

# Содержание

<b>1</b>	<b>Сеточное решение уравнения Пуассона</b>	<b>4</b>
1.1	Постановка задачи первого рода . . . . .	4
1.2	Метод конечных разностей . . . . .	4
1.2.1	Метод решения . . . . .	4
1.2.1.1	Нахождение численного решения . . . . .	4
1.2.1.2	Практическое определения порядка аппроксимации . . . . .	5
1.2.2	Программная реализация . . . . .	6
1.2.2.1	Функция верхнего уровня . . . . .	6
1.2.2.2	Детали реализации . . . . .	7
1.3	Метод конечных объёмов . . . . .	10
1.3.1	Конечнообъёмная сетка . . . . .	11
1.3.2	Конечнообъёмная аппроксимация . . . . .	11
1.3.2.1	Обработка внутренних граней . . . . .	12
1.3.2.2	Учёт граничных условий первого рода . . . . .	13
1.3.3	Одномерный случай . . . . .	13
1.3.4	Сборка системы линейных уравнений . . . . .	14
1.3.4.1	Алгоритм сборки в цикле по ячейкам . . . . .	15
1.3.4.2	Алгоритм сборки в цикле по граням . . . . .	15
<b>A</b>	<b>Задания для самостоятельной работы</b>	<b>17</b>
A.1	Лекция 2 (20.09.25) МКО для решения уравнения Пуассона . . . . .	18
A.1.1	Работа с сетками . . . . .	18
A.1.2	Тестовая задача . . . . .	19
A.1.3	График сходимости . . . . .	19
A.1.4	Рекомендации к программированию . . . . .	19
<b>B</b>	<b>Формулы и обозначения</b>	<b>21</b>
B.1	Векторы . . . . .	22
B.1.1	Обозначение . . . . .	22
B.1.2	Набла–нотация . . . . .	22
B.2	Интегрирование . . . . .	24
B.2.1	Формула Гаусса–Остроградского . . . . .	24
B.2.2	Интегрирование по частям . . . . .	24
B.2.3	Численное интегрирование в заданной области . . . . .	25
B.3	Интерполяционные полиномы . . . . .	26
B.3.1	Многочлен Лагранжа . . . . .	26
B.3.1.1	Узловые базисные функции . . . . .	26
B.3.1.2	Интерполяция в параметрическом отрезке . . . . .	27
B.3.1.3	Интерполяция в параметрическом треугольнике . . . . .	30
B.3.1.4	Интерполяция в параметрическом квадрате . . . . .	32

<b>С</b>	<b>Алгоритмы</b>	<b>35</b>
С.1	Геометрические алгоритмы . . . . .	36
С.1.1	Линейная интерполяция . . . . .	36
С.1.2	Преобразование координат . . . . .	36
С.1.2.1	Матрица Якоби . . . . .	37
С.1.2.2	Дифференцирование в параметрической плоскости . . . . .	38
С.1.2.3	Интегрирование в параметрической плоскости . . . . .	39
С.1.2.4	Двумерное линейное преобразование. Параметрический треугольник . . . . .	39
С.1.2.5	Двумерное билинейное преобразование. Параметрический квадрат . . . . .	40
С.1.2.6	Трёхмерное линейное преобразование. Параметрический тетраэдр . . . . .	40
С.1.3	Свойства многоугольника . . . . .	40
С.1.3.1	Площадь многоугольника . . . . .	40
С.1.3.2	Интеграл по многоугольнику . . . . .	42
С.1.3.3	Центр масс многоугольника . . . . .	42
С.1.4	Свойства многогранника . . . . .	43
С.1.4.1	Объём многогранника . . . . .	43
С.1.4.2	Интеграл по многограннику . . . . .	43
С.1.4.3	Центр масс многогранника . . . . .	43
С.1.5	Поиск многоугольника, содержащего заданную точку . . . . .	43
С.2	Форматы хранения разреженных матриц . . . . .	44
С.2.1	CSR-формат . . . . .	44
С.2.2	Массив словарей . . . . .	46
<b>D</b>	<b>Работа с инфраструктурой проекта CFDCourse</b>	<b>48</b>
D.1	Клонирование . . . . .	49
D.2	Разворачивание контейнера . . . . .	50
D.3	Базовая разработка . . . . .	50
D.3.1	Особенности проекта . . . . .	51
D.3.2	Сборка в отладочном режиме . . . . .	51
D.3.3	Сборка в релизном режиме . . . . .	52
D.3.4	Работа с кодом . . . . .	52
D.4	Разработка в vscode . . . . .	52
D.4.1	Подключение к контейнеру . . . . .	52
D.4.2	Настройки vscode . . . . .	53
D.4.3	Сборка и отладка . . . . .	54
D.5	Работа с системой контроля версий . . . . .	55
D.5.1	Порядок работы с репозиторием CFDCourse . . . . .	55
D.5.1.1	Получение последнего коммита . . . . .	55
D.5.1.2	Создание коммита с текущим дз . . . . .	55
D.5.1.3	Создание коммита с прошлым дз . . . . .	56
D.6	Paraview . . . . .	57
D.6.1	Данные на одномерных сетках . . . . .	57

D.6.2	Изолинии для двумерного поля . . . . .	60
D.6.3	Данные на двумерных сетках в виде поверхности . . . . .	61
D.6.4	Числовых значения в точках и ячейках . . . . .	62
D.6.5	Векторные поля . . . . .	62
D.6.6	Значение функции вдоль линии . . . . .	64
D.7	Hybmesh . . . . .	67
D.7.1	Построение сеток . . . . .	67

# 1 Сеточное решение уравнения Пуассона

## 1.1 Постановка задачи первого рода

Будем рассматривать дифференциальное уравнение в области  $\Omega$ :

$$-\nabla^2 u = f(x), \quad x \in \Omega \quad (1.1)$$

$$u = u^\Gamma(x), \quad x \in \partial\Omega \quad (1.2)$$

## 1.2 Метод конечных разностей

Рассмотрим задачу (1.1), (1.2) в упрощённой одномерной постановке:

$$-\frac{\partial^2 u}{\partial x^2} = f(x) \quad (1.3)$$

в области  $x \in [a, b]$  с граничными условиями первого рода

$$\begin{cases} u(a) = u_a, \\ u(b) = u_b. \end{cases} \quad (1.4)$$

Необходимо:

- Запрограммировать расчётную схему для численного решения этого уравнения методом конечных разностей на сетке с постоянным шагом,
- С помощью вычислительных экспериментов подтвердить порядок аппроксимации расчётной схемы.

### 1.2.1 Метод решения

#### 1.2.1.1 Нахождение численного решения

В области решения  $[a, b]$  введём равномерную сетку из  $N$  ячеек. Шаг сетки будет равен  $h = (b-a)/N$ . Узлы сетки запишем в виде сеточного вектора  $\{x_i\}$  длины  $N+1$ , где  $i = \overline{0, N}$ . Определим сеточный вектор  $\{u_i\}$  неизвестных, элементы которого определяют значение искомого численного решения в  $i$ -ом узле сетки.

Разностная схема второго порядка для уравнения (1.3) имеет вид

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i, \quad i = \overline{1, N-1}. \quad (1.5)$$

Здесь  $\{f_i\}$  – известный сеточный вектор, определяемый через известную аналитическую функцию  $f(x)$  в правой части уравнения (1.3) как

$$f_i = f(x_i). \quad (1.6)$$

Аппроксимация граничных условий (1.4) первого рода даёт дополнительные сеточные уравнения для граничных узлов

$$\begin{aligned} u_0 &= u_a, \\ u_N &= u_b \end{aligned} \quad (1.7)$$

Линейные уравнения (1.5), (1.7) составляют систему вида

$$\sum_{j=0}^N A_{ij} u_j = b_i, \quad i = \overline{0, N}$$

с матричными коэффициентами

$$A_{ij} = \begin{cases} 1, & i = 0, j = 0; \\ 2/h^2, & i = \overline{1, N-1}, j = i; \\ -1/h^2, & i = \overline{1, N-1}, j = i-1; \\ -1/h^2, & i = \overline{1, N-1}, j = i+1; \\ 1, & i = N, j = N; \\ 0, & \text{иначе.} \end{cases} \quad (1.8)$$

и правой частью

$$b_i = \begin{cases} u_a, & i = 0; \\ u_b, & i = N; \\ f_i, & i = \overline{1, N-1}. \end{cases} \quad (1.9)$$

Искомый вектор находится путём решения этой системы.

### 1.2.1.2 Практическое определения порядка аппроксимации

Порядок аппроксимации показывает скорость приближения численного решения к точному с уменьшением сетки. Поэтому для подтверждения порядка необходимо

- Знать точное решение,
- Уметь вычислять функционал (норму,  $\|\cdot\|$ ), характеризующий отклонение точного решения от численного,
- Сделать несколько расчётов на сетках с разной  $N$  и заполнить таблицу  $\|\{u_i - u^e(x_i)\}\|(N)$ ,
- На основе этой таблицы построить график в логарифмических осях и по углу наклона кривой сделать вывод о порядке аппроксимации.

Выберем произвольную функцию  $u^e$  (достаточно сильно изменяющуюся на целевом отрезке  $[a, b]$ ). Далее путём прямого вычисления определим параметры задачи  $f$ ,  $u_a$ ,  $u_b$  такие, для которых функция  $u^e$  является точным решением задачи (1.3), (1.4).

Зададимся числом разбиений  $N$  и решим задачу для выбранным параметрами. В результате определим сеточный вектор численного решения  $\{u_i\}$ .

