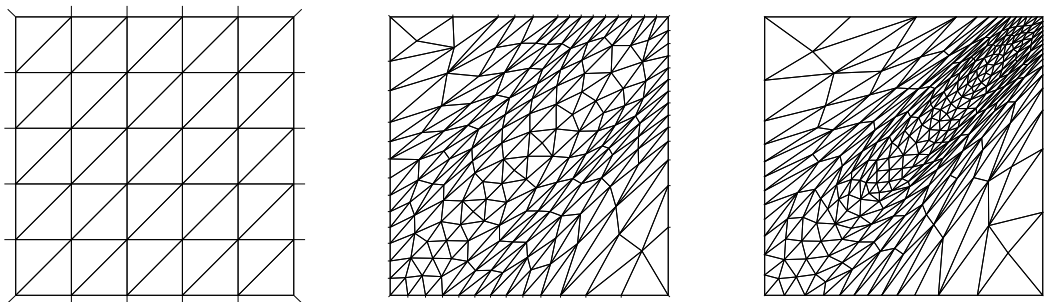

Рис. 21.2. Сетка в области $[0;1] \times [0;1]$ (файлы `Test0.ps`, `Test1.ps`, `Test2.ps`)

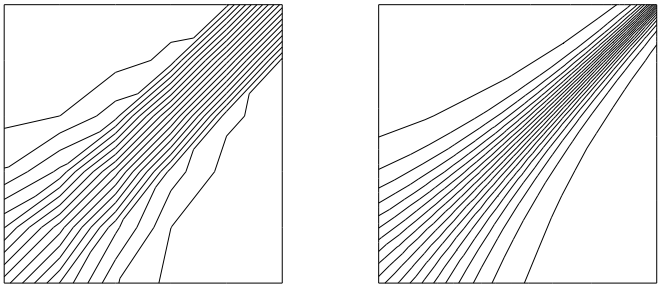

Рис. 21.3. Линии уровня функции $f(x, y)$ (файлы `TestFuncInit.ps`, `TestFunc.ps`)

Пример 21.2 (**изображение изолиний FE-функций**). Будем изображать изолинии некоторой функции, но выбранные не по умолчанию, а специальным образом (нас интересуют значения, различающиеся между собой на шаг 0,1). Для этого будем использовать параметр `viso`, который позволяет выбрать отдельные изолинии. Включение параметра `value` (`=1`) дает легенду — таблицу значений функции на изолиниях. Результат работы программы см. на рис. 21.4.

```
1 mesh Th = square(25,25);
2 fespace Vh(Th, P2);
3 Vh u = exp(-10*(x-0.5)^2)*exp(-10*(y-0.5)^2);
4 real[int] viso(8);
5 for (int i=0; i<viso.n; i++)
6   viso[i] = i*0.1;
7 plot(u, viso=viso(0:viso.n-1), value=1);
```

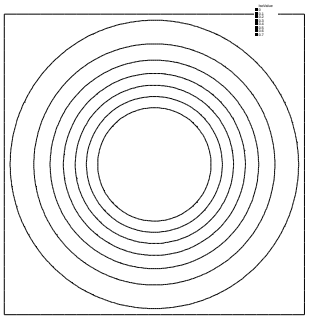

Рис. 21.4. Линии уровня функции $f(x, y) = c_i$, $c_i = 0{,}1i$, $i = 0, \ldots, 7$

Пример 21.3 (**изображение сечения FE-функции**). Для рисования сечения FE-функции значения аргументов и сеточной функции предварительно необходимо занести в массивы (строки 6–10).

```
1 real s = 50;              // количество точек графика
2 real[int] xx(s), uu(s);   // вспомогательные массивы
3 mesh Th = square(30, 30, [-1+2*x,-1+2*y]); // квадрат [-1,1]x[-1,1]
4 fespace Vh(Th, P2);
5 Vh uh = sin(2*pi*x) * cos(y); // задание сеточной функции uh
6 for (int i=0; i<s; i++)
```

```
7    { x=-1+2*i/s;     y=0;
8      xx[i]=i;  uu[i]=uh;
9    }               // занесение значений аргумента и функции в массивы
10 plot([xx,uu], wait=1);
```

