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.

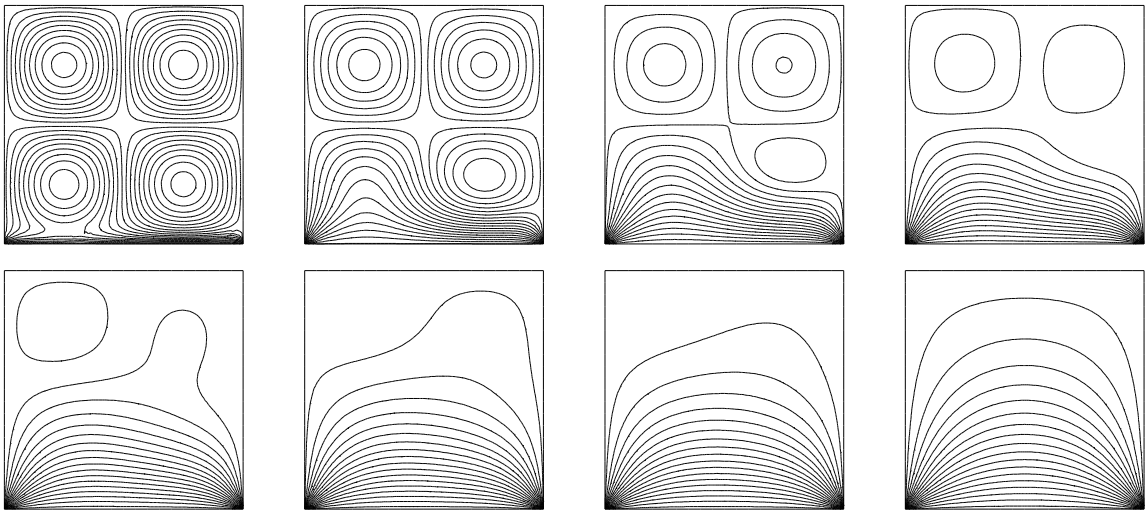

Рис. 4.1. Изолинии функции $u(x, y, t)$ при $t = 0$; 0,01; 0,02; 0,03; 0,04; 0,05; 0,06; 0,12

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.

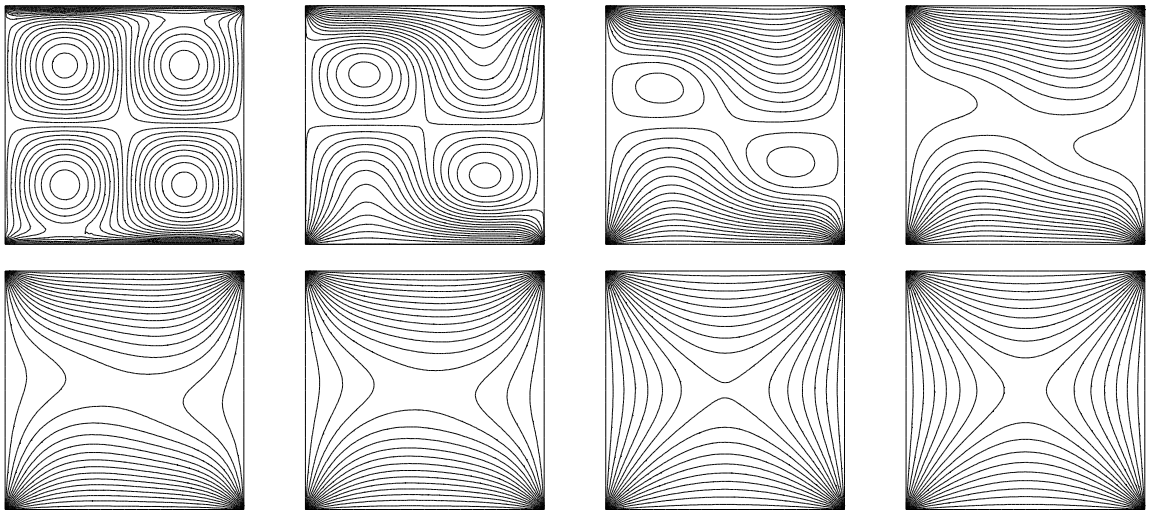

Рис. 4.2. Изолинии функции $u(x, y, t)$ при $t = 0$; 0,01; 0,02; 0,03; 0,04; 0,05; 0,11; 0,12

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.

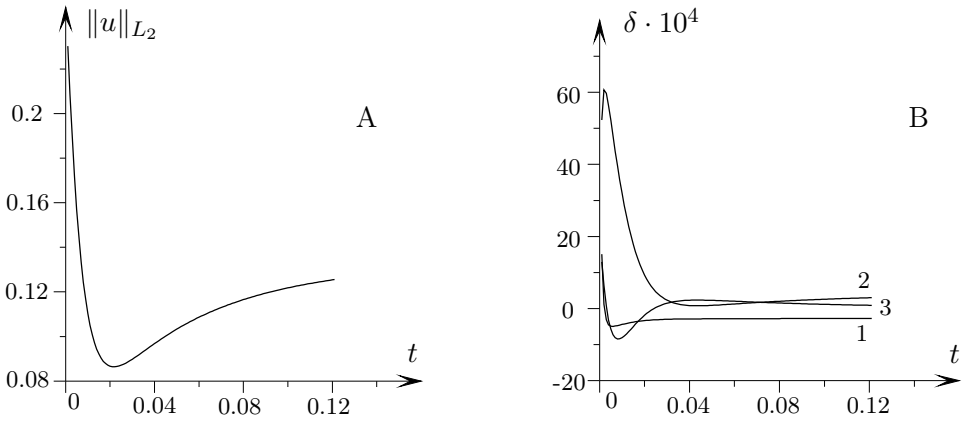

Рис. 4.3. Контроль погрешности

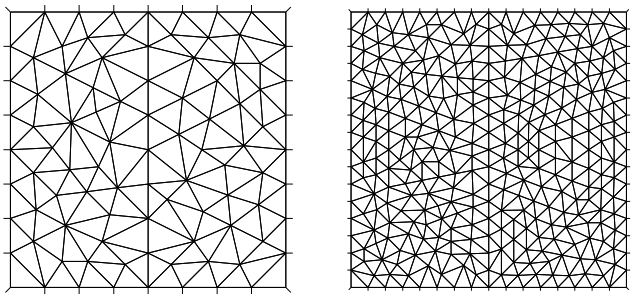

Рис. 4.4. Триангуляции `Th2` и `Th1`

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.