В качестве нормы выберем стандартное отклонение. В интегральном виде для многомерной функции  $y(\mathbf{x})$  в области  $\mathbf{x} \in D$  оно имеет вид

$$\|y(\mathbf{x})\|_2 = \sqrt{\frac{1}{|D|} \int_D y(\mathbf{x})^2 d\mathbf{x}}. \quad (1.10)$$

Упрощая до одномерного случая

$$\|y(x)\|_2 = \sqrt{\frac{1}{b-a} \int_a^b y(x)^2 dx}.$$

Вычислим этот интеграл численно на введённой ранее равномерной сетке  $\{x_i\}$ :

$$\|\{y_i\}\|_2 = \sqrt{\frac{1}{b-a} \sum_{i=0}^N w_i y_i^2},$$

где  $\{w_i\}$  – вес (или "площадь влияния")  $i$ -ого узла:

$$w_i = \begin{cases} h/2, & i = 0, N; \\ h, & i = 1, N-1, \end{cases}$$

такая что

$$\sum_{i=0}^N w_i = b - a.$$

Окончательно среднеквадратичная норма отклонения численного решения от точного запишется в виде

$$\|\{u_i - u^e(x_i)\}\|_2 = \sqrt{\frac{1}{b-a} \sum_{i=0}^N w_i (u_i - u_i^e)^2}. \quad (1.11)$$

## 1.2.2 Программная реализация

Тестовая программа для решения одномерного уравнения Пуассона реализована в файле `poisson_fdm_solve_test.cpp`.

В качестве аналитической тестовой функции используется

$$u^e = \sin(10x^2)$$

на отрезке  $x \in [0, 1]$ .

### 1.2.2.1 Функция верхнего уровня

объявлена как

```
113 TEST_CASE("Poisson 1D solver, Finite Difference Method", "[poisson1-fdm]") {
```

В программе в цикле по набору разбиений `n_cells`

```
125 for (size_t n_cells: {10, 20, 50, 100, 200, 500, 1000}){
```

создаётся решатель для тестовой задачи, использующий заданное число ячеек

```
127 TestPoisson1Worker worker(n_cells);
```

вычисляется среднеквадратичная норма отклонения численного решения от точного

```
130 double n2 = worker.solve();
```

полученное численное решение (вместе с точным) сохраняется в vtk файле

```
poisson1_n={10,20,...}.vtk
```

```
133 worker.save_vtk("poisson1_fdm_n=" + std::to_string(n_cells) + ".vtk");
```

а полученная норма печатается в консоль напротив количества ячеек

```
136 std::cout << n_cells << " " << n2 << std::endl;
```

В результате работы программы в консоли должна отобразиться таблица вида

```
--- [poisson1] ---
10 0.179124
20 0.0407822
50 0.00634718
100 0.00158055
200 0.000394747
500 6.31421e-05
1000 1.57849e-05
```

где первый столбец – это количество ячеек, а второй – полученная для этого количества ячеек норма. Нарисовав график этой таблицы в логарифмических осях подтвердим второй порядок аппроксимации (рис. 1).

Открыв один из сохранённых в процессе работы файлов vtk `poisson1_ncells=?.vtk` в paraview можно посмотреть полученные графики. В файле представлены как точное “exact”, так и численное решение “numerical” (рис. 2).

### 1.2.2.2 Детали реализации

Основная работа по решению задачи проводится в классе `TestPoisson1Worker`.

В его конструкторе происходит инициализация сетки (приватного поля класса) на отрезке  $[0, 1]$  с заданным разбиением `n_cells`:

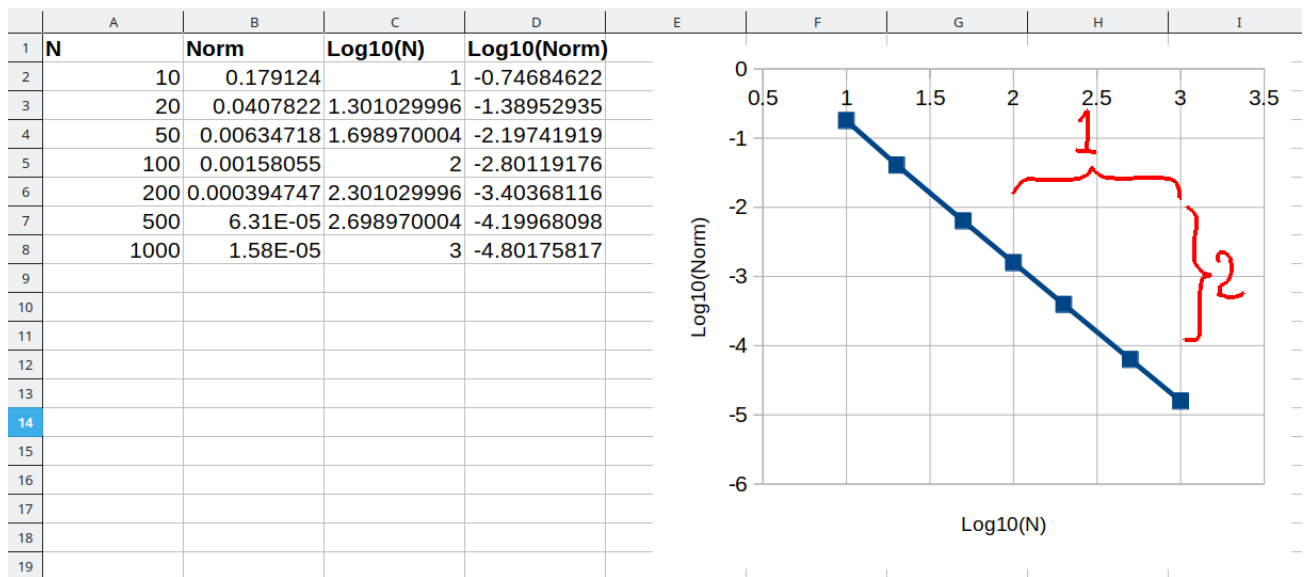


Рис. 1: Сходимость с уменьшением разбиения при решении одномерного уравнения Пуассона

```
15 class TestPoisson1Worker{
```

В методе

`solve()` производится численное решение задачи и вычисления нормы. Для этого последовательно

1. Строится матрица левой части и вектор правой части определяющей системы уравнений. Матрицы хранятся в разреженном формате CSR, удобном для последовательного чтения.
2. Вызывается решатель СЛАУ. Решение записывается в приватное поле класса `u`.
3. Вызывается функция вычисления нормы.

```
30 double solve(){
31     // 1. build SLAE
32     CsrMatrix mat = approximate_lhs();
33     std::vector<double> rhs = approximate_rhs();
34
35     // 2. solve SLAE
36     AmgcMatrixSolver solver;
37     solver.set_matrix(mat);
38     solver.solve(rhs, u_);
39
40     // 3. compute norm2
41     return compute_norm2();
42 }
```

Функции нижнего уровня (используемые в методе `solve`):



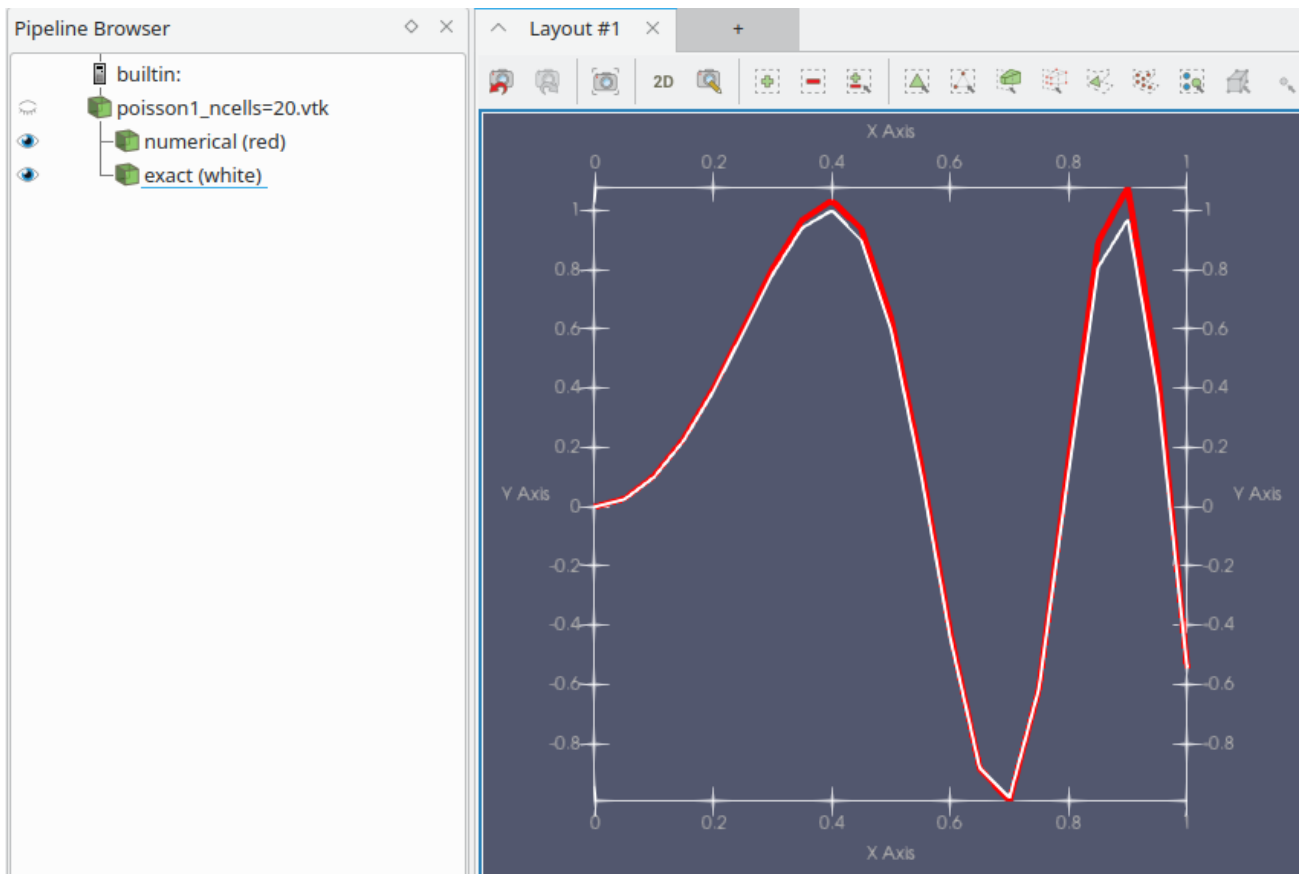


Рис. 2: Сравнение точного и численного решений уравнения Пуассона

- Сборка левой части СЛАУ. Реализует формулу (1.8). Для заполнения матрицы используется формат `cfd::LodMatrix`, удобный для непоследовательной записи, который в конце конвертируется CSR.

```

64  CsrMatrix approximate_lhs() const{
65      // constant h = x[1] - x[0]
66      double h = grid_.point(1).x - grid_.point(0).x;
67
68      // fill using 'easy-to-construct' sparse matrix format
69      LodMatrix mat(grid_.n_points());
70      mat.add_value(0, 0, 1);
71      mat.add_value(grid_.n_points()-1, grid_.n_points()-1, 1);
72      double diag = 2.0/h/h;
73      double nondiag = -1.0/h/h;
74      for (size_t i=1; i<grid_.n_points()-1; ++i){
75          mat.add_value(i, i-1, nondiag);
76          mat.add_value(i, i+1, nondiag);
77          mat.add_value(i, i, diag);
78      }
79