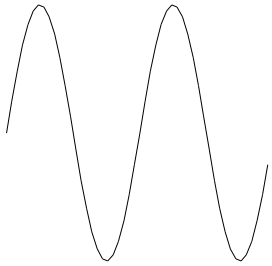

Рис. 21.5. График функции $f(x, y) = \sin 2\pi x \cdot \cos y$ при $y = 0$, $x \in [-1, 1]$

Пример 21.4 (**изображение векторного поля**). Для изображения векторного поля $\boldsymbol{F} = (Fx, Fy)$ необходимо задать сеточную функцию для каждой его компоненты (строка 3).

```
1  mesh Th = square(10,10,[-1+2*x,-1+2*y]);
2  fespace Vh(Th,P2);
3  Vh Fx = sin(2*pi*y),  Fy = -sin(2*pi*x);
4  plot([Fx,Fy], wait=1, coef=0.06);
```

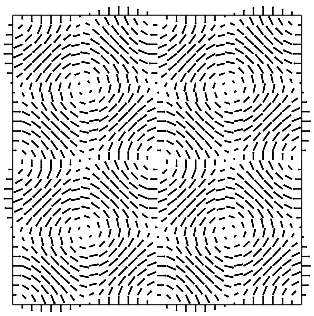

Рис. 21.6. Векторное поле $\boldsymbol{F} = (\sin(2\pi y), -\sin(2\pi x))$

Пример 21.5 (**параметр** `hsv`). Результат работы следующей программы сложно представить в черно-белом варианте.

```
1  real a=2.0, b=1.0;  // ширина и высота прямоугольной области
2  border Gamma2(t=0,1){ x=a*t; y=0; };          // bottom
3  border Gamma11(t=0,1){ x=a; y=b*t; };         // right
4  border Gamma3(t=0,1){ x=a*(1-t); y=b; };      // top
5  border Gamma12(t=0,1){ x=0; y=b*(1-t); };     // left
6  mesh Th = buildmesh(Gamma2(10)+Gamma11(5)+Gamma3(10)+Gamma12(5));
7  fespace Vh(Th, P2);
8  Vh u = 2*x;
9  plot(u, fill=1, wait=1);   // рисование до изменения параметра hsv
```

```
10 real[int] colorhsv = [ 0.75, 1.0, 1.0,
11                        0.5,  1.0, 0.75,
12                        0.0,  1.0, 1.0  ];
13 plot(u, fill=1, hsv=colorhsv, wait=1);
```

✓. *Команду* `plot` *можно использовать для вывода сразу нескольких функций (см., в частности, рис.* 12.2*). Например,*

```
plot(c1, c2, phi, nbiso=80);
```

будет выводить изолинии функций c_1, c_2, φ. *При этом количество изолиний, используемое* ***для всего*** *рисунка, будет определяться параметром* `nbiso`.

21.2 Визуализация с помощью программы medit

Программа medit[2] позволяет с помощью Open GL строить 3D-изображения по данным из текстовых файлов специального вида с расширениями `bb`, `faces`, `points`.

Для успешной работы с программой medit необходимо, чтобы в каталоге с запускающим файлом `medit.exe` находились файлы `glut32.dll`, `DEFAULT.medit`, а также файлы с данными (`*.bb`, `*.faces`, `*.points`). Программу medit можно запускать отдельно от FreeFem++, передавая ей имя файла для обработки, либо вызывая непосредственно в программе, написанной на языке FreeFem++.

