Министерство образования и науки Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

Курсовая работа
по дисциплине «Уравнения математической физики»

Решение начально–краевых задач для уравнений гиперболического типа в двумерных областях с помощью связки МКЭ и МКР

Факультет	ПМИ
Группа	ПМ 33
Студент	Жиляков А.
Преподаватель	Персова М. Г.
Вариант	7

Новосибирск, 2016

Содержание

Список иллюстраций

Список таблиц

Исходные тексты программы доступны в репозитории:
https://github.com/CATSPDEs

Комплекс программ для численного решения уравнений в частных производных разработан с использованием `C++11` (`Visual Studio 2015`, компилятор `MSVC`).

Для аналитических вычислений (расчёта локальных матриц и векторов, производных временного базиса и т. д.), оформления таблиц, рисунков и анимаций полученных решений использовалась мощная система символьной арифметики `Mathematica 10.4`.

I Стационарная задача

Пусть дано уравнение[1]

$$-\nabla\cdot(a\,\nabla u)+c\,u=f,\quad \mathbf{x}\in\Omega, \tag{1}$$

с краевыми условиями

$$-\hat{\mathbf{n}}\cdot(a\,\nabla u)=\kappa\,(u-g_D)-g_N,\quad \mathbf{x}\in\partial\Omega, \tag{2}$$

где $\mathbf{x}\coloneqq(x,\,y)\in\mathbb{R}^2$ — аргумент, a, c, f, κ, g_D и g_N — заданные $\mathbb{R}^2\to\mathbb{R}$ функции, $\Omega\subset\mathbb{R}^2$ — отрытая область с кусочно–гладкой границей $\partial\Omega$, $\hat{\mathbf{n}}:\mathbb{R}^2\to\mathbb{R}^2$ — нормированный вектор, нормальный к $\partial\Omega$ и направленный вне области.

Нужно найти решение $u=u(\mathbf{x})$, удовлетворяющее (1 – 2).

I.1 Вариационная постановка

Аналитическое решение задачи (I) на практике доступно редко — оно может быть найдено для «ограниченного» набора входных в (1 – 2) функций и «простых» областей. Подробно аналитические методы решения уравнений в частных производных рассмотрены в [1].

Мы рассмотрим методы численного решения задачи (I) (в этой секции — методом конечных элементов (МКЭ), в следующей — связкой МКЭ и метода конечных разностей (МКР) для нестационарной задачи).

Предположим[2], что решение задачи (I) живёт в гильбертовом пространстве

$$\mathbb{V}\coloneqq\{v=v(\mathbf{x}):\|v\|_{L_2(\Omega)}+\|\nabla v\|_{L_2(\Omega)}<\infty\}. \tag{3}$$

Домножим (в смысле скалярного произведения в L_2) уравнение (1) на тестовую функ-

[1] Уравнение вида $u_t-\nabla\cdot(a\,\nabla u)+r(u)=0$ называют **уравнением диффузии–реакции** (*diffusion–reaction eqn*). Слагаемое с оператором Лапласа «отвечает» за диффузию (или теплопроводность), зависящее от решения u слагаемое — за реакцию среды.

Если уравнение содержит конвективное слагаемое $q(\nabla u)$, то уравнение называют **уравнением конвекции–диффузии** (*convection–diffusion eqn*).

Если решение не зависит от времени ($u_t\equiv 0$) — т. е. мы имеем дело со стационарной проблемой — и $r(u)=c\,u-f$, то задача сводится к **стационарному уравнению диффузии–реакции** (*steady–state diffusion–reaction eqn*) (1).

По этим причинами мы называем входящие в (1) функции a и c коэффициентами диффузии и реакции соответственно.

[2] Зачастую функция u интерпретируется как температура плоской области Ω. Известно, что температура пропорциональна скорости движения частиц в точке $\mathbf{x}$: $u(\mathbf{x})\propto V(\mathbf{x})$, $u^2(\mathbf{x})\propto V^2(\mathbf{x})$. Принимая во внимание конечность кинетической энергии, можно утверждать, что

$$\infty>\int_\Omega u^2(\mathbf{x})\,\mathrm{d}\mathbf{x}=:\|u\|^2_{L_2(\Omega)}.$$

Стало быть, пространство Лебега — подходящее место жительства для искомого решения.

цию $v \in \mathbb{V}$:

$$\int_\Omega f\,v\,\mathrm{d}\mathbf{x} = \int_\Omega -\nabla\cdot(a\,\nabla u)\,v\,\mathrm{d}\mathbf{x} + \int_\Omega c\,u\,v\,\mathrm{d}\mathbf{x} \tag{4a}$$

$$= \int_\Omega a\,\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x} - \int_{\partial\Omega}\hat{\mathbf{n}}\cdot(a\,\nabla u)\,v\,\mathrm{d}s + \int_\Omega c\,u\,v\,\mathrm{d}\mathbf{x} \tag{4b}$$

$$= \int_\Omega a\,\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}(\kappa\,(u-g_D)-g_N)\,v\,\mathrm{d}s + \int_\Omega c\,u\,v\,\mathrm{d}\mathbf{x}. \tag{4c}$$

В (4b) мы воспользовались теоремой Грина интегрирования по частям, в (4c) — краевыми условиями (2).

Перепишем уравнение (4) так, чтобы все слагаемые, зависящие от u, остались слева:

$$\boldsymbol{a}(u,v) \coloneqq \int_\Omega a\,\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x} + \int_\Omega c\,u\,v\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}\kappa\,u\,v\,\mathrm{d}s = \int_\Omega f\,v\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}(\kappa\,g_D + g_N)\,v\,\mathrm{d}s =: \boldsymbol{\ell}(v). \tag{5}$$

Здесь $\boldsymbol{a}$ суть билинейная форма и $\boldsymbol{\ell}$ — линейный функционал, определённые на $\mathbb{V}$.

Потребуем, чтобы равенство (5) выполнялось для всех тестовых функций $v \in \mathbb{V}$. Такая задача называется **слабой формой** (или **вариационной постановкой**, или **задачей Галёркина**) для исходной задачи (I):

> Найти пробную функцию $u \in \mathbb{V}$, такую что равенство
> $\boldsymbol{a}(u,v) = \boldsymbol{\ell}(v)$
> справедливо для всех тестовых функций $v \in \mathbb{V}$. (6)

Очевидно, что если u есть решение задачи (I), то оно удовлетворяет задаче (6). Обратное не так очевидно. Состоятельность — существование и единственность решения — задачи (6) гарантируется при наложении некоторых ограничений на $\boldsymbol{a}$ и $\boldsymbol{\ell}$ (теорема Лакса—Мильграма).

Доказательства состоятельности слабых форм некоторых задач типа (I) (задача Дирихле для уравнения Пуассона, задача Неймана для уравнения диффузии–реакции и т.д.) можно найти в [2, с. 192]. О задачах Галёркина и Ритца можно прочесть в [3].

В данной работе мы не будем останавливаться на аналитике и перейдём к тому, как грамотно запрограммировать и решить задачу (6) на компьютере.

I.2 Переход к конечномерному подпространству

Пусть $\mathbb{V}_h \coloneqq \operatorname{span}\{\phi_1, \phi_2, \ldots, \phi_n\} \subset \mathbb{V}$ — некоторое известное конечномерное подпространство, $n \coloneqq \dim \mathbb{V}_h$. Будем искать приближение $u_h(\mathbf{x}) = \sum_1^n \xi_i\,\phi_i(x)$ к решению u задачи (6) в нём:

> Найти пробную функцию $u_h \in \mathbb{V}_h$, такую что равенство
> $\boldsymbol{a}(u_h,v) = \boldsymbol{\ell}(v)$
> справедливо для всех тестовых функций $v \in \mathbb{V}_h$. (7)

Очевидно, что определение функции u_h эквивалентно нахождению её n базисных коэффициентов (весов) ξ_i. Также очевидно, что требование «… справедливо для всех тестовых функций $v \in \mathbb{V}_h$» в (7) эквивалентно требованию «… справедливо для ϕ_1, ϕ_2, …, ϕ_n». Тогда задача (7) эквивалентна решению СЛАУ

$$\underbrace{(\mathbf{M}+\mathbf{S}+\mathbf{R})}_{=:\mathbf{A}}\,\boldsymbol{\xi} = \overbrace{\mathbf{f}+\mathbf{r}}^{=:\mathbf{b}}, \tag{8}$$

где $\mathbf{A}$ суть симметричная матрица системы, $\mathbf{b}$ — вектор правой части, $\boldsymbol{\xi} \coloneqq (\xi_1, \xi_2, \ldots, \xi_n)^T$ — вектор неизвестных и

$$\mathbf{M}_{ij} \coloneqq \int_\Omega c\,\phi_i\,\phi_j\,\mathrm{d}\mathbf{x}, \tag{9a}$$

$$\mathbf{S}_{ij} \coloneqq \int_\Omega a\,\nabla\phi_i \cdot \nabla\phi_j\,\mathrm{d}\mathbf{x}, \tag{9b}$$

$$\mathbf{R}_{ij} \coloneqq \int_{\partial\Omega} \kappa\,\phi_i\,\phi_j\,\mathrm{d}s, \tag{9c}$$

$$\mathbf{f}_i \coloneqq \int_\Omega f\,\phi_i\,\mathrm{d}\mathbf{x}, \tag{9d}$$

$$\mathbf{r}_i \coloneqq \int_{\partial\Omega} (\kappa\,g_D + g_N)\,\phi_i\,\mathrm{d}\mathbf{x}. \tag{9e}$$

По историческим причинам матрицу $\mathbf{M}$ называют **матрицей масс** (*mass matrix*), матрицу $\mathbf{S}$ — **матрицей жёсткости** (*stiffness matrix*) и вектор $\mathbf{f}$ — **вектором нагрузки** (*load vector*).

Матрица $\mathbf{R}$ и вектор $\mathbf{r}$ не имеют устоявшихся названий, но мы здесь будем обозначать их **матрицей и вектором Робина** соответственно. Причиной этому служит тот факт, что в них дают вклады интегралы по границе, интегранды которых определяются краевым условием (2), которое часто называют условием Робина (или обобщённым условием Неймана, или просто краевым условием 3–го типа).

Таким образом, решение задачи (7) сводится к

1. выбору базиса подпространства $\mathbb{V}_h$,
2. сборке (в МКЭ принято слово *ассемблирование*) матрицы $\mathbf{A}$ и вектора $\mathbf{b}$ системы (8)
3. и её решению.

I.3 Выбор конкретного конечномерного подпространства и дискретизация расчётной области

Наиболее распространённые дискретные области, которые используются в МКЭ для двумерных задач, — это области, разбитые на конечное множество треугольников (триангуляции) или прямоугольников.

Триангуляции — наиболее гибкий инструмент в том смысле, во–первых, что с помощью них можно очень эффективно приближать криволинейные границы, а значит, решать задачи на практически любых расчётных областях. Во–вторых, существует множество уже готовых и эффективных алгоритмов построения «качественных»[3] сеток (алгоритмы Делоне).
В этой работе мы будем рассматривать только треугольные сетки.

Пусть Ω_h есть триангуляция области Ω — множество вершин и элементов (треугольников). При этом каждый треугольник даёт в пересечении с другим треугольником либо пустое множество, либо вершину, либо ребро — иными словами, триангуляция не может иметь «висячих» вершин. Очевидным требованием является сходимость меры $\Omega_{h/k}$ к мере Ω при $k \to \infty$.
Для триангуляций, вообще говоря, придумано целое множество абстракций для представления в компьютере. Более подробно мы обсудим этот вопрос позже, когда станут ясны все требования к нашей сетке.

[3] Здесь под качеством понимается «мера близости» треугольников к правильным. Оказывается, наличие «плоских» треугольников (их называют *вырожденными*) оказывает влияние на качество МКЭ–решения. Но об этом позже.