```

```

80     // return 'easy-to-use' sparse matrix format
81     return mat.to_csr();
82 }

```

- Сборка правой части СЛАУ. Реализует формулу (1.9).

```

84     std::vector<double> approximate_rhs() const{
85         std::vector<double> ret(grid_.n_points());
86         ret[0] = exact_solution(grid_.point(0).x);
87         ret[grid_.n_points()-1] = exact_solution(grid_.point(grid_.n_points()-1).x);
88         for (size_t i=1; i<grid_.n_points()-1; ++i){
89             ret[i] = exact_rhs(grid_.point(i).x);
90         }
91         return ret;
92     }

```

- Вычисление нормы. Реализует формулу (1.11).

```

94     double compute_norm2() const{
95         // weights
96         double h = grid_.point(1).x - grid_.point(0).x;
97         std::vector<double> w(grid_.n_points(), h);
98         w[0] = w[grid_.n_points()-1] = h/2;
99
100        // sum
101        double sum = 0;
102        for (size_t i=0; i<grid_.n_points(); ++i){
103            double diff = u_[i] - exact_solution(grid_.point(i).x);
104            sum += w[i]*diff*diff;
105        }
106
107        double len = grid_.point(grid_.n_points()-1).x - grid_.point(0).x;
108        return std::sqrt(sum / len);
109    }

```

### 1.3 Метод конечных объёмов

Будем рассматривать задачу в многомерной постановке (1.1), (1.2).

### 1.3.1 Конечнообъёмная сетка

Разобьём область численного решения на непересекающиеся подобласти  $E_i$ ,  $i = \overline{0, N-1}$ , а её границу  $\partial\Omega$  на грани  $\Gamma_s$ ,  $s = \overline{0, N^\Gamma - 1}$  (рис. 3). Введем следующие сеточные примитивы:

- $E_i$  – ячейка сетки,
- $\Gamma_s$  – граничная грань,
- $\mathbf{c}_i$  – центр (масс) ячейки,
- $\mathbf{g}_s$  – центр (масс) грани  $\Gamma_{is}$ .
- $\gamma_{ij}$  – внутренняя грань между  $i$ -ой и  $j$ -ой ячейками,

Будем считать, что ячейки сетки выпуклые, а грани – плоские.

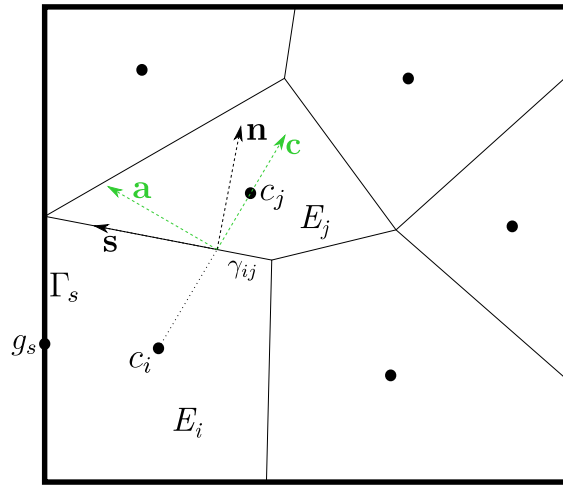


Рис. 3: Конечнообъёмная сетка

### 1.3.2 Конечнообъёмная аппроксимация

Проинтегрируем исходное уравнение по одной из подобластей  $E_i$ :

$$-\int_{E_i} \nabla^2 u \, ds = \int_{E_i} f \, d\mathbf{x}.$$

К интегралу в левой части применим формулу интегрирования по частям (B.13). Получим

$$-\int_{\partial E_i} \frac{\partial u}{\partial n} \, ds = \int_{E_i} f \, d\mathbf{x}. \quad (1.12)$$

Здесь  $\partial E_i$  – совокупность всех границ подобласти  $E_i$ , а  $\mathbf{n}$  – внешняя к подобласти нормаль.

Граница ячейки  $E_i$  состоит из внутренних граней  $\gamma_{ij}$  (индекс  $j$  здесь соответствует индексу соседней ячейки) и инцидентных ей граней  $\Gamma_s$ , лежащих на внешней границе расчётной области  $\Omega$ . Тогда

интеграл по общей границе ячейки распишется через сумму интегралов по плоским поверхностям

$$\int_{\partial E_i} \frac{\partial u}{\partial n} ds = \sum_j \int_{\gamma_{ij}} \frac{\partial u}{\partial n} ds + \sum_s \int_{\Gamma_s} \frac{\partial u}{\partial n} ds.$$

Аппроксимируем производную  $\partial u / \partial n$  на каждой из граней константой. Тогда её можно вынести из под интегралов и предыдущее выражение записать в виде

$$\int_{\partial E_i} \frac{\partial u}{\partial n} ds \approx \sum_j |\gamma_{ij}| \left( \frac{\partial u}{\partial n} \right)_{\gamma_{ij}} + \sum_s |\Gamma_s| \left( \frac{\partial u}{\partial n} \right)_{\Gamma_{is}} \quad (1.13)$$

Аналогично, анализируя интеграл правой части (1.12), приблизим значение функции правой части  $f$  внутри элемента  $E_i$  константой  $f_i$ , которую отнесём к центру элемента. Тогда

$$\int_{E_i} f d\mathbf{x} \approx f_i |E_i|. \quad (1.14)$$

Сеточный вектор  $\{f_i\}$  – есть конечнообъёмная аппроксимация функции  $f(\mathbf{x})$  на конечнообъёмную сетку. Значения  $f_i$  при аппроксимации чаще всего находятся как значения в центрах элементов

$$f_i = f(\mathbf{c}_i).$$

Хотя иногда может быть использовано и другое определение, следующее из (1.14):

$$f_i = \frac{1}{|E_i|} \int_{E_i} f(\mathbf{x}) d\mathbf{x}.$$

### 1.3.2.1 Обработка внутренних граней

Для начала будем рассматривать сетки, в которых вектора  $\mathbf{c}$ , соединяющие центры ячеек (зедёные вектора на рис. 3), коллинеарны (или почти коллинеарны) нормальям к граням  $\mathbf{n}$ . В этом случае производную искомой функции по нормали к грани можно записать в виде

$$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial c}.$$

Далее определим значения функции  $u$  в точках  $c_i, c_j$  как  $u_i, u_j$ . Тогда значение производной  $\partial u / \partial n$  на внутренней грани конечного объёма может быть приближена конечной разностью

$$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial c} \approx \frac{u_j - u_i}{h_{ij}}, \quad h_{ij} = |\mathbf{c}_j - \mathbf{c}_i|. \quad (1.15)$$

Определим *perbi* (perpendicular-bisector) сетки как сетки, удовлетворяющие следующим свойствам

- линии, соединяющие центры двух соседних ячеек, перпендикулярны грани между этими ячейками;
- внутренние грани делят линии, соединяющие центры соседних ячеек, пополам.

Очевидно, что равномерная структурированная сетка удовлетворяет этим свойствам. Для построения неструктурированных ребѐ-сеток используют алгоритмы построения ячеек Вороного. Для ребѐ-сеток разностная схема (1.15) является симметричной разностью и, поэтому, имеет второй порядок аппроксимации.

### 1.3.2.2 Учёт граничных условий первого рода

Для вычисления второго слагаемого в правой части (1.13) следует расписать значение нормальной к границе производной вида

$$\left( \frac{\partial u}{\partial n} \right)_{\Gamma_s}.$$

Это делается с помощью граничных условий.

Пусть в центре  $\mathbf{g}_s$  грани  $\Gamma_s$  задано значение искомой функции (1.2):

$$u(\mathbf{g}_s) = u_s^\Gamma. \quad (1.16)$$

Аппроксимацию производных будем проводить из тех же соображений, которые использовали при анализе внутренних граней. Только вместо центра соседнего элемента  $c_j$  будем использовать центр грани  $g_s$ . В первом приближении, отбрасывая касательные производные, придѐм к формуле аналогичной (1.15):

$$\frac{\partial u}{\partial n} \approx \frac{u_s^\Gamma - u_i}{h_{is}}, \quad h_{is} = |\mathbf{g}_s - \mathbf{c}_i|. \quad (1.17)$$

### 1.3.3 Одномерный случай

Рассмотрим результат конечнообъёмной аппроксимации задачи (1.1) в одномерном случае (1.3) на равномерной сетке с шагом  $h$  (рис. 4).