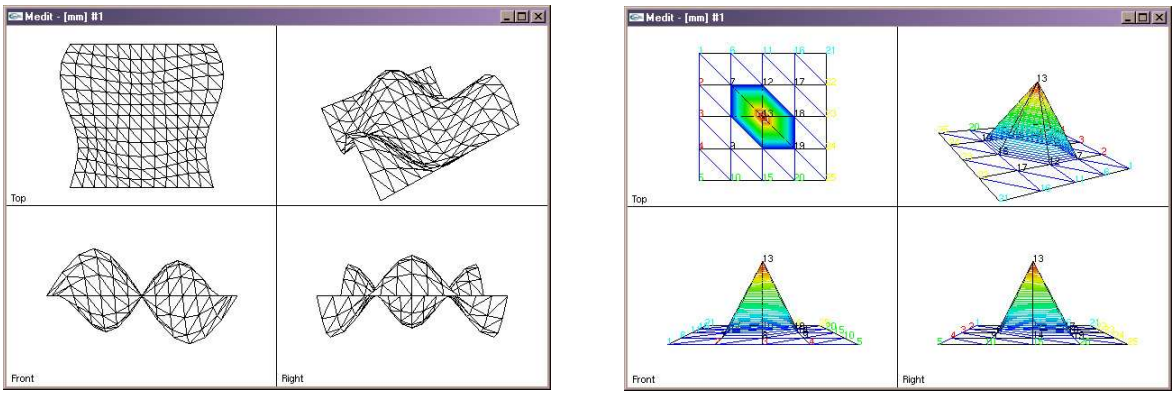

Рис. 21.7. Окно программы medit

Пример 21.6. Ниже приведен код, демонстрирующий сохранение данных для medit (строки 4–8), а также вызов программы medit из FreeFem++ (строка 9).

```
1 mesh Th = square(4*pi, 4*pi, [-1+2*x,-1+2*y]);
2 fespace Vh(Th, P2);
3 Vh u = 0.5 * sin(pi*x) * cos(pi*y);
4 savemesh(Th, "mm", [x,y,u]);
```

[2] http://www.ann.jussieu.fr/ frey/logiciels/medit.html

```
{ ofstream file("mm.bb");
  file << "2 1 1 "<< u[].n << " 2 \n";
  for (int j=0; j<u[].n; j++)  {  file << u[][j] << endl; }
}
exec("medit mm");  // запуск файла medit.exe на выполнение
```

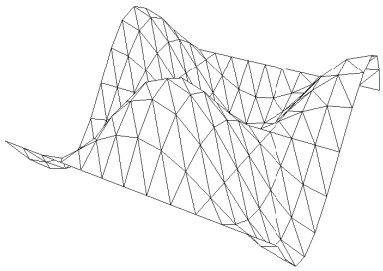

Рис. 21.8. Визуализация FE-функции с помощью программы medit

21.3 Визуализация с помощью программы gnuplot

В рабочий каталог из дистрибутива gnuplot[3] следует скопировать файлы: `wgnuplot.exe`, `wgnuplot.hlp`, `wgnuplot.mnu` и `prologue` (без этого файла не будут создаваться ps-файлы).

Пример 21.7. Приведем пример программного кода, позволяющего осуществить визуализацию результатов расчета с помощью программы gnuplot.

```
real r =10;
border C0(t=0,2*pi){ x=r*cos(t);  y=r*sin(t); };
mesh Th = buildmesh(C0(100));
fespace Vh(Th,P2);
// задание FE-функции
Vh  phi = sin(sqrt(x^2+y^2)) / sqrt(x^2+y^2);
plot(Th, phi, wait=true);
// сохранение данных о триангуляции и значений функции
{ ofstream ff("graph.txt");
 for (int i=0; i<Th.nt; i++)
  { for (int j=0; j <=2; j++)
     ff<<Th[i][j].x<<" "<<Th[i][j].y<<" "<<phi[][Vh(i,j)]<<endl;
     ff<<Th[i][0].x <<" "<<Th[i][0].y<<" "<<phi[][Vh(i,0)]<<"\n\n\n";
  }
}
// создание файла для программы gnuplot
{ ofstream ff("gnu_file.txt");
  ff << "set palette rgbformulae 30,31,32" << endl;
  ff << "unset clabel" << endl;
  ff << "splot \"graph.txt\" with lines pal" << endl;
  ff << "pause -1 \"Hit return to continue\"" << endl;
  ff << "set terminal postscript monochrome" << endl;
```