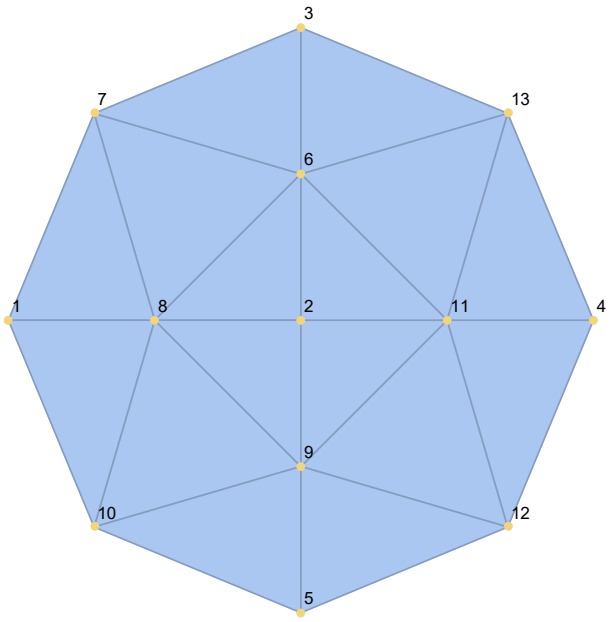

Рис. 1: Типичная триангуляция диска

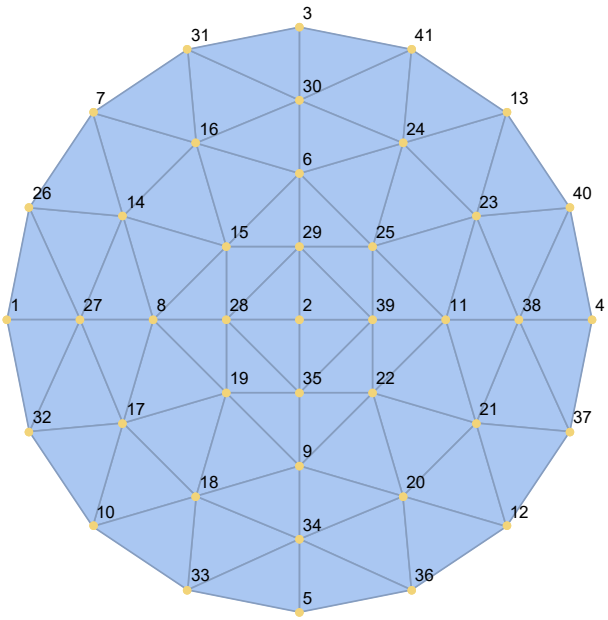

Рис. 2: Измельчённая триангуляция диска. Обратите внимание, что кривизна границы с измельчением учитывается

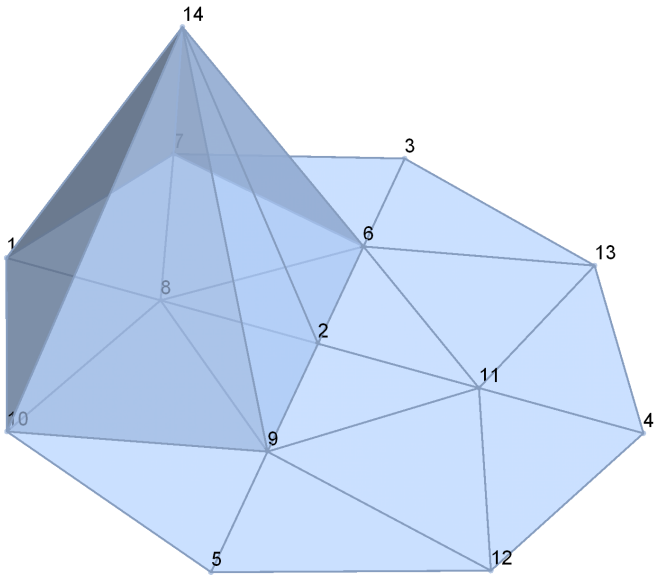

Рис. 3: Базисная функция ϕ_8, определённая на сетке рис. 1

Возьмём в качестве пространства $\mathbb{V}_h$ пространство всех непрерывных и линейных на каждом

треугольнике функций, определённых на Ω_h:

$$\mathbb{V}_h = C^0(\Omega) \cap \{v : v|_\Delta \in P_1(\Delta) \text{ для всех треугольников } \Delta \in \Omega_h\}, \tag{10}$$

$$P_1(\Delta) \coloneqq \{v : v(x, y) = c_1\, x + c_2\, y + c_3,\ (x, y) \in \Delta\} \tag{11}$$

В качестве базиса $\mathbb{V}_h$ выберем *вершинный* базис (см. рис. 3):

$$\phi_i(\mathbf{x}_j) = \delta_{i\,j} \text{ для всех вершин } \mathbf{x_j} \text{ триангуляции } \Omega_h, \tag{12}$$

т. е. i–я функция базиса принимает единицу в i–й вершине и ноль иначе.

В таком выборе базиса есть, как минимум, четыре преимущества:

1. простота интерпретации базисных коэффициентов — $u_h(\mathbf{x}_i) = \xi_i$,
2. (вытекает из первого пункта) простота построения интерполяции $\pi_1 g(\mathbf{x}) \coloneqq \sum_1^n g(\mathbf{x}_i)\, \phi_i(\mathbf{x})$ любой известной функции g. Это полезно для расчёта квадратур, входящих в систему (8),
3. финитность базисных функций и, как следствие, **разреженность матрицы A** и
4. элегантность алгоритма ассемблирования системы (8) (подробно рассмотрим в разделе (I.5)).

Третий пункт особенно важен, так как он позволяет решать задачи, приводящие к огромным СЛАУ, пользуясь ограниченной памятью компьютера.

I.4 Учёт краевых условий

Задание граничных условий (2) в виде тройки функций $(\kappa,\, g_D,\, g_N)$ удобно тем, что оно единообразно описывает все три классических типа краевых условий: Дирихле, Неймана и Робина.

Пусть Γ_D, Γ_N и $\Gamma_R \subset \partial\Omega$ суть не пересекающиеся части границы, в объединении дающие $\partial\Omega$, на которых заданы краевые условия Дирихле, Неймана и Робина соответственно. Задания условий на поток через границу очевидны; для условий Неймана достаточно положить $\kappa(\Gamma_N) = \{0\}$, для условий Робина — $g_D(\Gamma_R) = \{0\}$:

$$\begin{aligned} \hat{\mathbf{n}} \cdot (a\, \nabla u) &= g_N, \quad \mathbf{x} \in \Gamma_N, \\ \hat{\mathbf{n}} \cdot (a\, \nabla u) + \kappa\, u &= g_N, \quad \mathbf{x} \in \Gamma_R. \end{aligned}$$

Краевые условия Дирихле $u = g_D$, $\mathbf{x} \in \Gamma_D$ нельзя учесть явно с использованием (2), однако их можно аппроксимировать условиями Робина. Перепишем (2) в виде

$$u - g_D = \frac{g_N - \hat{\mathbf{n}} \cdot (a\, \nabla u)}{\kappa}.$$

Положив $g_N(\Gamma_D) = \{0\}$ и $\kappa(\Gamma_D)$ достаточно большим (скажем, порядка 10^{50}), получим $u - g_D \simeq 0$ — фактически равенство в конечной арифметике компьютера.

С практической точки зрения такой учёт условий Дирихле означает внесение возмущений в $\mathbf{A}$ и $\mathbf{b}$ — взгляните на элементы матрицы и вектора Робина (9c) и (9e).
В них дают вклад интегралы, в интегранды которых входит $\kappa = 10^{50}$.

Существует два распространённых способа учёта условий Дирихле:

1. для всех $\mathbf{x}_j \in \Gamma_D$ обнулить j–ю строчку матрицы $\mathbf{A}$, установить $\mathbf{A}_{j\,j} = 1$ и $\mathbf{b}_j = g_D(\mathbf{x}_j)$,
2. для всех $\mathbf{x}_j \in \Gamma_D$ положить $\mathbf{b}_j = g_D(\mathbf{x}_j)$ и домножить $\mathbf{b}_j$ и $\mathbf{A}_{j\,j}$ на большое число (скажем, на 10^{50}).

Первый подход фактически заменяет j–е уравнение системы (8) на уравнение $\xi_j = g_D(\mathbf{x}_j)$, второй — на уравнение $\xi_j \simeq g_D(\mathbf{x}_j)$, нивелируя вклад остальных слагаемых уравнения[4].

По своей сути наш подход совпадает со вторым, однако у него есть приятный бонус: нет никакой необходимости заботиться об учёте условий Дирихле отдельно, изменяя матрицу и вектор системы — *учёт условий всех типов происходит единообразно.*

Первый подход наиболее «честный» в том смысле, что решение ищется в «правильном» пространстве $\mathbb{V}_{h,D} := \mathbb{V}_h \cap \{v : v(\mathbf{x}) = g_D(\mathbf{x}) \text{ для всех } \mathbf{x} \in \Gamma_D\}$. Однако такой подход нарушает симметричность матрицы $\mathbf{A}$. Как следствие, приходится либо симметризовывать матрицу, либо использовать несимметричный формат хранения, увеличивая расходы на память компьютера.

В данной работе мы не будем использовать первый подход.

I.5 Ассемблирование СЛАУ

I.5.1 Вклады квадратур по элементам

В сердце МКЭ — ассемблирование матрицы $\mathbf{A}$ и вектора $\mathbf{b}$, которое очень удобно и быстро осуществлять при обходе **элементов** (в нашем случае — треугольников) сетки Ω_h. Отсюда первое требование к абстракции для сетки — **элементы необходимо хранить явно**.

Рассмотрим процесс сборки на примере Ω_h, представленной на рис. 4.

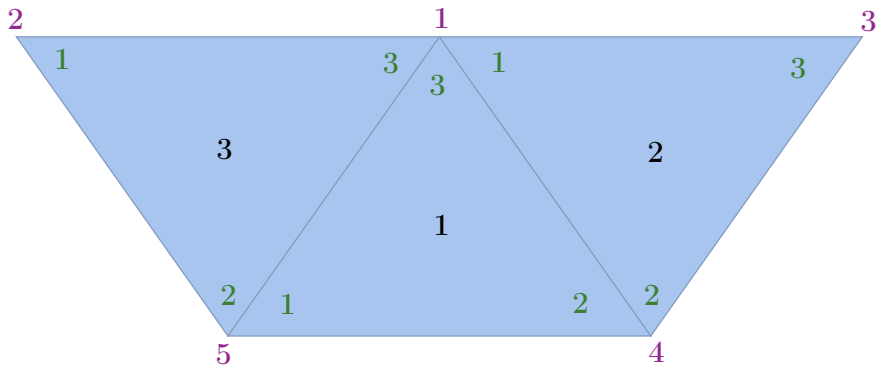

Рис. 4: Глобальная нумерация узлов, локальная нумерация узлов и глобальная нумерация элементов, задаваемые абстракциями $\mathcal{V}$ и $\mathcal{T}$

Триангуляция[5] $\Omega_h = \cup_1^3 \Delta^{(i)}$ (рис. 4) может быть представлена списком вершин $\mathcal{V}$ и «треугольников» (указатели на элементы $\mathcal{V}$) $\mathcal{T}$:

$$\mathcal{V} := \langle \quad (\frac{1}{2}, \frac{\sqrt{2}}{2}), (-\frac{1}{2}, \frac{\sqrt{2}}{2}), (\frac{3}{2}, \frac{\sqrt{2}}{2}), (1, 0), (0, 0) \quad \rangle,$$
$$\mathcal{T} := \langle \quad (5, 4, 1), (1, 4, 3), (2, 5, 1) \quad \rangle.$$

Заметим, что каждый элемент $\mathcal{T}_i$ определяет отображение $\boldsymbol{n}^{(i)} : \{1, 2, 3\} \to \{\mathcal{T}_{i1}, \mathcal{T}_{i2}, \mathcal{T}_{i3}\}$ локальной нумерации вершин на глобальную.
Например, для первого треугольника $\Delta^{(1)}$ соответствие между локальной и глобальную нумерацией определяется $\boldsymbol{n}^{(1)}(1) = 5$, $\boldsymbol{n}^{(1)}(2) = 4$ и $\boldsymbol{n}^{(1)}(3) = 1$ (см. рис. 4).

Вершины $\mathbf{x}_1^{(i)} := (x_1^{(i)}, y_1^{(i)})$, $\mathbf{x}_2^{(i)} := (x_2^{(i)}, y_2^{(i)})$ и $\mathbf{x}_3^{(i)} := (x_3^{(i)}, y_3^{(i)})$, образующие треугольник $\Delta^{(i)}$, могут быть получены так:

$$\mathbf{x}_j^{(i)} = \mathcal{V}_{n_i(j)}.$$

[4] В МКЭ в качестве базиса выбирают *вершинный базис*, т. е. такой, что $\phi_i(\mathbf{x}_j) = \delta_{ij}$ (символ Кронекера). Поэтому вес ξ_j суть значение функции u_h в j–м узле.

[5] Здесь и далее для упрощения нотаций мы будем использовать верхний индекс со скобками (i) для указания того, что объект имеет отношение к i–му элементу сетки $\Delta^{(i)}$.