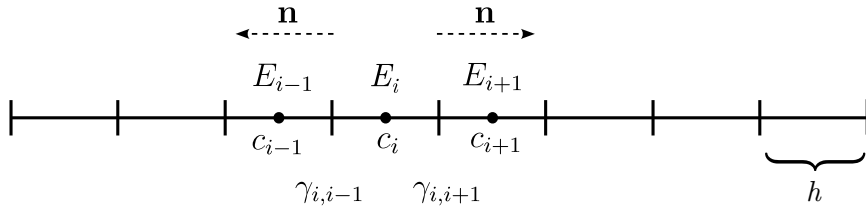


Рис. 4: Одномерная конечнообъёмная сетка

У внутренней ячейки  $i$  есть две границы:  $\gamma_{i,i-1}$  и  $\gamma_{i,i+1}$ . Нормали по этим границам аппроксимируются по формулам (1.15):

$$\begin{aligned} \gamma_{i,i-1} : \quad \frac{\partial u}{\partial n} &= \frac{u_{i-1} - u_i}{h} \\ \gamma_{i,i+1} : \quad \frac{\partial u}{\partial n} &= \frac{u_{i+1} - u_i}{h} \end{aligned}$$

Объѐм ячейки в одномерном случае равен её длине  $h$ . Площадь грани следует положить единице с тем, чтобы

$$|E_i| = |\gamma|h = h.$$

Тогда, подставляя эти значения в (1.12), получим знакомую конечноразностную схему аппроксимации уравнения Пуассона

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h} = f_i h,$$

которая имеет второй порядок точности. Разница с методом конечных разностей здесь состоит в том, что значения сеточных векторов  $\{u\}$ ,  $\{f\}$  здесь приписаны к центрам ячеек, а не к их узлам. Это отличие проявит себя в аппроксимации граничных условий. Так, если на левой границе  $x = a$  задано условие первого рода, то соответствующее уравнение согласно (1.17) примет вид

$$-\frac{u_a^\Gamma - u_0}{h/2} - \frac{u_1 - u_0}{h} = f_0 h.$$

В методе конечных разностей это условие выразилось бы в виде  $u_0 = u_a^\Gamma$ .

### 1.3.4 Сборка системы линейных уравнений

Подставим все полученные аппроксимации (1.15), (1.17) в уравнение (1.12). Получим  $i$ -ое уравнение искомой системы уравнений относительно неизвестных  $u_i$ :

$$-\sum_{j \in J_i} \frac{|\gamma_{ij}|}{h_{ij}} (u_j - u_i) - \sum_{s \in I_i} \frac{|\Gamma_s|}{h_{is}} (u_s^\Gamma - u_i) = f_i |E_i|.$$

Введены следующие обозначения множества индексов:  $J_i$  – индексы ячеек, соседних (имеющих общую грань) с текущей ячейкой  $i$ ,  $I_i$  – индексы граничных граней первого рода, инцидентных ячейке  $E_i$ . Здесь первое слагаемое в левой части отвечает за потоки через внутренние границы, второе – граничные условия первого рода. Далее перенесём все известные значения в правую часть и окончательно получим линейное уравнение для  $i$ -го конечного объёма:

$$\sum_{j \in J_i} \frac{|\gamma_{ij}|}{h_{ij}} (u_i - u_j) + \sum_{s \in I_i} \frac{|\Gamma_s|}{h_{is}} u_i = f_i |E_i| + \sum_{s \in I_i} \frac{|\Gamma_s|}{h_{is}} u_s^\Gamma \quad (1.18)$$

Таким образом мы получили систему из  $N$  (по количеству подобластей) линейных уравнений относительно неизвестного сеточного вектора  $\{u_i\}$

$$Au = b.$$

Полученные в результате сборочных процедур матрицы являются разреженными – то есть большинство их элементов равно нулю. Полное хранение таких матриц в памяти невозможно, поэтому применяют специальные процедуры разреженного хранения (см. п. C.2).

Ниже приведён псевдокод для сборки СЛАУ. Перед началом процедур сборки левую правую часть нужно инициализировать нулями.

#### 1.3.4.1 Алгоритм сборки в цикле по ячейкам

Матрицу  $A$  и правую часть  $b$  системы (1.18) можно собирать в цикле по ячейкам: строка за строкой. Такой алгоритм выглядел бы следующим образом

```

for  $i = \overline{0, N-1}$            – цикл по строкам СЛАУ
     $b_i = |E_i|f_i$ 
    for  $j \in \text{nei}(i)$          – цикл по ячейкам, соседним с ячейкой  $i$ 
         $v = |\gamma_{ij}|/h_{ij}$ 
         $A_{ii} += v$ 
         $A_{ij} -= v$ 
    endfor
    for  $s \in \text{bnd1}(i)$        – цикл по граням ячейки  $i$  с условиями первого рода
         $v = |\Gamma_{is}|/h_{is}$ 
         $A_{ii} += v$ 
         $b_i += u_i^\Gamma v$ 
    endfor
endfor

```

Первым недостатком такого алгоритма является наличие вложенных циклов. Во-вторых, коэффициент, отвечающий за поток через внутреннюю грань  $\gamma_{ij}$ , равный  $|\gamma_{ij}|/h_{ij}$  в таком алгоритме будет учитываться дважды: в строке  $i$  и в строке  $j$ .

#### 1.3.4.2 Алгоритм сборки в цикле по граням

Вместо общего цикла по ячейкам, будем использовать цикл по граням. В таком цикле коэффициенты потоков будут вычисляться один раз и вставляться сразу в две строки матрицы, соответствующие соседним с гранью ячейкам. Вложенных циклов в такой постановке удаётся избежать, потому что у грани есть только две соседние ячейки (в то время как у ячейки может быть произвольное количество соседних граней).

Разделим все грани на исходной сетки на внутренние и граничные (отдельный набор для каждого вида граничных условий). Тогда для внутренних граней можно записать

```

for  $s \in \text{internal}$          – цикл по внутренним граням
     $i, j = \text{nei\_cells}(s)$    – две ячейки, соседние с текущей гранью
     $v = |\gamma_{ij}|/h_{ij}$ 
     $A_{ii} += v; \quad A_{jj} += v$  – диагональные коэффициенты матрицы
     $A_{ij} -= v; \quad A_{ji} -= v$  – внедиагональные коэффициенты матрицы
endfor

```

(1.19)

Граничные условия учитываются в отдельных циклах. Здесь будем учитывать, что у грани, принад-

лежащей границе области, есть только одна соседняя ячейка. Условия первого рода:

$$\begin{aligned}
 &\mathbf{for} \ s \in \text{bnd1} && \text{-- грани с условиями первого рода} \\
 &\quad i = \text{nei\_cells}(s) && \text{-- соседняя с граничной гранью ячейка} \\
 &\quad v = |\Gamma_{is}|/h_{is} \\
 &\quad A_{ii} += v \\
 &\quad b_i += u_i^\Gamma v \\
 &\mathbf{endfor}
 \end{aligned} \tag{1.20}$$

Первое слагаемое в правой части (1.18) учтём отдельным циклом:

$$\begin{aligned}
 &\mathbf{for} \ i = \overline{0, N-1} && \text{-- цикл по ячейкам} \\
 &\quad b_i += |E_i| f_i \\
 &\mathbf{endfor}
 \end{aligned} \tag{1.21}$$



## **А   Задания для самостоятельной работы**

## А.1 Лекция 2 (20.09.25) МКО для решения уравнения Пуассона

Теория: п. 1.3

В тесте `poisson1-fvm` из файла `poisson_fvm_test.cpp` реализовано решение одномерного уравнения Пуассона с граничными условиями первого рода. Проводится расчёт на сгущающихся сетках с количеством ячеек от 10 до 1000 и рассчитываются среднеквадратичные нормы отклонения полученного численного решения от точного. Решения сохраняются в vtk-файлы `poisson1_fvm_n={}.vtk`. Отталкиваясь от этой реализации необходимо:

1. написать аналогичный тест для двумерного уравнения,
2. провести серию расчётов на сгущающихся сетках разных типов (структурированных, `reb1` и скошенных),
3. визуализировать в Paraview полученное на этих сетках решение,
4. построить графики сеточной сходимости решения и определить порядок аппроксимации метода,
5. в тестовой программе производится сборка в цикле по ячейкам (п. 1.3.4.1). Следует переписать процедуру сборки в циклах по граням (п. 1.3.4.2) и убедиться в идентичности результатов.

### А.1.1 Работа с сетками

Все сетки в программе наследуются от абстрактного класса

`IGrid`. Необходимые для работы с сетками таблицы узлов, свойств и связности доступны как виртуальные методы этого класса и объявлены в заголовочном файле

`grid/i_grid.hpp`. Например

- `Point IGrid::point(size_t ipoint)` – получить координату  $i$ -ой точки,
- `Point IGrid::face_area(size_t iface)` – площадь  $i$ -ой грани,
- `Point IGrid::tab_cell_face(size_t icell)` – получить список индексов граней для  $i$ -ой ячейки.

Грани (внутренние и граничные) пронумерованы сквозным образом. Координаты точек всегда трёхмерные. Для двумерных и одномерных задач “лишние” координаты приравниваются нулю.

С помощью метода

`IGrid::save_vtk` сетка может быть экспортирована в vtk формат и просмотрена в Paraview. В п. D.6 приведены некоторые приёмы визуализации численного решения.

Для построения двумерных сеток необходимо использовать класс `Grid2`:

```
// сетка 10x10 в единичном квадрате
auto grid = std::make_shared<RegularGrid2D>(0, 0, 1, 1, 10, 10);
```

Неструктурированные сетки должны быть прочитаны из файла:

```
// Читаем сетку из файла /app/test_data/pebigrid.vtk
std::string fn = test_directory_file("pebigrid.vtk");
UnstructuredGrid2D grid = UnstructuredGrid2D::vtk_read(fn);
```

Строить неструктурированные сетки следует с помощью утилиты `hybmesh`. В папке `test_data` корневой директории репозитория лежат скрипты построения сеток:

- `pebigrid.py` – pebi-сетка,
- `tetragrid.py` – сетка, состоящая из произвольных (скошенных) трех- и четырехугольников.

Инструкции по запуску этих скриптов смотри п. D.7. Эти скрипты строят равномерную неструктурированную сетку в единичном квадрате и записывают её в файл `vtk`, который впоследствии можно загрузить в расчётную программу. В каждом из скриптов есть параметр `N`, означающий примерное количество ячеек в итоговой сетке. Меняя его значение можно строить сетки разного разрешения.

Для загрузки построенной сетки в решатель необходимо файл с сеткой поместить в каталог `test_data` и далее загрузить её в класс `UnstructuredGrid2D`.