[3] http://www.gnuplot.info/

```
23    ff << "set output \"gnuplot\"" << endl;
24    ff << "replot" << endl;
25    ff << "reset" << endl;
26  }
27  // вызов wgnuplot для обработки файла gnu_file.txt
28  exec("wgnuplot \"gnu_file.txt\"");
```

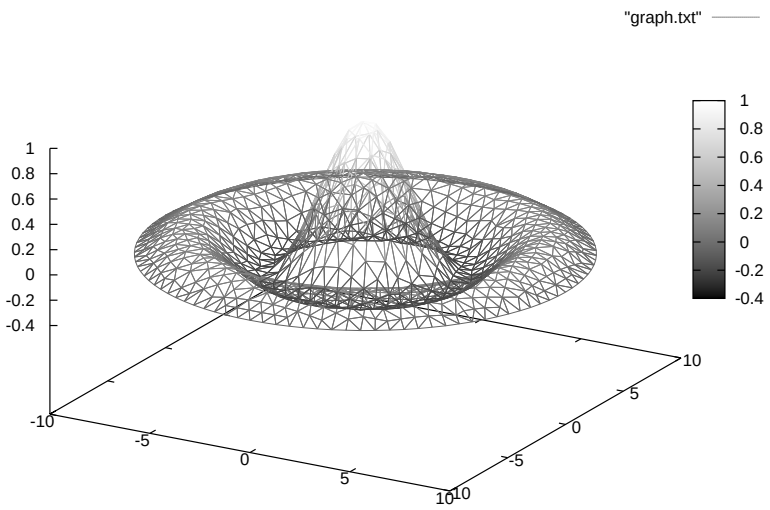

Рис. 21.9. Визуализация функции $F = \dfrac{\sin r}{r}$, $r = \sqrt{x^2 + y^2}$

В строке 6 задана функция $F = (\sin r)/r$, $r = \sqrt{x^2 + y^2}$. Данные о триангуляции и значениях функции заносятся в файл `graph.txt`. В строках 16–26 показан способ создания текстового файла:

для программы gnuplot

```
set palette rgbformulae 30,31,32
unset clabel
splot \"graph.txt\" with lines pal
pause -1 \"Hit return to continue\"
set terminal postscript monochrome
set output \"gnuplot\"
replot
reset
```

Полученный файл `gnu_file.txt` можно использовать и непосредственно в программе gnuplot для визуализации функции.

Подробное описание команд `wgnuplot` имеется в руководстве по работе с программой gnuplot.

Приложение А

Установка программного обеспечения

Комплект программ FreeFem++ можно найти по адресу:

`http://www.freefem.org/ff++/`.

Стандартный каталог в ОС Windows для установки FreeFem++:

`C:\Program Files\FreeFem++`.

В нем после инсталлирования пакета появятся необходимые исполняемые файлы (в частности, это `FreeFem++.exe`), библиотеки `dll` и каталоги, содержащие большое количество примеров.

✓. *Установку пакета FreeFem++ рекомендуется производить в каталог с именем, не содержащим символов пробела и кириллицы.*

Несмотря на то, что FreeFem++ имеет собственную IDE, лучше использовать его с более развитыми текстовыми оболочками, например, Crimson Editor.

А.1 Взаимодействие с текстовыми редакторами

Некоторые текстовые редакторы (TexnicCenter[1], Crimson Editor, WinEdt[2]) могут быть настроены на работу с пакетом FreeFem++, в частности, при подключении соответствующих файлов (которые могут быть скачаны с сайта FreeFem) в текстовых редакторах появляется возможность подключения цветовой схемы для подсветки FreeFem-синтаксиса.

Crimson Editor — бесплатная оболочка (IDE) для редактирования исходных текстов компьютерных программ. Поддерживает цветовое выделение синтаксиса исходных текстов таких языков программирования как HTML, C/C++, Perl и Java, а также файлов MATLAB[3] и LaTeX[4]. Домашняя страница Crimson Editor:

`http://www.crimsoneditor.com/`

[1] `http://www.toolscenter.org/`
[2] `http://www.winedt.org/`
[3] Пакет математического моделирования, основанный на применении матричных операций.
[4] Система для набора и верстки текстов с формулами.