Продемонстрируем ассемблирование на примере матрицы масс:

$$\mathbf{M} = \int_{\Omega_h} c \begin{bmatrix} \phi_1\,\phi_1 & \phi_1\,\phi_2 & \phi_1\,\phi_3 & \phi_1\,\phi_4 & \phi_1\,\phi_5 \\ & \phi_2\,\phi_2 & \phi_2\,\phi_3 & \phi_2\,\phi_4 & \phi_2\,\phi_5 \\ & & \phi_3\,\phi_3 & \phi_3\,\phi_4 & \phi_3\,\phi_5 \\ & & & \phi_4\,\phi_4 & \phi_4\,\phi_5 \\ \text{сим.} & & & & \phi_5\,\phi_5 \end{bmatrix} \mathrm{d}\mathbf{x} =$$

$$\int_{\Delta_1} c \begin{bmatrix} \phi_1\,\phi_1 & 0 & 0 & \phi_1\,\phi_4 & \phi_1\,\phi_5 \\ & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 0 \\ & & & \phi_4\,\phi_4 & \phi_4\,\phi_5 \\ \text{сим.} & & & & \phi_5\,\phi_5 \end{bmatrix} \mathrm{d}\mathbf{x} + \int_{\Delta_2} c \begin{bmatrix} \phi_1\,\phi_1 & 0 & \phi_1\,\phi_3 & \phi_1\,\phi_4 & 0 \\ & 0 & 0 & 0 & 0 \\ & & \phi_3\,\phi_3 & \phi_3\,\phi_4 & 0 \\ & & & \phi_4\,\phi_4 & 0 \\ \text{сим.} & & & & 0 \end{bmatrix} \mathrm{d}\mathbf{x} + \tag{13}$$

$$\int_{\Delta_3} c \begin{bmatrix} \phi_1\,\phi_1 & \phi_1\,\phi_2 & 0 & 0 & \phi_1\,\phi_5 \\ & \phi_2\,\phi_2 & 0 & 0 & \phi_2\,\phi_5 \\ & & 0 & 0 & 0 \\ & & & 0 & 0 \\ \text{сим.} & & & & \phi_5\,\phi_5 \end{bmatrix} \mathrm{d}\mathbf{x}$$

В (13) мы воспользовались аддитивностью интеграла и финитностью базисных функций — из построения базиса очевидно, что вклад в матрицу на треугольнике $\Delta^{(i)}$ дадут лишь три базисные функции, принимающие единицу в образующих его узлах (см. рис. 5).

На практике на каждом элементе $\Delta^{(i)}$ считается **локальная матрица** размерности 3×3

$$\hat{\mathbf{M}}^{(i)} = \int_{\Delta^{(i)}} c \begin{bmatrix} \hat\phi_1^{(i)}\,\hat\phi_1^{(i)} & \hat\phi_1^{(i)}\,\hat\phi_2^{(i)} & \hat\phi_1^{(i)}\,\hat\phi_3^{(i)} \\ & \hat\phi_2^{(i)}\,\hat\phi_2^{(i)} & \hat\phi_2^{(i)}\,\hat\phi_3^{(i)} \\ \text{сим.} & & \hat\phi_3^{(i)}\,\hat\phi_3^{(i)} \end{bmatrix} \mathrm{d}\mathbf{x}, \tag{14}$$

где $\hat\phi_j^{(i)} \coloneqq \phi_{\boldsymbol{n}^{(i)}(j)}|_{\Delta^{(i)}}$ — **локальная базисная функция** (см. рис. 5); затем вносится вклад в матрицу системы $\mathbf{A}_{\boldsymbol{n}^{(i)}(j)\,\boldsymbol{n}^{(i)}(k)} = \mathbf{A}_{\boldsymbol{n}^{(i)}(j)\,\boldsymbol{n}^{(i)}(k)} + \hat{\mathbf{M}}_{j\,k}$, $\quad j = 1,\,2,\,3, \quad k = j,\,\ldots,\,3$ и осуществляется переход к следующему элементу Δ_{i+1}.

Формулы для локальных базисных функций легко получить из определения (12), записанного в матричном виде. $\hat\phi_j^{(i)}(x,\,y) = c_{j\,1}^{(i)}\,x + c_{j\,2}^{(i)}\,y + c_{j\,3}^{(i)}$ и

$$\underbrace{\begin{bmatrix} c_{1\,1}^{(i)} & c_{1\,2}^{(i)} & c_{1\,3}^{(i)} \\ c_{2\,1}^{(i)} & c_{2\,2}^{(i)} & c_{2\,3}^{(i)} \\ c_{3\,1}^{(i)} & c_{3\,2}^{(i)} & c_{3\,3}^{(i)} \end{bmatrix}}_{=:\mathbf{C}^{(i)}} \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & x_3^{(i)} \\ y_1^{(i)} & y_2^{(i)} & y_3^{(i)} \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{15}$$

поэтому $\hat\phi_j^{(i)} = (\mathbf{C}^{(i)})_{\cdot\,j}^{-1} \cdot (x,\,y,\,1)$.

Коэффициент реакции $c(\mathbf{x})$ может быть, вообще говоря, произвольной функцией. Поэтому для расчёта квадратур в (14) удобно заменить его линейной (или квадратичной) интерполяцией

$$c|_{\Delta^{(i)}} \simeq \pi_1 c^{(i)}(\mathbf{x}) \coloneqq \sum_{j=1}^{3} c(\mathbf{x}_j^{(i)})\,\hat\phi_j^{(i)}(\mathbf{x}) \text{ или} \tag{16a}$$

$$\pi_2 c^{(i)}(\mathbf{x}) \coloneqq \sum_{j=1}^{6} c(\mathbf{x}_j^{(i)})\,\hat\psi_j^{(i)}(\mathbf{x}) \tag{16b}$$

где $\hat\psi_1^{(i)}$, $\hat\psi_2^{(i)}$, $\ldots$, $\hat\psi_6^{(i)}$ — квадратичные базисные функции (см. рис. 6), принимающие единицы на вершинах треугольника $\Delta^{(i)}$ и серединах его рёбер

$$\mathbf{x}_4^{(i)} \coloneqq \frac{\mathbf{x}_2^{(i)} + \mathbf{x}_3^{(i)}}{2}, \quad \mathbf{x}_5^{(i)} \coloneqq \frac{\mathbf{x}_1^{(i)} + \mathbf{x}_3^{(i)}}{2}, \quad \mathbf{x}_6^{(i)} \coloneqq \frac{\mathbf{x}_1^{(i)} + \mathbf{x}_2^{(i)}}{2}. \tag{17}$$

Формулы для квадратичного базиса могут быть получены аналогично (15). Более того, можно заметить, что локальные квадратичные базисные функции можно выразить через произведения локальных линейных базисных функций.

Очевидно, что

$$c|_{\Delta^{(i)}} = \pi_1 c^{(i)}(\mathbf{x}), \text{ если } c|_{\Delta^{(i)}} \in P_1(\Delta^{(i)}), \tag{18}$$

$$c|_{\Delta^{(i)}} = \pi_2 c^{(i)}(\mathbf{x}), \text{ если } c|_{\Delta^{(i)}} \in P_2(\Delta^{(i)}), \tag{19}$$

где $P_2(\Delta^{(i)})$ (аналогично $P_1(\Delta^{(i)})$, определённое в (11)) есть пространство полиномов 2-й степени, определённых на i–м элементе:

$$P_2(\Delta^{(i)}) \coloneqq \{v : v(x,\, y) = c_1\, x^2 + c_2\, y^2 + c_3\, x\, y + c_4\, x + c_5\, y + c_6,\ (x,\, y) \in \Delta^{(i)}\}, \tag{20}$$

$$P_1(\Delta^{(i)}) \subset P_2(\Delta^{(i)}). \tag{21}$$

Если же $c(\mathbf{x})$ не является полиномом, то с дроблением сетки интерполяция (16) будет всё лучше его аппроксимировать.

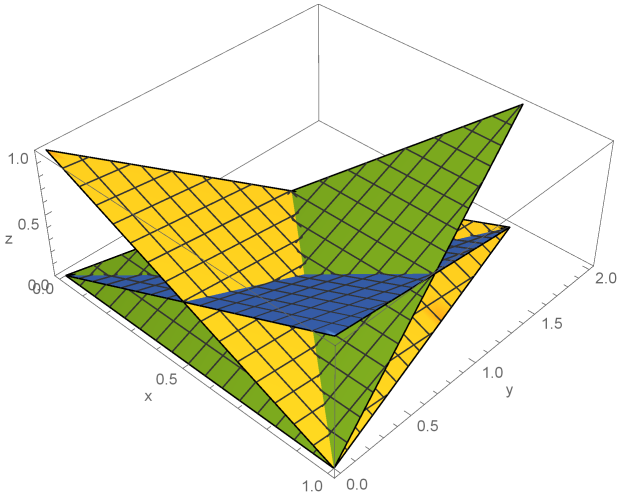

Рис. 5: Локальные базисные функции $\hat{\phi}_1^{(i)}$, $\hat{\phi}_2^{(i)}$ и $\hat{\phi}_3^{(i)}$, линейные на i-м элементе

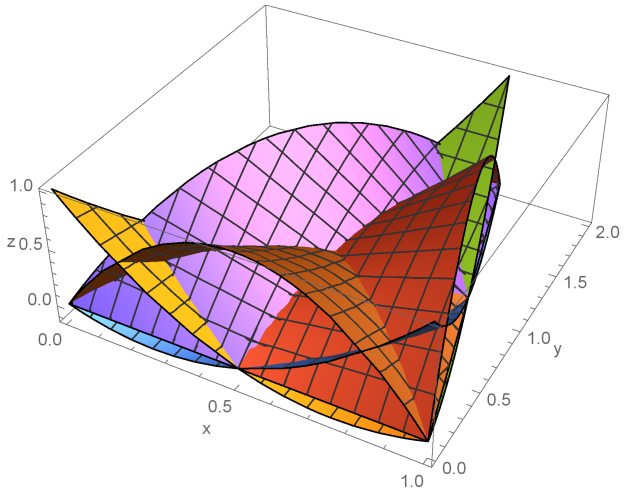

Рис. 6: Локальные базисные функции $\hat{\psi}_1^{(i)}$, $\hat{\psi}_2^{(i)}, \ldots, \hat{\psi}_6^{(i)}$, квадратичные на i-м элементе

С учётом интерполяции (16) и сказанного выше, вычисление квадратур для (14) сводится к вычислению интегралов типа

$$\int_{\Delta^{(i)}} (\hat{\phi}_1^{(i)}(\mathbf{x}))^m\, (\hat{\phi}_2^{(i)}(\mathbf{x}))^l\, (\hat{\phi}_3^{(i)}(\mathbf{x}))^k\, \mathrm{d}\mathbf{x}, \tag{22}$$

вид которых известен (или может быть быстро посчитан, например, в системе `Mathematica`).

Сборка матрицы $\mathbf{S}$ и вектора $\mathbf{f}$ абсолютно аналогична, поэтому выкладки мы приводить не будем.

В секции (I.5.3) приведены формулы для расчёта всех локальных матриц и векторов (9), необходимых для сборки системы.

I.5.2 Вклады квадратур по границе

Для того, чтобы ассемблировать вклады матрицы и вектора Робина, необходимо иметь доступ к границе Ω_h. **Это второе требование к абстракции для нашей триангуляции**.

В принципе, можно было бы хранить список граничных «рёбер» — список пар указателей на вершины в списке $\mathcal{V}$ (подобно тому, как мы поступили с абстракцией для элементов $\mathcal{T}$). Однако мы откажемся от такого подхода по причинам, изложенным ниже.

I.5.2.a Измельчение сетки и связанные с этим требования к абстракции для триангуляции

Для уточнения решения часто необходимо измельчить сетку Ω_h — решить задачу на более точной триангуляции $\Omega_{\frac{h}{2}}$. При этом дробиться могут не обязательно все[6] элементы $\Delta^{(i)}$.

Поэтому третье требование к абстракции — возможность измельчать исходную сетку Ω_h.

Добавление элементов (а значит, и новых узлов) приводит к появлению «висячих» вершин (мы обсудили это в секции I.3), нарушающих структуру триангуляции.

Чтобы избежать этой проблемы, удобно иметь доступ к $0 \leq k \leq 3$ смежным треугольникам («соседям») $\Delta^{(j)}$ текущего треугольника $\Delta^{(i)}$.
Для этого введём новую абстракцию $\boldsymbol{\mathcal{N}}$, элемент $\boldsymbol{\mathcal{N}}_i$ которой содержит тройку указателей на элементы $\boldsymbol{\mathcal{T}}$, являющиеся соседями элемента $\Delta^{(i)}$, описываемого i–м элементом абстракции $\boldsymbol{\mathcal{T}}$.
Для сетки рис. 4 описанная структура будет иметь вид:

$$\boldsymbol{\mathcal{V}} \coloneqq \langle \quad (\frac{1}{2}, \frac{\sqrt{2}}{2}), (-\frac{1}{2}, \frac{\sqrt{2}}{2}), (\frac{3}{2}, \frac{\sqrt{2}}{2}), (1,\, 0), (0,\, 0) \quad \rangle, \tag{23a}$$

$$\boldsymbol{\mathcal{T}} \coloneqq \langle \quad (5,\, 4,\, 1), (1,\, 4,\, 3), (2,\, 5,\, 1) \quad \rangle, \tag{23b}$$

$$\boldsymbol{\mathcal{N}} \coloneqq \langle \quad (2,\, 3,\, -1), (-1,\, -1,\, 1), (1,\, -1,\, -1) \quad \rangle. \tag{23c}$$

Обратите внимание, что если ребро треугольника $\boldsymbol{\mathcal{T}}_i$ напротив k–го узла $\boldsymbol{\mathcal{V}}_{\boldsymbol{n}_i(k)}$ является частью границы (т. е. у треугольника нет соседа по этому ребру), то $\boldsymbol{\mathcal{N}}_{i\,k} = -1$.
Например, как видно из рис. 4 и (23), 3–е ребро (ребро напротив 3–й вершины) 1–го треугольника является частью границы. Поэтому $\boldsymbol{\mathcal{N}}_{1\,3} = -1$.