### A.1.2 Тестовая задача

Для тестирования двумерной задачи следует использовать двумерное точное решение. Например,

```
double exact_solution(Point p) const override{
    double x = p.x();
    double y = p.y();
    return cos(10*x*x)*sin(10*y) + sin(10*x*x)*cos(10*x);
}
```

Для вычисления правой части (функции `exact_rhs`) нужно подставить точное решение в исходное уравнение (1.1).

### A.1.3 График сходимости

График сеточной сходимости следует строить по аналогии с графиком на рис. 1. в логарифмических осях, где по оси абсцисс отложено разбиение, а по оси ординат – норма. Где разбиение – это число, обратное к характерному линейному размеру ячейки. Его размерность –  $L^{-1}$ . Для получения корректного порядка аппроксимации в двух- и трёхмерных задачах следует внимательно отнестись к вычислению этого параметра. Для двумерной (трёхмерной) ячейки характерный размер можно определить как корень квадратный (кубический) от объёма ячейки.

### A.1.4 Рекомендации к программированию

Свои программы следует оформлять в виде отдельных тестов (вместо того, чтобы модифицировать существующие). Желательно, после того как программа заработает, сразу оставить несколько базовых `CHECK` проверок и сделать локальный коммит, чтобы впоследствии легко распознавать и исправлять внесённые в дальнейшей работе ошибки.

При написании новых тестов следует переиспользовать уже написанный код, избегая копирования. Для этого необходимо оформлять повторяющийся код в виде отдельных процедур и пользоваться механизмами наследования классов.

## В    Формулы и обозначения

## В.1 Векторы

### В.1.1 Обозначение

Геометрические вектора обозначаются жирным шрифтом  $\mathbf{v}$ . Скалярные координаты вектора – через нижний индекс с обозначением оси координат:  $(v_x, v_y, v_z)$ . Если вектор  $\mathbf{u}$  – вектор скорости, то его декартовы координаты имеют специальное обозначение  $\mathbf{u} = (u, v, w)$ . Единичные вектора, соответствующие осям координат, обозначаются знаком  $\hat{\cdot}$ :  $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ . Координатные векторы обозначаются по символу первой оси. Например,  $\mathbf{x} = (x, y, z)$  или  $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ .

Операции в векторах имеют следующее обозначение (расписывая в декартовых координатах):

- Умножение на скалярную функцию

$$f\mathbf{u} = (fu_x)\hat{\mathbf{x}} + (fu_y)\hat{\mathbf{y}} + (fu_z)\hat{\mathbf{z}}; \quad (\text{B.1})$$

- Скалярное произведение

$$\mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z; \quad (\text{B.2})$$

- Векторное произведение

$$\mathbf{u} \times \mathbf{v} = \begin{vmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} = (u_y v_z - u_z v_y)\hat{\mathbf{x}} - (u_x v_z - u_z v_x)\hat{\mathbf{y}} + (u_x v_y - u_y v_x)\hat{\mathbf{z}}. \quad (\text{B.3})$$

В двумерном случае можно считать, что  $u_z = v_z = 0$ . Тогда результатом векторного произведения согласно (B.3) будет вектор, направленный перпендикулярно плоскости  $xy$ :

$$\mathbf{u} \times \mathbf{v} = (u_x v_y - u_y v_x)\hat{\mathbf{z}}.$$

При работе с двумерными задачами, где ось  $\mathbf{z}$  отсутствует, обычно результатом векторного произведения считают скаляр

$$2D : \mathbf{u} \times \mathbf{v} = u_x v_y - u_y v_x. \quad (\text{B.4})$$

Геометрический смысл этого скаляра: площадь параллелограмма, построенного на векторах  $\mathbf{u}$  и  $\mathbf{v}$ .

### В.1.2 Набла–нотация

Символ  $\nabla$  – есть псевдовектор, который выражает покоординатные производные. Для декартовой системы координат  $(x, y, z)$  он запишется в виде

$$\nabla = \left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right).$$

В радиальной  $(r, \phi, z)$ :

$$\nabla = \left( \frac{\partial}{\partial r}, \frac{1}{r} \frac{\partial}{\partial \phi}, \frac{\partial}{\partial z} \right).$$

В цилиндрической  $(r, \theta, \phi)$ :

$$\nabla = \left( \frac{\partial}{\partial r}, \frac{1}{r} \frac{\partial}{\partial \theta}, \frac{1}{r \sin \theta} \frac{\partial}{\partial \phi} \right).$$

Удобство записи дифференциальных выражений с использованием  $\nabla$  заключается в независимости записи от вида системы координат. Но если требуется обозначить производную по конкретной координате, то, по аналогии с обычными векторами, это делается через нижний индекс:

$$\nabla_n f = \frac{\partial f}{\partial n}.$$

Для этого символа справедливы все векторные операции, описанные ранее. Так, применение  $\nabla$  к скалярной функции аналогично умножению вектора на скаляр (B.1) (здесь и далее приводятся покомпонентные выражения для декартовой системы):

$$\nabla f = (\nabla_x f, \nabla_y f, \nabla_z f) = \frac{\partial f}{\partial x} \hat{\mathbf{x}} + \frac{\partial f}{\partial y} \hat{\mathbf{y}} + \frac{\partial f}{\partial z} \hat{\mathbf{z}}. \quad (\text{B.5})$$

Результатом этой операции является вектор.

Скалярное умножение  $\nabla$  на вектор  $\mathbf{v}$  по аналогии с (B.2) – есть дивергенция:

$$\nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \quad (\text{B.6})$$

результат которой – скалярная функция.

Двойное применение  $\nabla$  к скалярной функции – это оператор Лапласа:

$$\nabla \cdot \nabla f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \quad (\text{B.7})$$

Ротор – аналог векторного умножения (B.3):

$$\nabla \times \mathbf{v} = \begin{vmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ \nabla_x & \nabla_y & \nabla_z \\ v_x & v_y & v_z \end{vmatrix} = \left( \frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \right) \hat{\mathbf{x}} - \left( \frac{\partial v_z}{\partial x} - \frac{\partial v_x}{\partial z} \right) \hat{\mathbf{y}} + \left( \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) \hat{\mathbf{z}}. \quad (\text{B.8})$$

## В.2 Интегрирование

### В.2.1 Формула Гаусса–Остроградского

Формула Гаусса–Остроградского, связывающая интегрирование по объёму  $E$  с интегрированием по границе этого объёма  $\Gamma$ , для векторного поля  $\mathbf{v}$  имеет вид

$$\int_E \nabla \cdot \mathbf{v} d\mathbf{x} = \int_{\Gamma} v_n ds, \quad (\text{В.9})$$

где  $\mathbf{n}$  – внешняя по отношению к области  $E$  нормаль. Смысл этой формулы можно проиллюстрировать на одномерном примере. Пусть одномерное векторное поле  $v_x = f(x)$  на отрезке  $E = [a, b]$  задано функцией, представленной на рис. 5. Разобьём область на  $N = 3$  равномерных подобласти

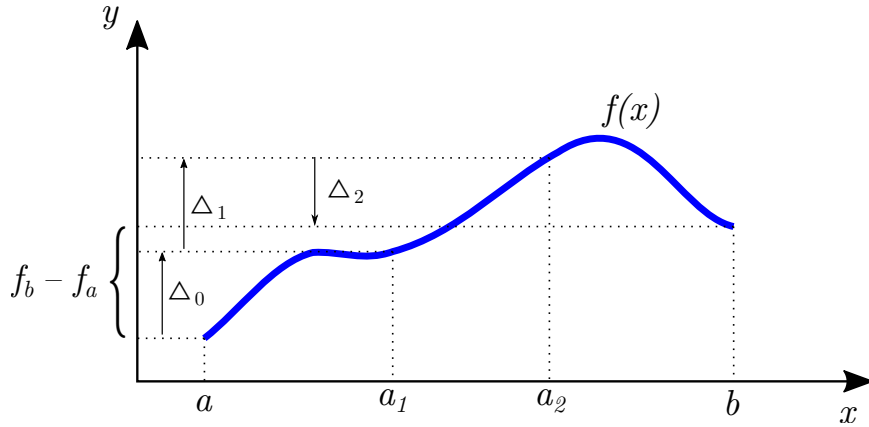


Рис. 5: Формула Гаусса–Остроградского в одномерном случае

длины  $h$ . Тогда расписывая интеграл как сумму, а производную через конечную разность, получим

$$\int_E \frac{\partial f}{\partial x} dx \approx \sum_{i=0}^2 h \left( \frac{\partial f}{\partial x} \right)_{i+\frac{1}{2}} \approx \sum_{i=0}^2 (f_{i+1} - f_i) = \Delta_0 + \Delta_1 + \Delta_2 = f_b - f_a.$$

Очевидно что, при устремлении  $N \rightarrow \infty$  правая часть предыдущего выражения не изменится. То есть, сумма всех изменений функции в области есть изменение функции по её границам:

$$\int_a^b \frac{\partial f}{\partial x} dx = f(b) - f(a).$$

А формула (В.9) – есть многомерное обобщение этого выражения.