После установки FreeFem++ (в частности, версии 2.17-1) в его основном каталоге будет находиться архив `crimon-freefem++.zip`, файлы из которого необходимо поместить в соответствующие подкаталоги (`link` и `spec`) каталога

`C:\Program Files\Crimson Editor`

(создается в результате установки программы Crimson Editor). Вызвав окно Preferences с помощью команд меню Tools → Conf. User Tools..., можно настроить трансляцию исходных файлов FreeFem++ (`.edp`) (см. рис. А.1).

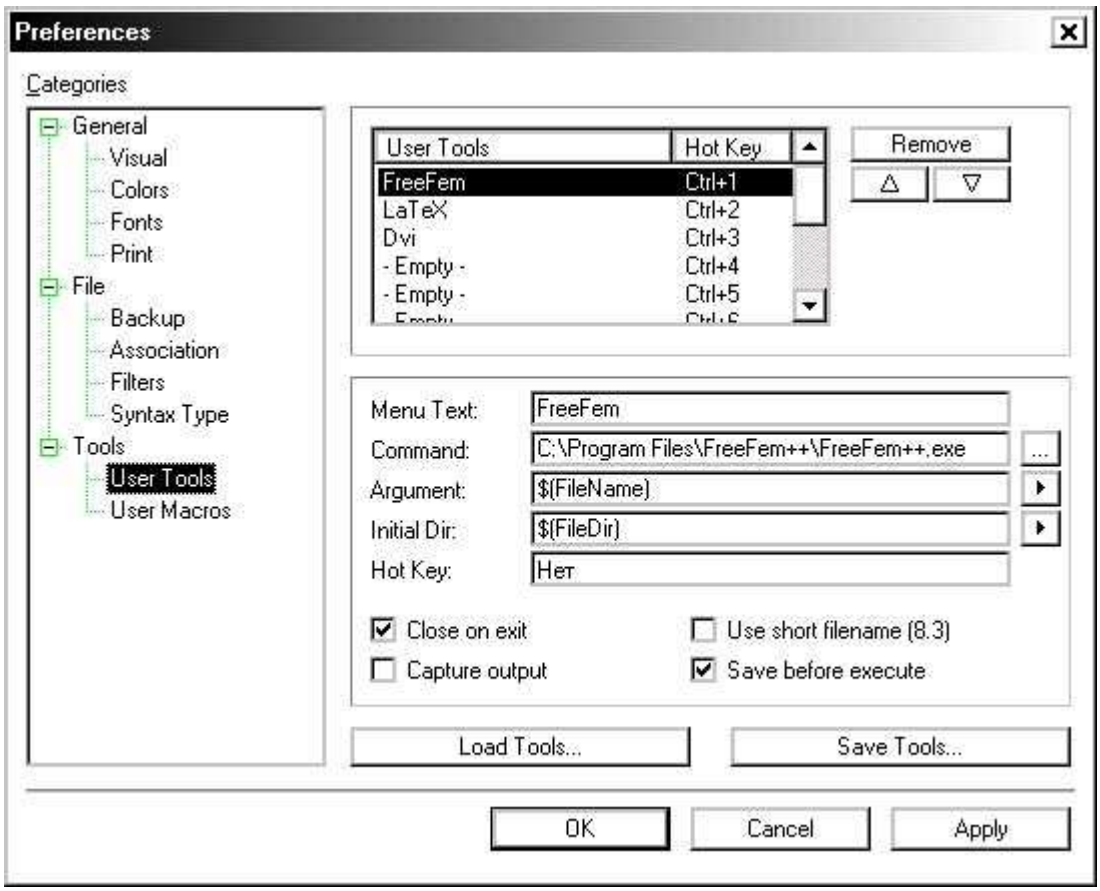

Рис. А.1. Настройка вызова FreeFem++ в оболочке Crimson Editor

Для вызова FreeFem++ из Crimson Editor необходимо в окне Preferences заполнить текстовые поля Menu Text, Command, Argument, Initial Dir следующим образом (см. также рис. А.1):

Menu Text:	FreeFem
Command:	указать полный путь на исполняемый файл `FreeFem++.exe`
Argument:	выбрать в списке аргумент File Name
Initial Dir:	выбрать File Directory

При желании можно поменять горячую клавишу (hot key), запускающую FreeFem++ на выполнение (по умолчанию Ctrl+1), нажав нужную клавишу на клавиатуре в поле «Hot key».