Условимся, что узлы в тройках $\boldsymbol{\mathcal{T}}_i$ занумерованы так, что обход вершин i-го треугольника ведётся **против часовой стрелки** (справедливо для нашего примера (23)). Это удобно, потому что интегралы по границе в (4) берутся при её обходе против часовой стрелки.
Таким образом, не будет путаницы при учёте знака в процессе сборки вкладов матрицы (9c) и вектора Робина (9e) в СЛАУ (8).

Структура $\langle \boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{N}} \rangle$ называется «**узлы и треугольники**» (см. [4, с. 14]).

При использовании данной структуры все перечисленные выше требования к абстракции для триангуляции выполняются. Как было замечено в начале раздела I.5.2, мы освобождены от необходимости явного хранения границы — мы можем легко получить её при обходе элементов $\Delta^{(i)}$: если $\boldsymbol{\mathcal{N}}_{i\,k} < 0$, то ребро $(\boldsymbol{\mathcal{V}}_{\boldsymbol{n}_i(k+1)}, \boldsymbol{\mathcal{V}}_{\boldsymbol{n}_i(k+2)})$ суть часть $\partial\Omega$. Здесь через «$\dot{+}$» обозначено сложение по модулю 4: $2 \dot{+} 1 = 3$, $2 \dot{+} 2 = 1$ и т. д.

Вывод формул для локальных матрицы и вектора Робина мы не будем приводить здесь, потому что он аналогичен выводу локальной матрицы масс $\hat{\mathbf{M}}^{(i)}$, приведённому в разделе I.5.1. Как видно из рис. 5, лишь две локальные базисные функции дают вклад в интеграл по ребру, поэтому размерности $\hat{\mathbf{R}}^{(i)}$ и $\hat{\mathbf{r}}^{(i)}$ — 2×2 и 2 соответственно.

В секции (I.5.3) приведены формулы для расчёта всех локальных матриц и векторов (9), необходимых для сборки системы.

---

[6] Примером может служить **адаптивный МКЭ** (*adaptive FEM*). По окончании решения используется апостериорная оценка ошибки (основанная на скачках в градиентах полученного решения u_h), на основании которой конечное количество треугольников измельчается и решение пересчитывается.
Об адаптивном МКЭ можно прочесть в [2, с. 97].

I.5.3 Формулы для локальных матриц и векторов

Для упрощения нотаций в таблице 1 мы не стали указывать индекс элемента (Δ вместо $\Delta^{(i)}$, $\hat{\mathbf{M}}$ — вместо $\hat{\mathbf{M}}^{(i)}$ и т. д.). Через Δ, как всегда, обозначен элемент, через e — граничное ребро.

В матрицах (векторах) с интегралами по элементам через (x_1, y_1), (x_2, y_2) и (x_3, y_3) обозначены образующие элемент вершины; в матрице и векторе Робина через (x_1, y_1) и (x_2, y_2) обозначены вершины, образующие ребро.

Для краткости образ точки (x_j, y_j) мы обозначили $f_j := f(x_j, y_j)$, $\kappa_j := \kappa(x_j, y_j)$ и т. д.

Локальная матрица (вектор)			**Формула**
$\hat{\mathbf{M}}$	$c \in P_1$	$\frac{\text{area of } \Delta}{60}$	$\begin{pmatrix} 6\,c_1 + 2\,c_2 + 2\,c_3 & 2\,c_1 + 2\,c_2 + c_3 & 2\,c_1 + c_2 + 2\,c_3 \\ & 2\,c_1 + 6\,c_2 + 2\,c_3 & c_1 + 2\,c_2 + 2\,c_3 \\ \text{сим.} & & 2\,c_1 + 2\,c_2 + 6\,c_3 \end{pmatrix}$
$\hat{\mathbf{S}}$	$a \in P_1$	$\frac{a_1+a_2+a_3}{12\,\text{area of } \Delta}$	$\begin{pmatrix} (x_2-x_3)^2 + (y_2-y_3)^2 & (x_1-x_3)(x_3-x_2) + (y_1-y_3)(y_3-y_2) & (x_1-x_2)(x_2-x_3) + (y_1-y_2)(y_2-y_3) \\ & (x_1-x_3)^2 + (y_1-y_3)^2 & (x_2-x_1)(x_1-x_3) + (y_2-y_1)(y_1-y_3) \\ \text{сим.} & & (x_1-x_2)^2 + (y_1-y_2)^2 \end{pmatrix}$
$\hat{\mathbf{S}}$	$a \in P_2$	$\frac{a_3+a_4+a_5}{12\,\text{area of } \Delta}$	$\begin{pmatrix} (x_2-x_3)^2 + (y_2-y_3)^2 & (x_1-x_3)(x_3-x_2) + (y_1-y_3)(y_3-y_2) & (x_1-x_2)(x_2-x_3) + (y_1-y_2)(y_2-y_3) \\ & (x_1-x_3)^2 + (y_1-y_3)^2 & (x_2-x_1)(x_1-x_3) + (y_2-y_1)(y_1-y_3) \\ \text{сим.} & & (x_1-x_2)^2 + (y_1-y_2)^2 \end{pmatrix}$
$\hat{\mathbf{R}}$	$\kappa \in P_1$	$\frac{\text{length of } e}{12}$	$\begin{pmatrix} 3\,\kappa_1 + \kappa_2 & \kappa_1 + \kappa_2 \\ \text{сим.} & \kappa_1 + 3\,\kappa_2 \end{pmatrix}$
$\hat{\mathbf{f}}$	$f \in P_1$	$\frac{\text{area of } \Delta}{12}$	$\begin{pmatrix} 2\,f_1 + f_2 + f_3 \\ f_1 + 2\,f_2 + f_3 \\ f_1 + f_2 + 2\,f_3 \end{pmatrix}$
$\hat{\mathbf{f}}$	$f \in P_2$	$\frac{\text{area of } \Delta}{60}$	$\begin{pmatrix} 2f_1 - f_2 - f_3 + 4f_4 + 8f_5 + 8f_6 \\ f_1 - 2f_2 + f_3 - 4\,(2f_4 + f_5 + 2f_6) \\ f_1 + f_2 - 2\,(f_3 + 4\,(f_4 + f_5) + 2f_6) \end{pmatrix}$
$\hat{\mathbf{r}}$	$\kappa,\, g_D,\, g_N \in P_1$	$\frac{\text{length of } e}{12}$	$\begin{pmatrix} (\kappa_1 + \kappa_2)\,g_{D_2} + (3\kappa_1 + \kappa_2)\,g_{D_1} + 4g_{N_1} + 2g_{N_2} \\ (\kappa_1 + \kappa_2)\,g_{D_1} + (\kappa_1 + 3\kappa_2)\,g_{D_2} + 2g_{N_1} + 4g_{N_2} \end{pmatrix}$

Таблица 1: Формулы для локальных матриц (векторов), необходимые для сборки вкладов от (9) в СЛАУ (8)

I.6 Генерация портрета матрицы, абстракция для матрицы и решение системы

Мы уже упомянули тот факт, что матрица системы есть **разреженная** матрица. Портрет матрицы тесно связан с видом триангуляции: его легко определить, имея список смежных вершин к данной вершине — произведение базисных функций $\phi_i(\mathbf{x})\,\phi_j(\mathbf{x})$, как видно из их определения, даст не ноль только тогда, когда $\mathbf{x}_i$ и $\mathbf{x}_j$ связаны ребром.

Используемый формат для абстракции разреженной матрицы — **симметричный CSlR–формат** (*compressed sparse (lower triangular) row*), который иногда называют **симметричным разреженно–строчным форматом**.
О данном формате можно прочесть в [5, с. 5].

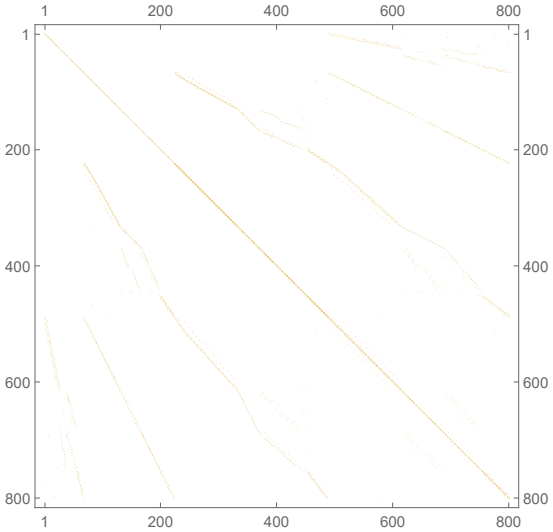

Рис. 7: Портрет матрицы для рис. (9)

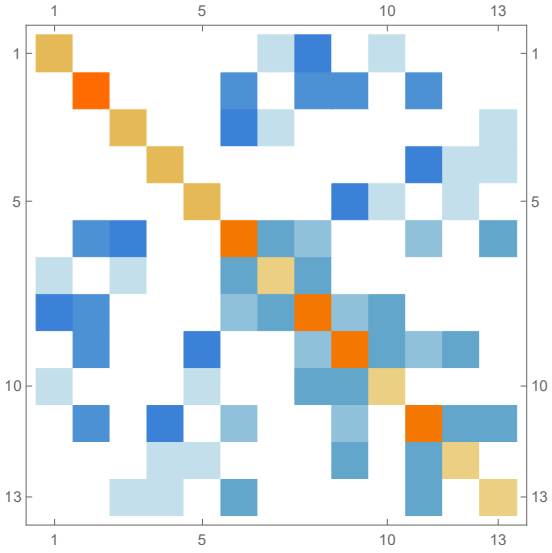

Рис. 8: Портрет матрицы для рис. (10)

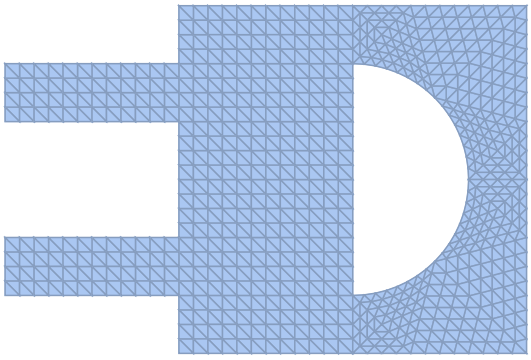

Рис. 9: Сетка с $n = 800$ для сложной области

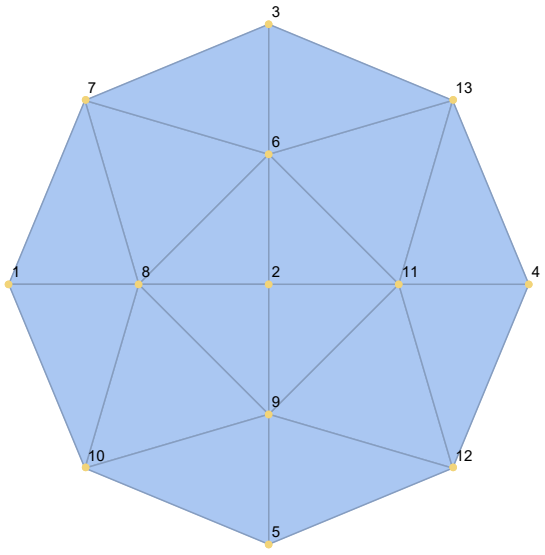

Рис. 10: Простая триангуляция круга

Можно показать, что для многих входных в I параметров матрица системы $\mathbf{A}$ будет положительно определена. Поэтому для решения СЛАУ разумно использовать **метод сопряжённых градиентов** (например, с $\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T$–предобуславливанием).

Подробнее о реализации можно прочесть в [6].

II Задача, зависящая от времени

Пусть дано **гиперболическое** уравнение

$$\chi\,\ddot{u} + \sigma\,\dot{u} - \nabla\cdot(a\,\nabla u) = f, \quad \mathbf{x}\in\Omega, \quad t\in[t_0,\,t_l]\,, \tag{24}$$

с краевыми условиями (2) и начальными условиями

$$u(\mathbf{x}, t_0) = u_0(\mathbf{x}), \tag{25a}$$
$$\dot{u}(\mathbf{x}, t_0) = v_0(\mathbf{x}). \tag{25b}$$

Отличия от (1 – 2): здесь через «˙» обозначена производная по времени t, входные параметры $\chi = \chi(\mathbf{x})$, $\sigma = \sigma(\mathbf{x})$ и $a = a(\mathbf{x})$ есть функции пространства, а $\kappa = \kappa(\mathbf{x},\,t)$, $g_D = g_D(\mathbf{x},\,t)$ и $g_N = g_N(\mathbf{x},\,t)$ — функции пространства и времени.
u_0 есть начальное положение, v_0 — начальная скорость.

Через $[t_0,\,t_l]\subset[0,\infty)$ обозначен временной сегмент, в котором нужно найти решение $u = u(\mathbf{x},\,t)$, удовлетворяющее (24 – 2 – 25).

Положив в (24) параметр $\chi\equiv 0$, получим задачу **параболического** типа. Поэтому все схемы решения, выведенные ниже, справедливы для задач и гиперболического, и параболического типа (с известными упрощениями).