### В.2.2 Интегрирование по частям

Подставив в (В.9)  $\mathbf{v} = f\mathbf{u}$ , где  $f$  – некоторая скалярная функция, и расписав дивергенцию в виде

$$\nabla \cdot (f\mathbf{u}) = f\nabla \cdot \mathbf{u} + \mathbf{u} \cdot \nabla f$$



получим формулу интегрирования по частям

$$\int_E \mathbf{u} \cdot \nabla f \, d\mathbf{x} = \int_{\Gamma} f u_n \, ds - \int_E f \nabla \cdot \mathbf{u} \, d\mathbf{x} \quad (\text{B.10})$$

Распишем некоторые частные случаи для формулы (B.10). Для  $\mathbf{u} = (n_x, 0, 0)$  получим

$$\int_E \frac{\partial f}{\partial x} \, d\mathbf{x} = \int_{\Gamma} f \cos(\widehat{\mathbf{n}}, \mathbf{x}) \, ds \quad (\text{B.11})$$

При  $\mathbf{u} = \nabla g$

$$\int_E f (\nabla^2 g) \, d\mathbf{x} = \int_{\Gamma} f \frac{\partial g}{\partial n} \, ds - \int_E \nabla f \cdot \nabla g \, d\mathbf{x} \quad (\text{B.12})$$

При  $f = 1$  и  $\mathbf{u} = \nabla g$

$$\int_E \nabla^2 g \, d\mathbf{x} = \int_{\Gamma} \frac{\partial g}{\partial n} \, ds \quad (\text{B.13})$$

### **В.2.3 Численное интегрирование в заданной области**

Квадратурная формула

$$\int_E f(\mathbf{x}) \, d\mathbf{x} = \sum_{i=0}^{N-1} w_i f(\mathbf{x}_i) \quad (\text{B.14})$$

Она определяется заданием узлов интегрирования  $\mathbf{x}_i$  и соответствующих весов  $w_i$ .

## В.3 Интерполяционные полиномы

### В.3.1 Многочлен Лагранжа

#### В.3.1.1 Узловые базисные функции

Рассмотрим функцию  $f(\xi)$ , заданную в области  $D$ . Внутри этой области зададим  $N$  узловых точек  $\xi_i, i = \overline{0, N-1}$ . Приближение функции  $f$  будем искать в виде

$$f(\xi) \approx \sum_{i=0}^{N-1} f_i \phi_i(\xi), \quad (\text{B.15})$$

где  $f_i = f(\xi_i)$ ,  $\phi_i$  – узловая базисная функция. Потребуем, чтобы это выражение выполнялось точно для всех заданных узлов интерполяции  $\xi = \xi_i$ . Тогда, исходя из определения (B.15), запишем условие на узловую базисную функцию

$$\phi_i(\xi_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \quad (\text{B.16})$$

Дополнительно потребуем, чтобы формула (B.15) была точной для постоянных функций

$$f(\xi) = \text{const} \quad \Rightarrow \quad f_i = \text{const}.$$

Тогда для любого  $\xi$  должно выполняться условие

$$\sum_{i=0}^{N-1} \phi_i(\xi) = 1, \quad \xi \in D. \quad (\text{B.17})$$

Задача построения интерполяционной функции состоит в конкретном определении узловых базисов  $\phi_i(\xi)$  по заданному набору узловых точек  $\xi_i$  и значениям функции в них  $f_i$ . Будем искать базисы в виде многочленов вида

$$\phi_i(\xi) = \sum_a A_i^{(a)} \xi^a = A_i^{(0)} + A_i^{(1)} \xi + A_i^{(2)} \xi^2 + \dots, \quad i = \overline{0, N-1}. \quad (\text{B.18})$$

Определять коэффициенты  $A_i^{(a)}$  будем из условий (B.16), которое даёт  $N$  линейных уравнений относительно неизвестных  $A_i^{(a)}$  для каждого  $i = \overline{0, N-1}$ . Таким образом, в выражениях (B.18) должно быть ровно  $N$  слагаемых. Будем использовать последовательный набор степеней:  $a = \overline{0, N-1}$ . Выпишем систему линейных уравнений для 0-ой базисной функции

$$\begin{aligned} \phi_0(\xi_0) &= A_0^{(0)} + A_0^{(1)} \xi_0 + A_0^{(2)} \xi_0^2 + A_0^{(3)} \xi_0^3 + \dots = 1, \\ \phi_0(\xi_1) &= A_0^{(0)} + A_0^{(1)} \xi_1 + A_0^{(2)} \xi_1^2 + A_0^{(3)} \xi_1^3 + \dots = 0, \\ \phi_0(\xi_2) &= A_0^{(0)} + A_0^{(1)} \xi_2 + A_0^{(2)} \xi_2^2 + A_0^{(3)} \xi_2^3 + \dots = 0, \\ &\dots \end{aligned}$$

или в матричном виде

$$\begin{pmatrix} 1 & \xi_0 & \xi_0^2 & \xi_0^3 & \dots \\ 1 & \xi_1 & \xi_1^2 & \xi_1^3 & \dots \\ 1 & \xi_2 & \xi_2^2 & \xi_2^3 & \dots \\ 1 & \xi_3 & \xi_3^2 & \xi_3^3 & \dots \\ \dots & & & & \end{pmatrix} \begin{pmatrix} A_0^{(0)} \\ A_0^{(1)} \\ A_0^{(2)} \\ A_0^{(3)} \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

Записывая аналогичные выражения для остальных базисных функций, получим систему матричных уравнений вида  $CA = E$ :

$$\begin{pmatrix} 1 & \xi_0 & \xi_0^2 & \xi_0^3 & \dots \\ 1 & \xi_1 & \xi_1^2 & \xi_1^3 & \dots \\ 1 & \xi_2 & \xi_2^2 & \xi_2^3 & \dots \\ 1 & \xi_3 & \xi_3^2 & \xi_3^3 & \dots \\ \dots & & & & \end{pmatrix} \begin{pmatrix} A_0^{(0)} & A_1^{(0)} & A_2^{(0)} & A_3^{(0)} & \dots \\ A_0^{(1)} & A_1^{(1)} & A_2^{(1)} & A_3^{(1)} & \dots \\ A_0^{(2)} & A_1^{(2)} & A_2^{(2)} & A_3^{(2)} & \dots \\ A_0^{(3)} & A_1^{(3)} & A_2^{(3)} & A_3^{(3)} & \dots \\ \vdots & & & & \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ \vdots & & & & \end{pmatrix}$$

Отсюда матрица неизвестных коэффициентов  $A$  определится как

$$A = C^{-1} = \begin{pmatrix} 1 & \xi_0 & \xi_0^2 & \xi_0^3 & \dots \\ 1 & \xi_1 & \xi_1^2 & \xi_1^3 & \dots \\ 1 & \xi_2 & \xi_2^2 & \xi_2^3 & \dots \\ 1 & \xi_3 & \xi_3^2 & \xi_3^3 & \dots \\ \dots & & & & \end{pmatrix}^{-1}. \quad (\text{B.19})$$

Подставляя полином (B.18) в условие согласованности (B.17), получим требование

$$\sum_{i=0}^{N-1} A_i^{(a)} = \begin{cases} 1, & a = 0, \\ 0, & a = \overline{1, N-1}. \end{cases}$$

То есть сумма всех свободных членов в интерполяционных полиномах должна быть равна единице, а сумма коэффициентов при остальных степенях – нулю. Можно показать, что это свойство выполняется для любой матрицы  $A = C^{-1}$ , в случае, если первый столбец матрицы  $C$  состоит из единиц. То есть условие согласованности требует наличие свободного члена с интерполяционным полиномом.

### В.3.1.2 Интерполяция в параметрическом отрезке

Будем рассматривать область интерполяции  $D = [-1, 1]$ . В качестве первых двух узлов интерполяции возьмем границы области:  $\xi_0 = -1$ ,  $\xi_1 = 1$ .

**Линейный базис** Будем искать интерполяционный базис в виде

$$\phi_i(\xi) = A_i^{(0)} + A_i^{(1)}\xi.$$

на основе двух условий:

$$\phi_i(-1) = A_i^{(0)} - A_i^{(1)} = \delta_{0i}, \quad \phi_i(1) = A_i^{(0)} + A_i^{(1)} \delta_{1i}.$$

Составим матрицу  $C$ , записав эти условия в матричном виде

$$C = \left( \begin{array}{c|cc} & A^{(0)} & A^{(1)} \\ \hline \phi(-1) & 1 & -1 \\ \phi(1) & 1 & 1 \end{array} \right)$$

и, согласно (B.19), найдём матрицу коэффициентов

$$A = \begin{pmatrix} A_0^{(0)} & A_1^{(0)} \\ A_0^{(1)} & A_1^{(1)} \end{pmatrix} = C^{-1} = \begin{pmatrix} & \phi_0 & \phi_1 \\ \hline 1 & \frac{1}{2} & \frac{1}{2} \\ \xi & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}.$$

Отсюда узловые базисные функции примут вид (рис. 6)

$$\begin{aligned} \phi_0(\xi) &= \frac{1-\xi}{2}, \\ \phi_1(\xi) &= \frac{1+\xi}{2}. \end{aligned} \tag{B.20}$$

Окончательно интерполяционная функция из определения (B.15) примет вид

$$f(\xi) \approx \frac{1-\xi}{2}f(-1) + \frac{1+\xi}{2}f(1).$$

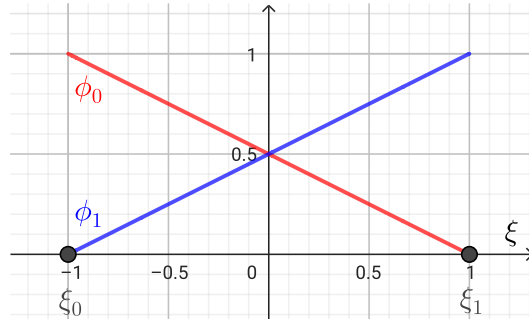


Рис. 6: Линейный базис в параметрическом отрезке

**Квадратичный базис** Будем искать интерполяционный базис в виде

$$\phi_i(\xi) = A_i^{(0)} + A_i^{(1)}\xi + A_i^{(2)}\xi^2.$$

По сравнению с линейным случаем, в форму базиса добавился ещё один неизвестный коэффициент  $A_i^{(2)}$ , поэтому в набор условий (B.16) требуется ещё одно уравнение (ещё одна узловая точка). Поче-

стим её в центр параметрического сегмента  $\xi_2 = 0$ . Далее будем действовать по аналогии с линейным случаем:

$$C = \left( \begin{array}{c|ccc} & A^{(0)} & A^{(1)} & A^{(2)} \\ \hline \phi(-1) & 1 & -1 & 1 \\ \phi(1) & 1 & 1 & 1 \\ \phi(0) & 1 & 0 & 0 \end{array} \right) \Rightarrow A = C^{-1} = \left( \begin{array}{c|ccc} & \phi_0 & \phi_1 & \phi_2 \\ \hline 1 & 0 & 0 & 1 \\ \xi & -\frac{1}{2} & \frac{1}{2} & 0 \\ \xi^2 & \frac{1}{2} & \frac{1}{2} & -1 \end{array} \right).$$