Приложение B

Используемые обозначения и формулы

D — область, $D \subset \mathbb{R}^2$,

Γ — граница области D, $\quad \Gamma = \partial D$,

T_h — триангуляция области D,

$T_k = (\boldsymbol{q}_{k_1}, \boldsymbol{q}_{k_2}, \boldsymbol{q}_{k_3})$ — треугольник с вершинами $\boldsymbol{q}_{k_1}$, $\boldsymbol{q}_{k_2}$, $\boldsymbol{q}_{k_3}$; $\quad T_k \in T_h$,

$[\boldsymbol{q}_i, \boldsymbol{q}_k]$ — сегмент, связывающий точки $\boldsymbol{q}_i$, $\boldsymbol{q}_k$,

$$\frac{\partial u}{\partial x} = \partial_x u = u_x; \quad \frac{\partial u}{\partial y} = \partial_y u = u_y,$$

$$\nabla u = (u_x, u_y, u_z),$$

$$\operatorname{div} \boldsymbol{a} = \nabla \cdot \boldsymbol{a} = \frac{\partial a_1}{\partial x} + \frac{\partial a_2}{\partial y} + \frac{\partial a_3}{\partial z}; \quad \boldsymbol{a} = (a_1, a_2, a_3),$$

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = u_{xx} + u_{yy} + u_{zz},$$

$$\operatorname{rot} \boldsymbol{v} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right), \quad \boldsymbol{v} = (u, v, w),$$

$$\iint\limits_D v \Delta u \, dx \, dy = - \iint\limits_D \nabla u \cdot \nabla v \, dx \, dy + \int\limits_\Gamma v \frac{\partial u}{\partial n} \, ds \text{ — формула Грина,}$$

$$\frac{\partial u}{\partial n} = \boldsymbol{n} \cdot \nabla u,$$

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla,$$

Цилиндрические координаты (r, θ, z), $\boldsymbol{v} = (u, v, w)$:

$$\operatorname{div} \boldsymbol{v} = \frac{1}{r}(ru)_r + \frac{1}{r} v_\theta + w_z, \quad \Delta = \frac{1}{r} \frac{\partial}{\partial r} r \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} + \frac{\partial^2}{\partial z^2},$$

$$\boldsymbol{v} \cdot \nabla = u(\ )_r + \frac{1}{r} v(\ )_\theta + w(\ )_z.$$

Приложение C

Список ключевых слов

```
adaptmesh
Cmatrix
R3
bool
border
break
buildmesh
catch
cin
complex
continue
cout
element
else
end
fespace
for
func
if
ifstream
include
int
intalledge
load
macro
matrix
mesh
movemesh
ofstream
plot
problem
real
return
savemesh
solve
string
try
throw
vertex
varf
while

int1d
int2d
on
square

dx, dxx
dy, dyy, dxy
convect
jump
mean

wait
ps
solver
CG
LU
UMFPACK
factorize
init
endl

x, y, z, pi, i
sin, cos, tan, atan, asin, acos
cotan, sinh, cosh, tanh, cotanh
exp, log, log10, sqrt
abs, max, min
```