II.1 Вариационная постановка

Подобно (3), определим пространство

$$\mathbb{V}_t \coloneqq \{v = v(\mathbf{x},\,t) : \|v(\cdot,t)\|_{L_2(\Omega)} + \|\nabla v(\cdot,\,t)\|_{L_2(\Omega)} < \infty\}. \tag{26}$$

Аналогично тому, как мы поступили в разделе I.1, домножим уравнение (24) на тестовую функцию $v\in\mathbb{V}_t$, воспользуемся формулой Грина интегрирования по частям и краевыми условиями (2):

$$\overbrace{\int_\Omega \chi\,\ddot{u}\,v\,\mathrm{d}\mathbf{x} + \int_\Omega \sigma\,\dot{u}\,v\,\mathrm{d}\mathbf{x} + \int_\Omega a\,\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}\kappa\,u\,v\,\mathrm{d}s}^{\boldsymbol{a}_t(u,v):=} = \underbrace{\int_\Omega f\,v\,\mathrm{d}\mathbf{x} + \int_{\partial\Omega}(\kappa\,g_D + g_N)\,v\,\mathrm{d}s}_{=:\boldsymbol{\ell}_t(v)}. \tag{27}$$

Тогда слабая формулировка задачи II, аналогичная (6), будет звучать так:

> Найти пробную функцию $u\in\mathbb{V}_t$, такую что равенство
> $\boldsymbol{a}_t(u,v) = \boldsymbol{\ell}_t(v)$
> справедливо для всех тестовых функций $v\in\mathbb{V}_t$. (28)

Будем искать приближение u_h в следующем виде:

$$u_h(\mathbf{x},\,t) = \sum_1^n \xi_i(t)\,\phi_i(x), \tag{29}$$

т. е. предположим, что в каждый фиксированный момент времени t наше решение есть житель конечномерного кусочно–линейного непрерывного на Ω_h подпространства $\mathbb{V}_h$, определённого в разделе I.2.

Подставляя представление (29) в 28 и (учитывая конечномерность пространства $\mathbb{V}_h$) заменяя тестовую функцию на базисную, получим:

$$\sum_{i=1}^{n}\left(\ddot{\xi}_i(t)\int_\Omega \chi\,\phi_i\,\phi_j\,\mathrm{d}\mathbf{x}+\dot{\xi}_i(t)\int_\Omega \sigma\,\phi_i\,\phi_j\,\mathrm{d}\mathbf{x}+\xi_i(t)\left(\int_\Omega a\,\nabla\phi_i\,\nabla\phi_j\,\mathrm{d}\mathbf{x}+\int_{\partial\Omega}\kappa\,\phi_i\,\phi_j\,\mathrm{d}s\right)\right)=$$
$$\int_\Omega f\,\phi_j\,\mathrm{d}\mathbf{x}+\int_{\partial\Omega}(\kappa\,g_D+g_N)\,\phi_j\,\mathrm{d}s\qquad\text{для всех }\phi_j,\quad j=1,\,2,\,\ldots,\,n,$$

что есть ничто иное, как система линейных дифференциальных уравнений (СЛДУ)

$$\mathbf{M}^\chi\,\ddot{\boldsymbol{\xi}}+\mathbf{M}^\sigma\,\dot{\boldsymbol{\xi}}+\big(\mathbf{S}+\mathbf{R}(t)\big)\,\boldsymbol{\xi}=\mathbf{f}(t)+\mathbf{r}(t),\tag{30}$$

где $\boldsymbol{\xi}(t)\coloneqq\big(\xi_1(t),\,\xi_2(t),\,\ldots,\,\xi_n(t)\big)^T$ — вектор неизвестных функций–весов, $\mathbf{M}^\chi$ и $\mathbf{M}^\sigma$ суть привычные матрицы масс (см. (9)) с входящими в интегранд χ и σ соответственно, $\mathbf{S}$ — матрица жёсткости, $\mathbf{R}$ — матрица Робина, $\mathbf{f}$ — вектор нагрузки и $\mathbf{r}$ — вектор Робина.

II.2 Решение СЛДУ методом Крэнка—Николсона (CN3) и неявной трёхслойной схемой (BDF3)

Выберем $l-1$ точку $t_1,\,t_2,\,\ldots,\,t_{l-1}$ так, что $t_0<t_1<\cdots<t_{l-1}<t_l$. Точки–элементы списка $T_l\coloneqq\langle\,t_0,\,t_1,\,\ldots,\,t_l\,\rangle$ назовём **временными слоями** и будем искать решение в них.

Классический способ решения СЛДУ — метод конечных разностей (МКР). Предполагая константный временной шаг $\Delta t_m\coloneqq t_m-t_{m-1}$, аппроксимируем дифференциальные операторы

$$\ddot{\boldsymbol{\xi}}(t_m)=\frac{\boldsymbol{\xi}(t_m)-2\,\boldsymbol{\xi}(t_{m-1})+\boldsymbol{\xi}(t_{m-2})}{\Delta t^2}+O(\Delta t^3),\tag{31a}$$
$$\dot{\boldsymbol{\xi}}(t_m)=\frac{\boldsymbol{\xi}(t_m)-\boldsymbol{\xi}(t_{m-2})}{2\,\Delta t}+O(\Delta t^3)\tag{31b}$$

в системе (30) на m–м временном слое:

$$\mathbf{M}^\chi\,\frac{\boldsymbol{\xi}^m-2\,\boldsymbol{\xi}^{m-1}+\boldsymbol{\xi}^{m-2}}{\Delta t^2}+\mathbf{M}^\sigma\,\frac{\boldsymbol{\xi}^m-\boldsymbol{\xi}^{m-2}}{2\,\Delta t}+$$
$$\mathbf{S}\,\frac{\boldsymbol{\xi}^m+\boldsymbol{\xi}^{m-2}}{2}+\frac{1}{2}\left(\mathbf{R}(t_m)\,\boldsymbol{\xi}^m+\mathbf{R}(t_{m-2})\,\boldsymbol{\xi}^{m-2}\right)=\frac{1}{2}\left(\mathbf{f}(t_m)+\mathbf{f}(t_{m-2})+\mathbf{r}(t_m)+\mathbf{r}(t_{m-2})\right),\tag{32a}$$
$$\mathbf{M}^\chi\,\frac{\boldsymbol{\xi}^m-2\,\boldsymbol{\xi}^{m-1}+\boldsymbol{\xi}^{m-2}}{\Delta t^2}+\mathbf{M}^\sigma\,\frac{\boldsymbol{\xi}^m-\boldsymbol{\xi}^{m-2}}{2\,\Delta t}+\Big(\mathbf{S}+\mathbf{R}(t_m)\Big)\,\boldsymbol{\xi}^m=\mathbf{f}(t_m)+\mathbf{r}(t_m).\tag{32b}$$

Разностная схема (32a) называется **трёхслойной схемой Крэнка—Николсона** (*Crank—Nicolson scheme*, CN3), а разностная схема (32b) — **трёхслойной неявной схемой** (*Backward difference formula*, BDF3).

Снимем предположение о константности временного шага Δt_m. Вывести аналог трёхслойных разностных операторов (31) для первой и второй производной проще всего в терминах так называемого *временного базиса*.

Ясно, что для полиномов второго порядка ошибка аппроксимации $O(\Delta t^3)$ в (31) равна нулю[7]. Предположим, что на отрезке $[t_{m-2},\,t_m]$ вектор весов есть некоторая квадратичная функция, $\boldsymbol{\xi}|_{[t_{m-2},\,t_m]}\in\operatorname{span}\{\,t^2,\,t,\,1\,\}=\operatorname{span}\{\,\eta^{m-2}(t),\,\eta^{m-1}(t),\,\eta^m(t)\,\}$.
Здесь $\eta^{m-2}(t)$, $\eta^{m-1}(t)$ и $\eta^m(t)$ есть квадратичные функции вершинного временного базиса, такие что $\eta^{m-j}(t_{m-k})=\delta_{j\,k}$ (см. рис. 11). Их аналитический вид легко получить аналогично (15).

[7] Четвёртый член ряда Тейлора включает в себя третью производную, которая равна нулю у функций вида $c_1\,t^2+c_2\,t+c_3$; производные высших порядков также дадут ноль. Поэтому дифференциальный и разностный операторы в (31) будут совпадать.

Тогда $\boldsymbol{\xi}(t)$ можно представить в виде

$$\boldsymbol{\xi}(t) = \eta^{m-2}(t)\,\boldsymbol{\xi}(t_{m-2}) + \eta^{m-1}(t)\,\boldsymbol{\xi}(t_{m-1}) + \eta^{m}(t)\,\boldsymbol{\xi}(t_m), \quad t \in [t_{m-2},\, t_m]$$

и разностные схемы (32) будут иметь вид

$$\mathbf{M}^\chi \sum_{k=0}^{2} \ddot{\eta}^{m-k}(t_m)\,\boldsymbol{\xi}^{m-k} + \mathbf{M}^\sigma \sum_{k=0}^{2} \dot{\eta}^{m-k}(t_m)\,\boldsymbol{\xi}^{m-k} + \mathbf{S}\,\frac{\boldsymbol{\xi}^m + \boldsymbol{\xi}^{m-2}}{2} +$$
$$\frac{1}{2}\Big(\mathbf{R}(t_m)\,\boldsymbol{\xi}^m + \mathbf{R}(t_{m-2})\,\boldsymbol{\xi}^{m-2}\Big) = \frac{1}{2}\Big(\mathbf{f}(t_m) + \mathbf{f}(t_{m-2}) + \mathbf{r}(t_m) + \mathbf{r}(t_{m-2})\Big), \tag{33a}$$

$$\mathbf{M}^\chi \sum_{k=0}^{2} \ddot{\eta}^{m-k}(t_m)\,\boldsymbol{\xi}^{m-k} + \mathbf{M}^\sigma \sum_{k=0}^{2} \dot{\eta}^{m-k}(t_m)\,\boldsymbol{\xi}^{m-k} + \Big(\mathbf{S} + \mathbf{R}(t_m)\Big)\,\boldsymbol{\xi}^m = \mathbf{f}(t_m) + \mathbf{r}(t_m). \tag{33b}$$

Очевидно, решение по обеим схемам эквивалентно решению последовательности СЛАУ вида

$$\mathbf{A}^m\,\boldsymbol{\xi}^m = \mathbf{b}^m, \quad m = 2,\, 3,\, \ldots,\, l.$$

Для (33a) имеем

$$\mathbf{A}^m = \ddot{\eta}^m(t_m)\,\mathbf{M}^\chi + \dot{\eta}^m(t_m)\,\mathbf{M}^\sigma + \frac{1}{2}\Big(\mathbf{S} + \mathbf{R}(t_m)\Big), \tag{34a}$$

$$\begin{aligned}\mathbf{b}^m = {}& \frac{1}{2}\Big(\mathbf{f}(t_m) + \mathbf{f}(t_{m-2}) + \mathbf{r}(t_m) + \mathbf{r}(t_{m-2}) - \big(\mathbf{S} + \mathbf{R}(t_{m-2})\big)\,\boldsymbol{\xi}^{m-2}\Big) \\ & - \ddot{\eta}^{m-1}(t_m)\,\mathbf{M}^\chi\,\boldsymbol{\xi}^{m-1} - \ddot{\eta}^{m-2}(t_m)\,\mathbf{M}^\chi\,\boldsymbol{\xi}^{m-2} \\ & - \dot{\eta}^{m-1}(t_m)\,\mathbf{M}^\sigma\,\boldsymbol{\xi}^{m-1} - \dot{\eta}^{m-2}(t_m)\,\mathbf{M}^\sigma\,\boldsymbol{\xi}^{m-2},\end{aligned} \tag{34b}$$

а для (33b) —

$$\mathbf{A}^m = \ddot{\eta}^m(t_m)\,\mathbf{M}^\chi + \dot{\eta}^m(t_m)\,\mathbf{M}^\sigma + \mathbf{S} + \mathbf{R}(t_m), \tag{35a}$$

$$\begin{aligned}\mathbf{b}^m = {}& \mathbf{f}(t_m) + \mathbf{r}(t_m) \\ & - \ddot{\eta}^{m-1}(t_m)\,\mathbf{M}^\chi\,\boldsymbol{\xi}^{m-1} - \ddot{\eta}^{m-2}(t_m)\,\mathbf{M}^\chi\,\boldsymbol{\xi}^{m-2} \\ & - \dot{\eta}^{m-1}(t_m)\,\mathbf{M}^\sigma\,\boldsymbol{\xi}^{m-1} - \dot{\eta}^{m-2}(t_m)\,\mathbf{M}^\sigma\,\boldsymbol{\xi}^{m-2}.\end{aligned} \tag{35b}$$

Аналитический вид производных временного базиса доступен в II.2.1.