Узловые базисные функции для квадратичной интерполяции примут вид (рис. 7)

$$\begin{aligned} \phi_0(\xi) &= \frac{\xi^2 - \xi}{2}, \\ \phi_1(\xi) &= \frac{\xi^2 + \xi}{2}, \\ \phi_2(\xi) &= 1 - \xi^2. \end{aligned} \tag{B.21}$$

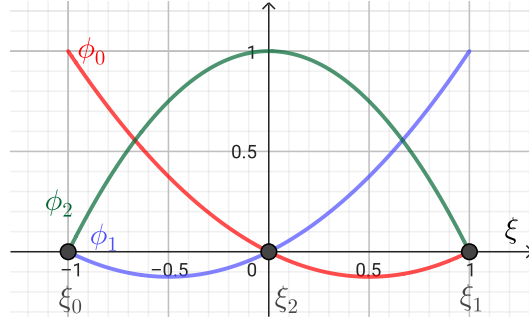


Рис. 7: Квадратичный базис в параметрическом отрезке

**Кубический базис** Интерполяционный базис будет иметь вид

$$\phi_i(\xi) = A_i^{(0)} + A_i^{(1)}\xi + A_i^{(2)}\xi^2 + A_i^{(3)}\xi^3.$$

Для нахождения четырёх коэффициентов нам понадобится четыре узла интерполяции. Две из них – это границы параметрического отрезка. Остальные две разместим так, чтобы разбить отрезок на равные интервалы:  $\xi_2 = -\frac{1}{3}$ ,  $\xi_3 = \frac{1}{3}$ . Далее вычислим матрицу коэффициентов:

$$C = \left( \begin{array}{c|cccc} & A^{(0)} & A^{(1)} & A^{(2)} & A^{(3)} \\ \hline \phi(-1) & 1 & -1 & 1 & -1 \\ \phi(1) & 1 & 1 & 1 & 1 \\ \phi(-\frac{1}{3}) & 1 & -\frac{1}{3} & \frac{1}{9} & -\frac{1}{27} \\ \phi(\frac{1}{3}) & 1 & \frac{1}{3} & \frac{1}{9} & \frac{1}{27} \end{array} \right) \Rightarrow A = C^{-1} = \frac{1}{16} \left( \begin{array}{c|cccc} & \phi_0 & \phi_1 & \phi_2 & \phi_3 \\ \hline 1 & -1 & -1 & 9 & 9 \\ \xi & 1 & -1 & -27 & 27 \\ \xi^2 & 9 & 9 & -9 & -9 \\ \xi^3 & -9 & 9 & 27 & -27 \end{array} \right)$$

Узловые базисные функции для квадратичной интерполяции примут вид (рис. 8)

$$\begin{aligned}
 \phi_0(\xi) &= \frac{1}{16} (-1 + \xi + 9\xi^2 - 9\xi^3), \\
 \phi_1(\xi) &= \frac{1}{16} (-1 - \xi + 9\xi^2 + 9\xi^3), \\
 \phi_2(\xi) &= \frac{1}{16} (9 - 27\xi - 9\xi^2 + 27\xi^3), \\
 \phi_3(\xi) &= \frac{1}{16} (9 + 27\xi - 9\xi^2 - 27\xi^3),
 \end{aligned} \tag{B.22}$$

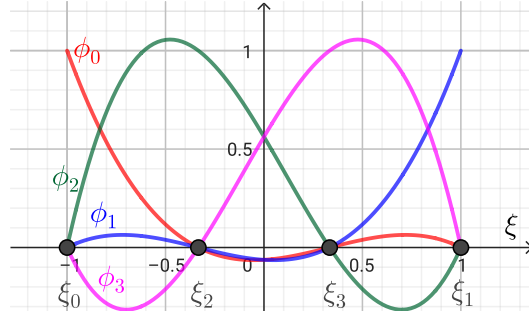


Рис. 8: Кубический базис в параметрическом отрезке

На рис. 9 представлено сравнение результатов аппроксимации функции  $f(x) = -x + \sin(2x + 1)$  линейным, квадратичным и кубическим базисом. Видно, что все интерполяционные приближения точно попадают в функцию в своих узлах интерполяции, а между узлами происходит аппроксимация полиномом соответствующей степени.

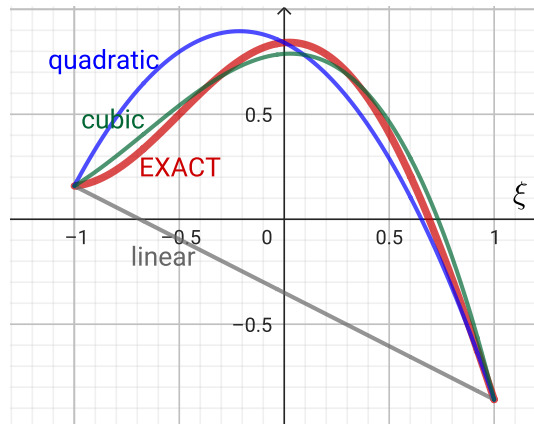


Рис. 9: Результат интерполяции

### В.3.1.3 Интерполяция в параметрическом треугольнике

Теперь рассмотрим двумерное обобщение формулы

#### Линейный базис

$$\phi_i(\xi, \eta) = A_i^{(00)} + A_i^{(10)}\xi + A_i^{(01)}\eta.$$

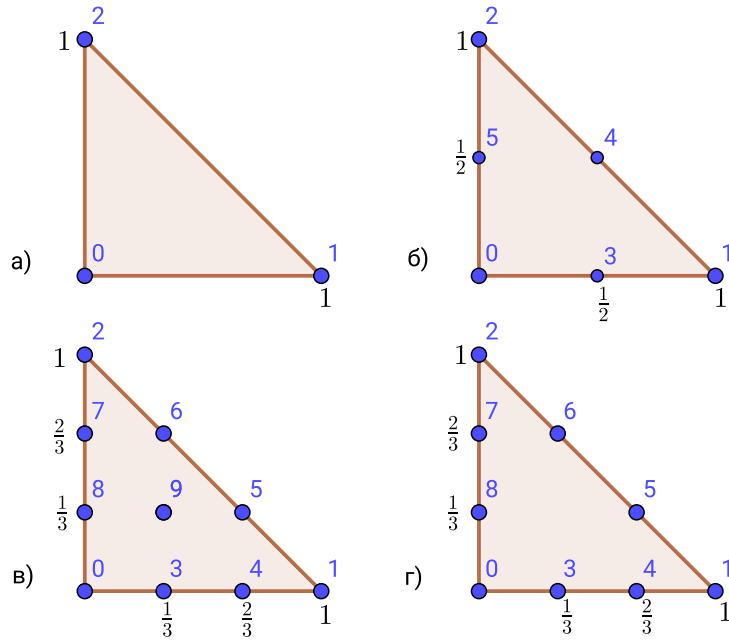


Рис. 10: Расположение узловых точек в параметрическом треугольнике. а) линейный базис, б) квадратичный базис, в) кубический базис, г) неполный кубический базис

$$C = \left( \begin{array}{c|ccc} & A^{(00)} & A^{(10)} & A^{(01)} \\ \hline \phi(0,0) & 1 & 0 & 0 \\ \phi(1,0) & 1 & 1 & 0 \\ \phi(0,1) & 1 & 0 & 1 \end{array} \right) \Rightarrow A = C^{-1} = \left( \begin{array}{c|ccc} & \phi_0 & \phi_1 & \phi_2 \\ \hline 1 & 1 & 0 & 0 \\ \xi & -1 & 1 & 0 \\ \eta & -1 & 0 & 1 \end{array} \right)$$

$$\begin{aligned} \phi_0(\xi, \eta) &= 1 - \xi - \eta, \\ \phi_1(\xi, \eta) &= \xi, \\ \phi_2(\xi, \eta) &= \eta, \end{aligned} \tag{B.23}$$

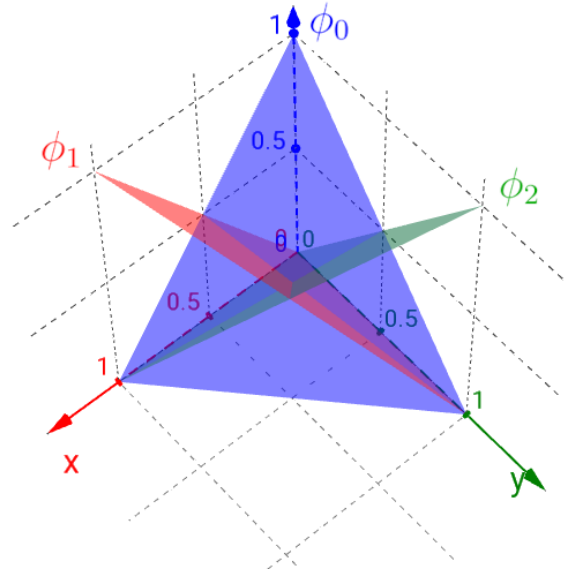


Рис. 11: Линейный базис в параметрическом треугольнике

## Квадратичный базис

$$\phi_i(\xi, \eta) = A_i^{(00)} + A_i^{(10)}\xi + A_i^{(01)}\eta + A_i^{(11)}\xi\eta + A_i^{(20)}\xi^2 + A_i^{(02)}\eta^2.$$

$$C = \left( \begin{array}{c|cccccc} & A^{(00)} & A^{(10)} & A^{(01)} & A^{(11)} & A^{(20)} & A^{(02)} \\ \hline \phi(0,0) & 1 & 0 & 0 & 0 & 0 & 0 \\ \phi(1,0) & 1 & 1 & 0 & 0 & 1 & 0 \\ \phi(0,1) & 1 & 0 & 1 & 0 & 0 & 1 \\ \phi(\frac{1}{2},0) & 1 & \frac{1}{2} & 0 & 0 & \frac{1}{4} & 0 \\ \phi(\frac{1}{2},\frac{1}{2}) & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \phi(0,\frac{1}{2}) & 1 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{4} \end{array} \right) \Rightarrow A = \left( \begin{array}{c|cccccc} & \phi_0 & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \xi & -3 & -1 & 0 & 4 & 0 & 0 \\ \eta & -3 & 0 & -1 & 0 & 0 & 4 \\ \xi\eta & 4 & 0 & 0 & -4 & 4 & -4 \\ \xi^2 & 2 & 2 & 0 & -4 & 0 & 0 \\ \eta^2 & 2 & 0 & 2 & 0 & 0 & -4 \end{array} \right)$$