Литература

1. *Hecht F., Pironneau O., Le Hyaric A., Ohtsuka K.* FreeFem++. Version 2.17-1. http://www.freefem.org/ff++.
2. *Марчук Г. И.* Методы вычислительной математики. М.: Наука, 1980.
3. *Марчук Г. И., Агошков В. И.* Введение в проекционно-сеточные методы. М.: Наука. Главная редакция физико-математической литературы, 1981.
4. *Бахвалов Н. С., Жидков Н. П., Кобельков Г. М.* Численные методы. М.: Наука, 1987.
5. *Деклу Ж.* Метод конечных элементов. М.: Мир, 1976.
6. *Зенкевич О.* Метод конечных элементов в технике. М.: Мир, 1975.
7. *Зенкевич О., Морган К.* Конечные элементы и аппроксимация. М.: Мир, 1986.
8. *Сабоннадьер Ж.-К., Кулон Ж.-Л.* Метод конечных элементов и САПР. М.: Мир, 1989.
9. *Оден Дж.* Конечные элементы в нелинейной механике сплошных сред. М.: Мир, 1976.
10. *Норри Д., де Фриз Ж.* Введение в метод конечных элементов. М.: Мир, 1981.
11. *Стренг Г., Фикс Дж.* Теория метода конечных элементов. М.: Мир, 1977.
12. *Митчелл Э., Уэйт Р.* Метод конечных элементов для уравнений с частными производными. М.: Мир, 1981.
13. *Михлин С. Г.* Вариационные методы в математической физике. М.: Наука, 1970.
14. *Михлин С. Г.* Численная реализация вариационных методов. М.: Наука, 1966.
15. *Лионс Ж.-Л., Мадженес Э.* Неоднородные граничные задачи и их приложения. М.: Мир, 1971.
16. *Ландау Л. Д., Лифшиц Е. М.* Гидродинамика. М.: Наука, 1986.
17. *Рождественский Б. Л., Яненко Н. Н.* Системы квазилинейных уравнений. М.: Наука, 1978.
18. *Уизем Дж.* Линейные и нелинейные волны. М.: Мир, 1977.
19. *Марри Дж.* Нелинейные дифференциальные уравнения в биологии. Лекции о моделях: Пер. с англ. М.: Мир, 1983.
20. *Chorin A.* A numerical method for solving incompressible viscous flow problems // J. Comput. Phys. 2 (1967). P. 12–26.
21. *Флетчер К.* Вычислительные методы в динамике жидкостей. Т. 1, 2. М.: Мир, 1991.
22. *Гершуни Г. З., Жуховицкий Е. М.* Конвективная устойчивость несжимаемой жидкости. М.: Наука, 1972.
23. *Rannacher R.* On Chorin's projection method for the incompressible Navier-Stokes equations, in 'Navier-Stokes Equations: Theory and Numerical Methods' (R. Rautmann, et al., eds.). Proc. Oberwolfach Conf., August 19–23, 1991. Springer, 1992.
24. *Бабский В. Г., Жуков М. Ю., Юдович В. И.* Математическая теория электрофореза. Киев: Наукова думка, 1983.
25. *Бабский В. Г., Жуков М. Ю.* Биофизические методы: Теоретические основы электрофореза. М.: Изд. МГУ, 1990.
26. *Жуков М. Ю.* Массоперенос электрическим полем. Ростов н/Д: Изд. РГУ, 2005.
27. Hecht F. The mesh adapting software: bamg. INRIA report 1998.

Список иллюстраций

Предметный указатель

Учебно-научное издание

ЖУКОВ Михаил Юрьевич
ШИРЯЕВА Елена Владимировна

ИСПОЛЬЗОВАНИЕ ПАКЕТА КОНЕЧНЫХ ЭЛЕМЕНТОВ FreeFem++ ДЛЯ ЗАДАЧ ГИДРОДИНАМИКИ, ЭЛЕКТРОФОРЕЗА И БИОЛОГИИ

Компьютерная верстка *М. Ю. Жукова, Е. В. Ширяевой*

Сдано в набор 05.06.08 г. Подписано в печать 05.06.08 г. Заказ № 053.
Тираж 100 экз. Формат 60*84 1/16. Печ. лист 16,0. Усл.печ.л. 14,8.
Типография Южного федерального университета.
344091, г. Ростов-на-Дону, пр. Стачки, 200/1, тел (863) 243-41-66.