Для начала работы схемы необходимы векторы $\boldsymbol{\xi}^0$ и $\boldsymbol{\xi}^1$. Воспользуемся начальными условиями (25) и положим

$$\xi_i^0 = u(\mathbf{x}_i,\, t_0) = u_0(\mathbf{x}_i), \tag{36a}$$

$$\begin{aligned}\xi_i^1 &= u(\mathbf{x}_i,\, t_1) \simeq u(\mathbf{x}_i,\, t_0) + \dot{u}(\mathbf{x}_i,\, t_0)\,\Delta t_1 \\ &= \mathbf{x}_i + v_0(\mathbf{x}_i)\,\Delta t_1, \quad \mathbf{x}_1,\, \mathbf{x}_2,\, \ldots,\, \mathbf{x}_n \text{ — узлы сетки } \Omega_h.\end{aligned} \tag{36b}$$

Погрешность аппроксимации в (36b) есть $O(\Delta t_1^2)$, что на порядок ниже, чем погрешность аппроксимации в (31). Поэтому при анализе порядка сходимости на модельных задачах (задачах с известным аналитическим решением) в разделе II.3 мы положим

$$\xi_i^1 = u(\mathbf{x}_i,\, t_1).$$

Процесс генерации портрета матрицы, сборки СЛАУ, — учёт вноса вкладов от локальных матриц и векторов, — и процесс решения аналогичны тому, это описывалось в секции **??**, поэтому мы не будем обсуждать это здесь.

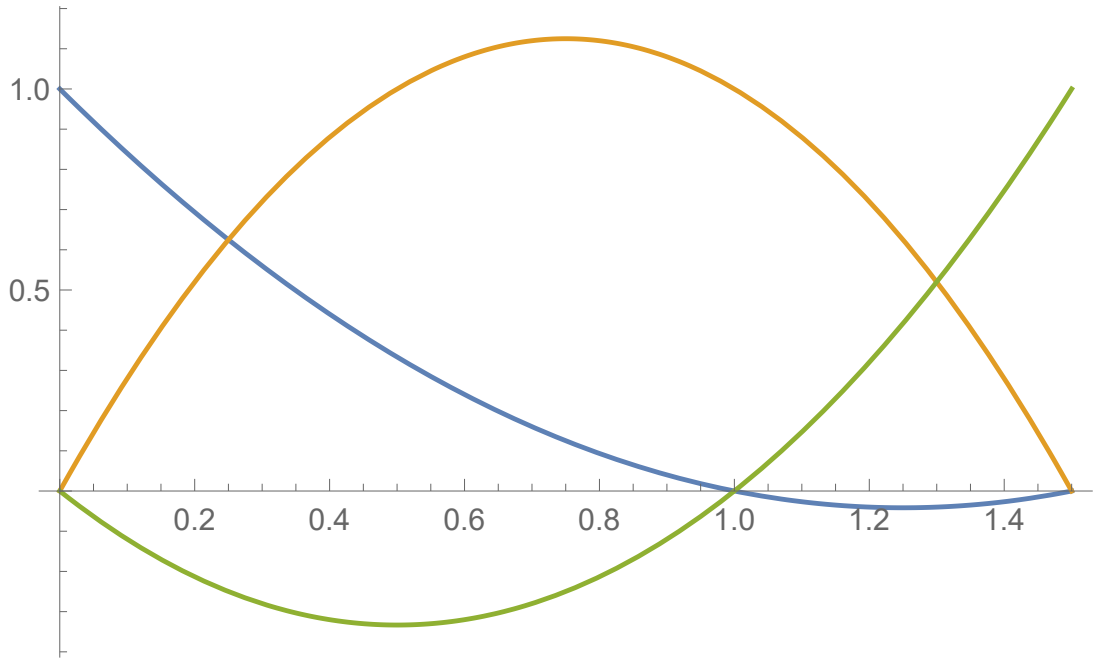

Рис. 11: Функции временного базиса $\eta^{m-2}(t)$, $\eta^{m-1}(t)$ и $\eta^{m}(t)$, $t_{m-2} = 0$, $t_{m-1} = 1$ и $t_{m-1} = 1.5$

II.2.1 Формулы для производных временного базиса

$$\dot{\eta}^{m-2}(t_m) = \frac{t_m - t_{m-1}}{(t_{m-2} - t_{m-1})(t_{m-2} - t_m)} \qquad \ddot{\eta}^{m-2}(t_m) = \frac{2}{(t_{m-2} - t_{m-1})(t_{m-2} - t_m)}$$
$$\dot{\eta}^{m-1}(t_m) = \frac{t_{m-2} - t_m}{(t_{m-2} - t_{m-1})(t_{m-1} - t_m)} \qquad \ddot{\eta}^{m-1}(t_m) = \frac{2}{(t_{m-2} - t_{m-1})(t_m - t_{m-1})}$$
$$\dot{\eta}^{m}(t_m) = \frac{t_{m-2} + t_{m-1} - 2t_m}{(t_{m-2} - t_m)(t_m - t_{m-1})} \qquad \ddot{\eta}^{m}(t_m) = \frac{2}{(t_{m-2} - t_m)(t_{m-1} - t_m)}$$

II.3 Тестирование на модельных задачах

II.3.1 Расчётная область

Следующие в разделе II.3 модельные задачи будут решены на триангуляции Ω_h, построенной для квадрата Ω с вершинами (0, 0), (1, 0), (1, 1) и (0, 1) и с краевыми условиями всех типов (см. рис. 12).

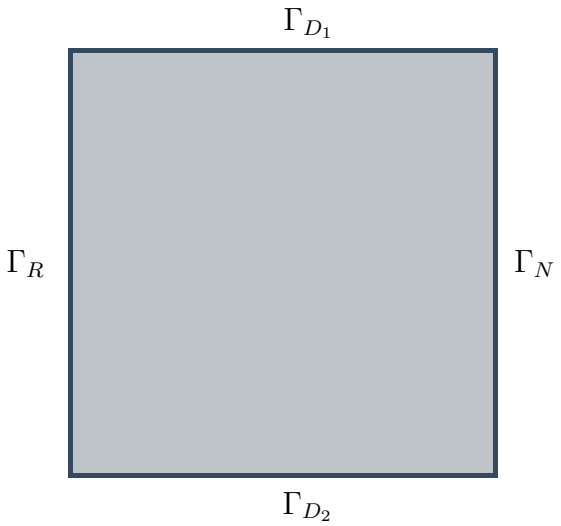

Рис. 12: Расчётная область Ω_h

II.3.2 Априорная оценка ошибки

Известно, что численное решение (см. [2, с. 124]) удовлетворяет

$$\|u(\cdot,\, t_m) - u_h(\cdot,\, t_m)\|_{L_2(\Omega)} \le C\, h^2 \left(\|u_0''\|_{L_2(\Omega)} + \int_{t_0}^{t_l} \|\dot{u}''(\cdot,\, s)\|_{L_2(\Omega)}\, \mathrm{d}s \right) + C\, k \int_{t_0}^{t_l} \|\ddot{u}''(\cdot,\, s)\|_{L_2(\Omega)}\, \mathrm{d}s, \tag{37}$$

где $h \coloneqq$ максимальная длина ребра триангуляции Ω_h.
Отсюда видно, что имеет место **2-й порядок сходимости по пространству и 1-й — по времени**.

Далее мы попробуем на модельных задачах апостериорно установить порядок сходимости по времени и пространству в смысле 2–нормы.

II.3.3 «Простая» задача

Рассмотрим сначала задачу

$$2\,\ddot{u} + \dot{u} - \nabla^2 u = 2\,(2 + t), \quad \mathbf{x} \in \Omega, \quad t \in [0,\, 1]\,, \tag{38}$$

с краевыми условиями

$$\begin{aligned} (-1,\, 0) \cdot \nabla u + u &= t^2 + x - 2\,y - 1, && \mathbf{x} \in \Gamma_R, \\ (1,\, 0) \cdot \nabla u &= 1, && \mathbf{x} \in \Gamma_N, \\ u &= t^2 + x - 2\,y, && \mathbf{x} \in \Gamma_{D_1} \cup \Gamma_{D_1}, \end{aligned}$$

и начальными условиями

$$\begin{aligned} u(\mathbf{x}, 0) &= x - 2\,y, \\ \dot{u}(\mathbf{x}, 0) &= 0. \end{aligned}$$

Она имеет аналитическое решение $u(\mathbf{x}, t) = t^2 + x - 2\,y$.

II.3.3.a Решение по схеме `CN3`

h	n	Δt	l	Δ_h	δ_h	$\frac{\delta_{2\,h}}{\delta_h}$
$\frac{\sqrt{2}}{6}$	49	$\frac{1}{2}$	3	3.62×10^{-1}	5.76×10^{-2}	
$\frac{\sqrt{2}}{6}$	49	$\frac{1}{8}$	9	1.8×10^{-1}	2.86×10^{-2}	2.01
$\frac{\sqrt{2}}{6}$	49	$\frac{1}{32}$	33	4.87×10^{-2}	7.79×10^{-3}	3.67
$\frac{\sqrt{2}}{6}$	49	$\frac{1}{128}$	129	1.24×10^{-2}	1.99×10^{-3}	3.92
$\frac{\sqrt{2}}{6}$	49	$\frac{1}{512}$	513	3.11×10^{-3}	4.99×10^{-4}	3.98

Таблица 2: Решение «простой» II.3.3 задачи по схеме `CN3`, дробление по времени

h	n	Δt	l	Δ_h	δ_h	$\frac{\delta_{2\,h}}{\delta_h}$
$\frac{\sqrt{2}}{3}$	16	$\frac{1}{2}$	3	$2. \times 10^{-1}$	5.16×10^{-2}	
$\frac{\sqrt{2}}{6}$	49	$\frac{1}{8}$	9	1.8×10^{-1}	2.86×10^{-2}	1.8
$\frac{\sqrt{2}}{12}$	169	$\frac{1}{32}$	33	9.33×10^{-2}	8.45×10^{-3}	3.38
$\frac{\sqrt{2}}{24}$	625	$\frac{1}{128}$	129	4.65×10^{-2}	2.26×10^{-3}	3.74
$\frac{\sqrt{2}}{48}$	2401	$\frac{1}{512}$	513	2.31×10^{-2}	5.84×10^{-4}	3.88

Таблица 3: Решение «простой» задачи II.3.3 по схеме `CN3`, дробление по времени и пространству

Поясним нотации в таблицах 2 – 3:

- h — максимальная длина ребра сетки Ω_h,
- n — количество узлов триангуляции (размерность СЛАУ, решаемой на каждом временном слое t_m),
- Δt — размер шага по времени,
- l — количество временных слоёв,
- $\Delta_h \coloneqq \max_{0\le m\le l} \|\mathbf{u}^m - \boldsymbol{\xi}^m\|_2$ — максимальная 2-норма ошибки,
  $\mathbf{u}^m \coloneqq (u^m(\mathbf{x}_1),\, u^m(\mathbf{x}_2),\, \ldots,\, u^m(\mathbf{x}_n))$ — вектор значений точного решения u на m–м временном слое,
- $\delta_h \coloneqq \max_{0\le m\le l} \frac{\|\mathbf{u}^m - \boldsymbol{\xi}^m\|_2}{\|\mathbf{u}^m\|_2}$ — максимальная относительная погрешность и
- $\frac{\delta_{2\,h}}{\delta_h}$ — отношение предыдущей относительной погрешности к текущей.

II.3.3.b Решение по схеме `BDF3`

Решение по схеме `BDF3` сразу же при $n = 49$ и $m = 3$ (см. первую строку таблиц 2 – 3) даёт ошибку $\Delta_h = 3.21 \times 10^{-15}$ и относительную ошибку $\delta_h = 5.12 \times 10^{-16}$, которые сохраняются незначительными с дроблением временного шага.

II.3.4 «Косинус–волна»

Рассмотрим теперь задачу

$$
\begin{aligned}
&\ddot{u} + .1\,\dot{u} - \nabla\cdot(x\,y\,\nabla u) = \\
&5\,(x + y - .1)\,\sin\big(5\,(t + x + y)\big) + 25(2\,x\,y - 1)\,\cos\big(5\,(t + x + y)\big), \\
&\mathbf{x}\in\Omega, \quad t\in[0,\,1]\,,
\end{aligned}
$$

с краевыми условиями

$$
\begin{aligned}
(-1,\,0)\cdot\nabla u + u &= 5\,x\,y\,\sin\big(5\,(t + x + y)\big) + \cos\big(5\,(t + x + y)\big), && \mathbf{x}\in\Gamma_R,\\
(1,\,0)\cdot\nabla u &= -5\,x\,y\,\sin\big(5\,(t + x + y)\big), && \mathbf{x}\in\Gamma_N,\\
u &= \cos\big(5\,(t + x + y)\big), && \mathbf{x}\in\Gamma_{D_1}\cup\Gamma_{D_1},
\end{aligned}
$$

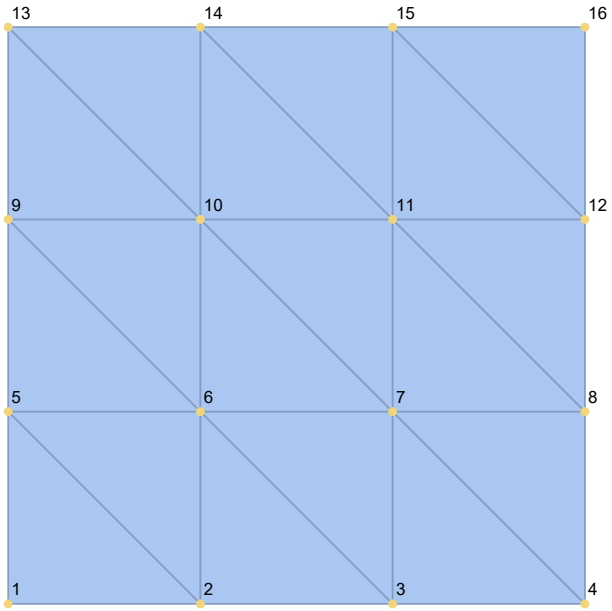

Рис. 13: Сетка Ω_h из 2–й строки таблицы 3, $h = \frac{\sqrt{2}}{3}$ и ...

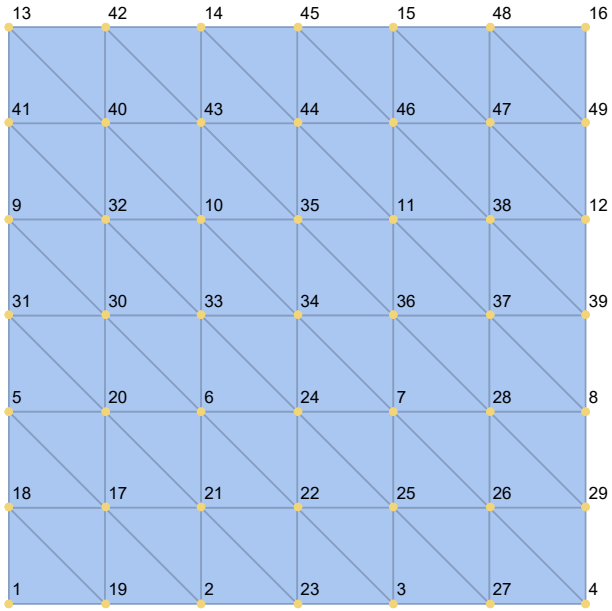

Рис. 14: ... измельчённая сетка из 3–й строки, $h = \frac{\sqrt{2}}{6}$

и начальными условиями

$$u(\mathbf{x}, 0) = \cos\big(5\,(x+y)\big),$$
$$\dot{u}(\mathbf{x}, 0) = -5\sin\big(5\,(x+y)\big).$$

Она имеет аналитическое решение $u(\mathbf{x}, t) = \cos\big(5\,(t+x+y)\big)$.

II.3.4.a Решение по схеме CN3

h	n	Δt	l	Δ_h	δ_h	$\frac{\delta_{2\,h}}{\delta_h}$
$\frac{\sqrt{2}}{2}$	9	$\frac{1}{2}$	3	4.15	1.86	
$\frac{\sqrt{2}}{4}$	25	$\frac{1}{8}$	9	7.4×10^{-1}	2.09×10^{-1}	8.87
$\frac{\sqrt{2}}{8}$	81	$\frac{1}{32}$	33	4.27×10^{-1}	6.7×10^{-2}	3.13
$\frac{\sqrt{2}}{16}$	289	$\frac{1}{128}$	129	2.6×10^{-1}	2.15×10^{-2}	3.11
$\frac{\sqrt{2}}{32}$	1089	$\frac{1}{512}$	513	1.36×10^{-1}	5.78×10^{-3}	3.72

Таблица 4: Решение задачи II.3.4 по схеме CN3, дробление по времени и пространству

II.3.4.b Решение по схеме BDF3

h	n	Δt	l	Δ_h	δ_h	$\frac{\delta_{2\,h}}{\delta_h}$
$\frac{\sqrt{2}}{2}$	9	$\frac{1}{2}$	3	2.88	1.29	
$\frac{\sqrt{2}}{4}$	25	$\frac{1}{8}$	9	4.63	1.31	0.985
$\frac{\sqrt{2}}{8}$	81	$\frac{1}{32}$	33	2.57	4.02×10^{-1}	3.25
$\frac{\sqrt{2}}{16}$	289	$\frac{1}{128}$	129	1.34	1.11×10^{-1}	3.64
$\frac{\sqrt{2}}{32}$	1089	$\frac{1}{512}$	513	6.71×10^{-1}	2.86×10^{-2}	3.87

Таблица 5: Решение задачи II.3.4 по схеме BDF3, дробление по времени и пространству

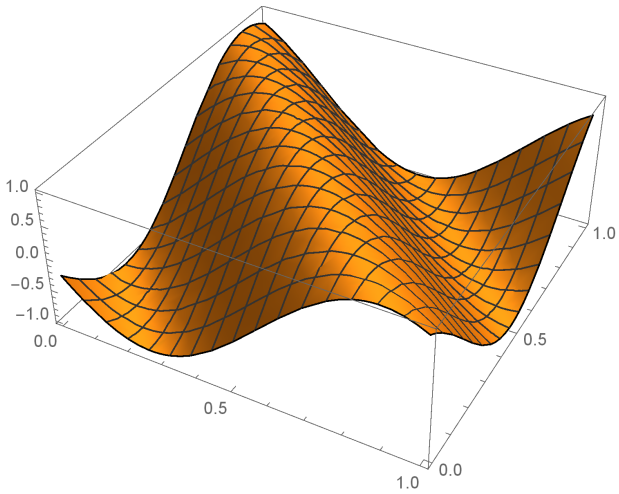

Рис. 15: Аналитическое решение $u(\mathbf{x}, .375) = \cos\left(5\,(.375 + x + y)\right)$ задачи II.3.4 и ...

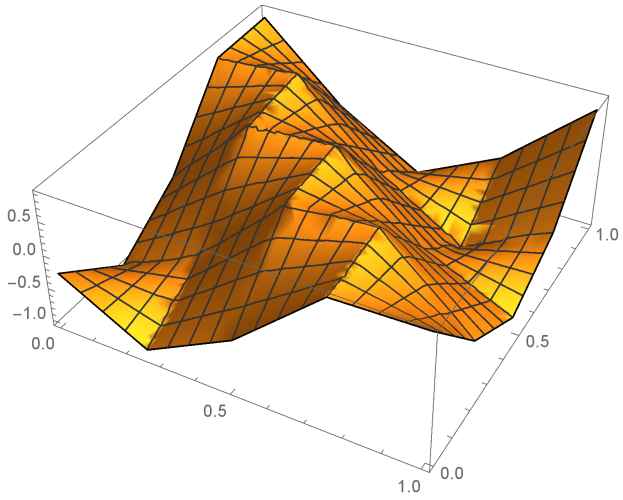

Рис. 16: ... численное решение u_h на 4–м временном слое $t_4 = .375$, определённое на сетке с $n = 25$ узлами — см. 3–ю строку таблицы 4

II.3.5 Выводы

Главное, на что следует обратить внимание в решениях модельных задач II.3.3 и II.3.4:

- Во всех таблицах (за исключением 2) видно, что мы дробим шаг по пространству h в два раза, а шаг по времени Δt — в четыре. При этом ясно наблюдается сходимость отношения максимумов относительных погрешностей к четырём.
  Отсюда делаем вывод, что **численное решение u_h в смысле 2-нормы сходится к аналитическому со вторым порядком по пространству и с первым — по времени** (см. априорную оценку (37)).
  Сходимость справедлива для обоих тестов.

- Очевидно, что решение задачи II.3.3 $u(\mathbf{x}, t) = t^2 + x - 2\,y$ для любого t есть житель подпространства $\mathbb{V}_h$, в котором мы ищем приближение к аналитическому решению. Стало быть нет необходимости в дроблении пространственной сетки (и, как следствие, в нагрузке на ресурсы компьютера) — на скорость сходимости это не влияет, как видно из таблиц 2 и 3.

- На «простом» тесте II.3.3 `CN3` сработала гораздо хуже `BDF3`, однако
- для уравнения с «косинус–волной» II.3.4 «усреднённые»[8] вклады от соседних временных слоёв в (35) дали о себе знать: погрешность $\Delta_h = 1.36 \times 10^{-1}$ метода `CN3` в 5 раз меньше, чем погрешность $\Delta_h = 6.71 \times 10^{-1}$ метода `BDF3`. Более того, из таблицы 5 видно, что только на самом мелком шаге по времени и пространству погрешность метода `BDF3` упала до второго знака;
- отсюда делаем вывод, что — хотя оба метода обладают одним порядком сходимости — скорость сходимости может очень сильно зависеть от конкретной задачи.

[8] Говорят, что схемы типа Крэнка–Николсона отражают закон сохранения энергии, свойственный некоторым волновым уравнениям, не давая решению «расплываться» — см. [2, с. 137].

II.4 Исходный текст программы

Здесь мы приведём основной модуль — пространство имён `FEMt`, содержащее решатели `CN3` и `BDF3` и вспомогательные функции для расчёта локальных матриц и векторов. Весь комплекс программ, как было отмечено в начале, доступен в репозитории: https://github.com/CATSPDEs

```
1  #include "FEMt.hpp"
2  #include "SymmetricCSlRMatrix.hpp" //for final linear system matrix
3  #include "krylov.hpp" //conjugate gradients
4  #include "array.hpp" //utility for array operations
5
6  vector<vector<double>> FEMt::CN3(HyperbolicPDE const & PDE,
7                                   InitialBoundaryConditions& IBCs,
8                                   vector<double> const & t, //vector of time
                                       frames
9                                   Triangulation& Omega,
10                                 TimeFunction u) { //exact soln (if we are dealing
                                       w/ model problem)
11   if (t.size() < 2) throw invalid_argument("CN3() needs at least 2 time
       frames to compute soln");
12   //data structures for final linear system A.xi[m] = b:
13   SymmetricCSlRMatrix A(Omega.generateAdjList()); //build final matrix portrait
14   vector<double> b(Omega.numbOfNodes(), 0.); //load vector
15   vector<vector<double>> xi(t.size(), b); //discrete solution--xi_{mi} is our
       solution at time = t_m and at node_i
16   //data structures for assemby of A and b:
17   SymmetricContainer<double> localMassMatrixChi(3),
       localMassMatrixSigma(3), //for hat functions on
       triangles
18                                 localStiffnessMatrix(3), //we have 3 × 3 element
                                       matricies
19                                 localRobinMatrix_m(2), //and 2 × 2 element
                                       matricies for Robin BCs (just like element matrix in
                                       1D)
20                                 localRobinMatrix_m_2(2); //computed on t_m and
                                       t_{m-2}--for CN3-scheme
21   array<double, 3> localLoadVector_m, localLoadVector_m_2; //and their
22   array<double, 2> localRobinVector_m, localRobinVector_m_2; //friends,
       element vectors
23   array<Node, 3> elementNodes, //nodes of the current triangle
24                  elementMiddleNodes; //and nodes on the middle of edges
25   array<Node, 2> edgeNodes; //nodes spanning an edge of the current triangle that is
       part of bndry
26   double measure; //area of ith triangle / length of bndry edge of ith thiangle
27   array<size_t, 3> l2g_elem; //local to global mapping of nodes on the element
28   array<size_t, 2> l2g_edge; //and on the edge
29   localIndex j, k, leftNodeIndex, rightNodeIndex; //dummy indicies
30   array<array<double, 2>, 3> eta; //time-basis
31   //stencil:
32   //t_{-2} - - - - - t_{-1} - - - t_0
33   //eta_i = unity at t_{-i}, zero elsewhere, eta_i = at^2 + bt + c
34   //etam i means (j+1)'s derivative of eta_i computed at t_0
35   //we will need these derivatives to approximate differential operators
36   //(I) apply initial conditions to get xi_0 and xi_1
37   for (size_t i = 0; i < Omega.numbOfNodes(); ++i)
```

```
38      xi[0][i] = IBCs.initialPosition(Omega.getNode(i));
39    if (u == emptyTimeFunc) for (size_t i = 0; i < Omega.numbOfNodes(); ++i)
40      //normally, we have to use initial velocity to approximate xi_1 ...
41      xi[1][i] = xi[0][i] + IBCs.initialVelocity(Omega.getNode(i)) * (t[1] -
           t[0]);
42    else for (size_t i = 0; i < Omega.numbOfNodes(); ++i)
43      //...but for model problems we have exact soln, so
44      xi[1][i] = u(Omega.getNode(i), t[1]);
45    //(II) solve for xi_m, m = 2, 3, ...w/ Crank-Nicolson scheme
46    for (size_t m = 2; m < t.size(); ++m) {
47      //compute derivatives
48      eta[2][0] = (t[m] - t[m - 1])                    / (t[m - 2] - t[m - 1])
           / (t[m - 2] - t[m]);
49      eta[2][1] = 2.                                   / (t[m - 2] - t[m - 1])
           / (t[m - 2] - t[m]);
50      eta[1][0] = (t[m - 2] - t[m])                    / (t[m - 2] - t[m - 1])
           / (t[m - 1] - t[m]);
51      eta[1][1] = - 2.                                 / (t[m - 2] - t[m - 1])
           / (t[m - 1] - t[m]);
52      eta[0][0] = (2. * t[m] - t[m - 2] - t[m - 1]) / (t[m - 2] - t[m]) /
           (t[m - 1] - t[m]);
53      eta[0][1] = 2.                                   / (t[m - 2] - t[m]) /
           (t[m - 1] - t[m]);
54      //assemble SLDE
55      for (size_t i = 0; i < Omega.numbOfTriangles(); ++i) {
56        //(1) quadratures over elements
57        //in order to assemble A and b,
58        //it is convenient to iterate over mesh elements (i.e. triangles)
59        elementNodes = Omega.getNodes(i); //get nodes of ith triangle
60        for (j = 0; j < 3; ++j) //and middle nodes of its edges
61          elementMiddleNodes[j] = elementNodes[k =
               nextIndex(j)].midPoint(elementNodes[nextIndex(k)]);
62        measure = Omega.area(i); //compute area of ith triangle
63        l2g_elem = Omega.l2g(i); //local to global mapping of nodes of ith element
64        //local matrices and vectors
65        localMassMatrixChi = computeLocalMassMatrix(PDE.chi(), elementNodes,
             measure);
66        localMassMatrixSigma = computeLocalMassMatrix(PDE.sigma(),
             elementNodes, measure);
67        localStiffnessMatrix =
             computeLocalStiffnessMatrix(PDE.diffusionTerm(), elementNodes,
             elementMiddleNodes, measure);
68        localLoadVector_m = computeLocalLoadVector(PDE.forceTerm(), t[m],
             elementNodes, elementMiddleNodes, measure);
69        localLoadVector_m_2 = computeLocalLoadVector(PDE.forceTerm(), t[m -
             2], elementNodes, elementMiddleNodes, measure);
70        //assemble contributions
71        for (j = 0; j < 3; ++j) {
72          for (k = 0; k < 3; ++k) {
73            if (k >= j) //symmetric
74              A(l2g_elem[j], l2g_elem[k]) +=
75                eta[0][0] * localMassMatrixSigma(j, k) +
76                eta[0][1] * localMassMatrixChi(j, k) +
77                localStiffnessMatrix(j, k) / 2.;
78            b[l2g_elem[j]] -= (
```

```
79              localStiffnessMatrix(j, k) * xi[m - 2][l2g_elem[k]] / 2. +
80              eta[1][0] * localMassMatrixSigma(j, k) * xi[m -
                    1][l2g_elem[k]] + //parabolic
81              eta[2][0] * localMassMatrixSigma(j, k) * xi[m -
                    2][l2g_elem[k]] +
82              eta[1][1] * localMassMatrixChi(j, k)   * xi[m -
                    1][l2g_elem[k]] + //hyperbolic
83              eta[2][1] * localMassMatrixChi(j, k)   * xi[m - 2][l2g_elem[k]]
84            );
85          }
86          b[l2g_elem[j]] += (localLoadVector_m[j] + localLoadVector_m_2[j])
                / 2.;
87        }
88        //(2) quadratures over edges
89        //iterate over list of local indicies of boundary nodes
90        for (localIndex edgeIndex : Omega.getBoundaryIndicies(i)) {
91          //if edgeIndex = 2, then the edge against second node of ith triangle
92          //is part of the boundary
93          //so we need to assemble BCs here
94          leftNodeIndex = nextIndex(edgeIndex); //local indicies of nodes that
95          rightNodeIndex = nextIndex(leftNodeIndex); //define the edge
96          edgeNodes = { elementNodes[leftNodeIndex],
                elementNodes[rightNodeIndex] }; //and the nodes
                themselves
97          l2g_edge[0] = l2g_elem[leftNodeIndex]; //local to global nodes
98          l2g_edge[1] = l2g_elem[rightNodeIndex]; //numeration mapping
99          measure = Omega.length(i, edgeIndex);
100         //compute
101         //(2.1) local Robin matrix
102         //(2.2) local Robin vector
103         localRobinMatrix_m = FEMt::computeLocalRobinMatrix(IBCs, t[m],
                edgeNodes, measure);
104         localRobinMatrix_m_2 = FEMt::computeLocalRobinMatrix(IBCs, t[m -
                2], edgeNodes, measure);
105         localRobinVector_m = FEMt::computeLocalRobinVector(IBCs, t[m],
                edgeNodes, measure);
106         localRobinVector_m_2 = FEMt::computeLocalRobinVector(IBCs, t[m -
                2], edgeNodes, measure);
107         //(2.3) assemble contributions
108         for (j = 0; j < 2; ++j) {
109           for (k = 0; k < 2; ++k) {
110             if (k >= j) //symmetric
111               A(l2g_edge[j], l2g_edge[k]) += localRobinMatrix_m(j, k) / 2.;
112             b[l2g_edge[j]] -= localRobinMatrix_m_2(j, k) * xi[m -
                    2][l2g_edge[k]] / 2.;
113           }
114           b[l2g_edge[j]] += (localRobinVector_m[j] +
                  localRobinVector_m_2[j]) / 2.;
115         }
116       }
117     }
118     //now we are ready to compute xi[m], A.xi[m] = b
119     xi[m] = CG(A, b, xi[m - 1], 10e-70);
120     //clear A and b
121     A.setZero();
```

```
122       fill(b.begin(), b.end(), 0.);
123     }
124     return xi;
125   }
126
127   vector<vector<double>> FEMt::BDF3(HyperbolicPDE const & PDE,
128                                     InitialBoundaryConditions& IBCs,
129                                     vector<double> const & t, //vector of time
                                          frames
130                                     Triangulation& Omega,
131                                     TimeFunction u) { //exact soln (if we are
                                          dealing w/ model problem)
132     //. . .
133   }
134
135
136   SymmetricContainer<double> FEMt::computeLocalMassMatrix(Function
        reactionTerm,
137                                                           array<Node, 3>&
                                                              nodes,
138                                                           double area) {
139     SymmetricContainer<double> m(3);
140     m(0, 0) = area * (6. * reactionTerm(nodes[0]) + 2. *
          reactionTerm(nodes[1]) + 2. * reactionTerm(nodes[2])) / 60.;
141     m(0, 1) = area * (2. * reactionTerm(nodes[0]) + 2. *
          reactionTerm(nodes[1]) + reactionTerm(nodes[2])) / 60.;
142     m(0, 2) = area * (2. * reactionTerm(nodes[0]) + reactionTerm(nodes[1]) +
          2. * reactionTerm(nodes[2])) / 60.;
143     m(1, 1) = area * (2. * reactionTerm(nodes[0]) + 6. *
          reactionTerm(nodes[1]) + 2. * reactionTerm(nodes[2])) / 60.;
144     m(1, 2) = area * (reactionTerm(nodes[0]) + 2. * reactionTerm(nodes[1]) +
          2. * reactionTerm(nodes[2])) / 60.;
145     m(2, 2) = area * (2. * reactionTerm(nodes[0]) + 2. *
          reactionTerm(nodes[1]) + 6. * reactionTerm(nodes[2])) / 60.;
146     return m;
147   }
148
149   SymmetricContainer<double> FEMt::computeLocalStiffnessMatrix(Function
        diffusionTerm,
150                                                                array<Node,
                                                                      3>& nodes,
151                                                                array<Node,
                                                                      3>&
                                                                      middleNodes,
152                                                                double area) {
153     SymmetricContainer<double> s(3);
154     s(0, 0) = (nodes[1].x() - nodes[2].x()) * (nodes[1].x() - nodes[2].x()) +
155       (nodes[1].y() - nodes[2].y()) * (nodes[1].y() - nodes[2].y());
156     s(0, 1) = (nodes[0].x() - nodes[2].x()) * (nodes[2].x() - nodes[1].x()) +
157       (nodes[0].y() - nodes[2].y()) * (nodes[2].y() - nodes[1].y());
158     s(0, 2) = (nodes[0].x() - nodes[1].x()) * (nodes[1].x() - nodes[2].x()) +
159       (nodes[0].y() - nodes[1].y()) * (nodes[1].y() - nodes[2].y());
160     s(1, 1) = (nodes[0].x() - nodes[2].x()) * (nodes[0].x() - nodes[2].x()) +
161       (nodes[0].y() - nodes[2].y()) * (nodes[0].y() - nodes[2].y());
162     s(1, 2) = (nodes[1].x() - nodes[0].x()) * (nodes[0].x() - nodes[2].x()) +
```

```
163       (nodes[1].y() - nodes[0].y()) * (nodes[0].y() - nodes[2].y());
164     s(2, 2) = (nodes[0].x() - nodes[1].x()) * (nodes[0].x() - nodes[1].x()) +
165       (nodes[0].y() - nodes[1].y()) * (nodes[0].y() - nodes[1].y());
166     for (localIndex i = 0; i < 3; ++i)
167       for (localIndex j = i; j < 3; ++j)
168         s(i, j) *= (diffusionTerm(middleNodes[0]) +
              diffusionTerm(middleNodes[1]) + diffusionTerm(middleNodes[2])) /
              area / 12.;
169     return s;
170   }
171
172   array<double, 3> FEMt::computeLocalLoadVector(TimeFunction forceTerm,
173                                                 double t,
174                                                 array<Node, 3>& nodes,
175                                                 array<Node, 3>& middleNodes,
176                                                 double area) {
177     array<double, 3> f;
178     return (f = {
179       2. * forceTerm(nodes[0], t) - forceTerm(nodes[1], t) -
              forceTerm(nodes[2], t) +
180       4. * forceTerm(middleNodes[0], t) + 8. * (forceTerm(middleNodes[1], t)
              + forceTerm(middleNodes[2], t)),
181       2. * forceTerm(nodes[1], t) - forceTerm(nodes[0], t) -
              forceTerm(nodes[2], t) +
182       4. * (2. * (forceTerm(middleNodes[0], t) + forceTerm(middleNodes[2],
              t)) + forceTerm(middleNodes[1], t)),
183       2. * (forceTerm(nodes[2], t) + 4. * (forceTerm(middleNodes[0], t) +
              forceTerm(middleNodes[1], t)) + 2. * forceTerm(middleNodes[2], t)) -
184       forceTerm(nodes[0], t) - forceTerm(nodes[1], t)
185     }) *= area / 60.;
186   }
187
188
189   SymmetricContainer<double>
        FEMt::computeLocalRobinMatrix(InitialBoundaryConditions& IBCs,
190                                                               double t,
191                                                               array<Node, 2>&
                                                                    nodes,
192                                                             double length) {
193     IBCs.computeBoundaryType(nodes);
194     double RobinCoefficientLeft  = IBCs.RobinCoefficient(nodes[0], t),
195            RobinCoefficientRight = IBCs.RobinCoefficient(nodes[1], t);
196     SymmetricContainer<double> r(2);
197     r(0, 0) = length * (3. * RobinCoefficientLeft + RobinCoefficientRight) /
            12.;
198     r(0, 1) = length * (RobinCoefficientLeft + RobinCoefficientRight) / 12.;
199     r(1, 1) = length * (RobinCoefficientLeft + 3. * RobinCoefficientRight) /
            12.;
200     return r;
201   }
202
203   array<double, 2> FEMt::computeLocalRobinVector(InitialBoundaryConditions &
        IBCs,
204                                                  double t,
205                                                  array<Node, 2>& nodes,
```

```
206                                                 double length) {
207   IBCs.computeBoundaryType(nodes);
208   double RobinCoefficientLeft    = IBCs.RobinCoefficient(nodes[0], t),
209        RobinCoefficientRight   = IBCs.RobinCoefficient(nodes[1], t),
210        DirichletConditionLeft  = IBCs.DirichletCondition(nodes[0], t),
211        DirichletConditionRight = IBCs.DirichletCondition(nodes[1], t);
212   array<double, 2> r;
213   return (r = {
214     4. * IBCs.NeumannValue(nodes[0], t) + 2. * IBCs.NeumannValue(nodes[1],
          t) +
215     DirichletConditionLeft * (3. * RobinCoefficientLeft +
          RobinCoefficientRight) +
216     DirichletConditionRight * (RobinCoefficientLeft +
          RobinCoefficientRight),
217     2. * IBCs.NeumannValue(nodes[0], t) + 4. * IBCs.NeumannValue(nodes[1],
          t) +
218     DirichletConditionLeft * (RobinCoefficientLeft +
          RobinCoefficientRight) +
219     DirichletConditionRight * (RobinCoefficientLeft + 3. *
          RobinCoefficientRight)
220   }) *= length / 12.;
221 }
```

Список литературы

[1] Walter A. Strauss
Partial Differential Equations: an Introduction
Brown University
John Wiley & Sons, 2008

[2] Mats G. Larson, Fredrik Bengzon
The Finite Element Method: Theory, Implementation, and Applications
Department of Mathematics, Umeå University
Springer, 2013

[3] Баландин М. Ю., Шурина Э. П.
Векторный метод конечных элементов
Новосибирский государственный технический университет
Издательство НГТУ, 2001

[4] Скворцов А. В.
Триангуляция Делоне и её применение
Томский государственный университет
Издательство ТГУ, 2002

[5] Youcef Saad
SPARSKIT: a basic tool kit for sparse matrix computations
CSRD, University of Illinois; RIACS (NASA Ames Research Center)
http://www-users.cs.umn.edu/~saad/software/SPARSKIT/

[6] Youcef Saad
Iterative Methods for Sparse Linear Systems: Second Edition
2003