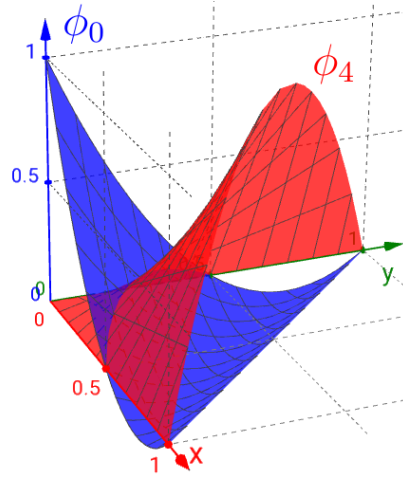


Рис. 12: Квадратичные функции  $\phi_0, \phi_4$  в параметрическом треугольнике

Кубический базис    TODO

Неполный кубический базис    TODO

### В.3.1.4 Интерполяция в параметрическом квадрате

Билинейный базис

$$\phi_i = A_i^{00} + A_i^{10}\xi + A_i^{01}\eta + A_i^{11}\xi\eta.$$



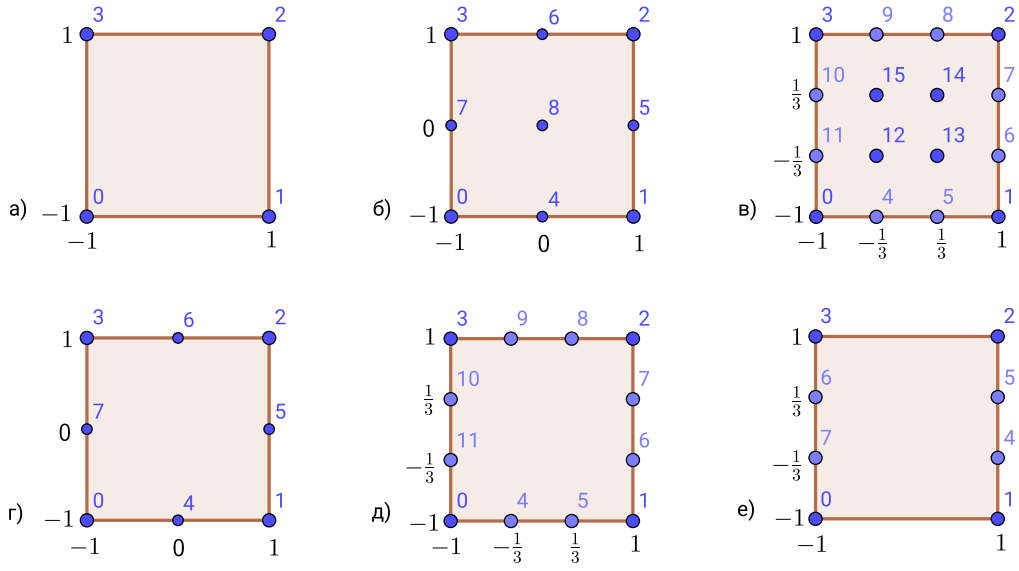


Рис. 13: Расположение узловых точек в параметрическом квадрате

$$C = \left( \begin{array}{c|cccc} & A^{(00)} & A^{(10)} & A^{(01)} & A^{(11)} \\ \hline \phi(-1, -1) & 1 & -1 & -1 & 1 \\ \phi(1, -1) & 1 & 1 & -1 & -1 \\ \phi(1, 1) & 1 & 1 & 1 & 1 \\ \phi(-1, 1) & 1 & -1 & 1 & -1 \end{array} \right) \Rightarrow A = C^{-1} = \frac{1}{4} \left( \begin{array}{c|cccc} & \phi_0 & \phi_1 & \phi_2 & \phi_3 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \xi & -1 & 1 & 1 & -1 \\ \eta & -1 & -1 & 1 & 1 \\ \xi\eta & 1 & -1 & 1 & -1 \end{array} \right)$$

$$\phi_0(\xi, \eta) = \frac{1 - \xi - \eta + \xi\eta}{4}$$

$$\phi_1(\xi, \eta) = \frac{1 + \xi - \eta - \xi\eta}{4}$$

$$\phi_2(\xi, \eta) = \frac{1 + \xi + \eta + \xi\eta}{4}$$

$$\phi_3(\xi, \eta) = \frac{1 - \xi + \eta - \xi\eta}{4}$$

(B.24)

**Определение двумерных базисов через комбинацию одномерных** Обратим внимание, что в искомые билинейные базисные функции линейны в каждом из направлений  $\xi, \eta$ , если брать их по отдельности. Значит можно представить эти функции как комбинацию одномерных линейных базисов (B.20) в каждом из направлений. Узлы двумерного параметрического квадрата можно выразить через узлы линейного базиса в параметрическом одномерном сегменте, рассмотренном в п. B.3.1.2:

$$\xi_0 = (\xi_0^{1D}, \xi_0^{1D}), \quad \xi_1 = (\xi_1^{1D}, \xi_0^{1D}), \quad \xi_2 = (\xi_1^{1D}, \xi_1^{1D}), \quad \xi_3 = (\xi_0^{1D}, \xi_1^{1D}).$$

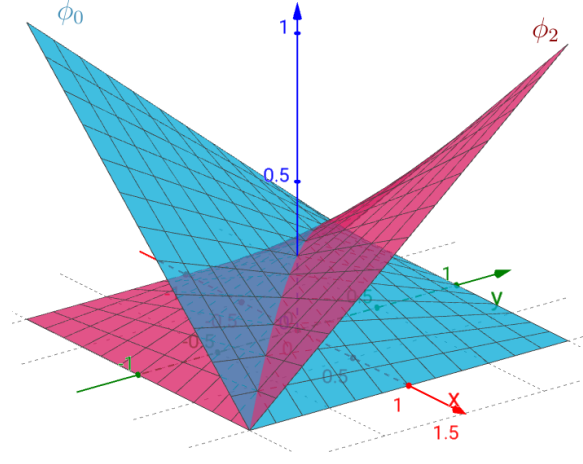


Рис. 14: Билинейные функции  $\phi_0, \phi_2$  в параметрическом квадрате

Значит и соответствующие базисные функции можно выразить через линейный одномерный базис  $\phi^{1D}$  из соотношений (B.20):

$$\begin{aligned}\phi_0(\xi, \eta) &= \phi_0^{1D}(\xi)\phi_0^{1D}(\eta) = \frac{1-\xi}{2}\frac{1-\eta}{2}, \\ \phi_1(\xi, \eta) &= \phi_1^{1D}(\xi)\phi_0^{1D}(\eta) = \frac{1+\xi}{2}\frac{1-\eta}{2}, \\ \phi_2(\xi, \eta) &= \phi_1^{1D}(\xi)\phi_1^{1D}(\eta) = \frac{1+\xi}{2}\frac{1+\eta}{2}, \\ \phi_3(\xi, \eta) &= \phi_0^{1D}(\xi)\phi_1^{1D}(\eta) = \frac{1-\xi}{2}\frac{1+\eta}{2}.\end{aligned}$$

Раскрыв скобки можно убедиться, что мы получили тот же билинейный базис, что и ранее (B.24).

**Биквадратичный базис** Применим этот метод для вычисления биквадратичного базиса, определённого в точках на рис. 13б. В качестве основе возьмём квадратичный одномерный базис  $\phi_i^{1D}$  из (B.21).

$$\begin{aligned}\phi_0(\xi, \eta) &= \phi_0^{1D}(\xi)\phi_0^{1D}(\eta) = \frac{\xi^2 - \xi\eta^2 - \eta}{2}, & \phi_1(\xi, \eta) &= \phi_1^{1D}(\xi)\phi_0^{1D}(\eta) = \frac{\xi^2 + \xi\eta^2 - \eta}{2}, \\ \phi_2(\xi, \eta) &= \phi_1^{1D}(\xi)\phi_1^{1D}(\eta) = \frac{\xi^2 + \xi\eta^2 + \eta}{2}, & \phi_3(\xi, \eta) &= \phi_0^{1D}(\xi)\phi_1^{1D}(\eta) = \frac{\xi^2 - \xi\eta^2 + \eta}{2}, \\ \phi_4(\xi, \eta) &= \phi_2^{1D}(\xi)\phi_0^{1D}(\eta) = (1 - \xi^2)\frac{\eta^2 - \eta}{2}, & \phi_5(\xi, \eta) &= \phi_1^{1D}(\xi)\phi_2^{1D}(\eta) = \frac{\xi^2 + \xi}{2}(1 - \eta^2), \\ \phi_6(\xi, \eta) &= \phi_2^{1D}(\xi)\phi_1^{1D}(\eta) = (1 - \xi^2)\frac{\eta^2 + \eta}{2}, & \phi_7(\xi, \eta) &= \phi_0^{1D}(\xi)\phi_2^{1D}(\eta) = \frac{\xi^2 - \xi}{2}(1 - \eta^2), \\ \phi_8(\xi, \eta) &= \phi_2^{1D}(\xi)\phi_2^{1D}(\eta) = (1 - \xi^2)(1 - \eta^2).\end{aligned}\tag{B.25}$$

**Бикубический базис**

**Неполный биквадратичный базис**

**Неполный бикубический базис**



## С.1 Геометрические алгоритмы

### С.1.1 Линейная интерполяция

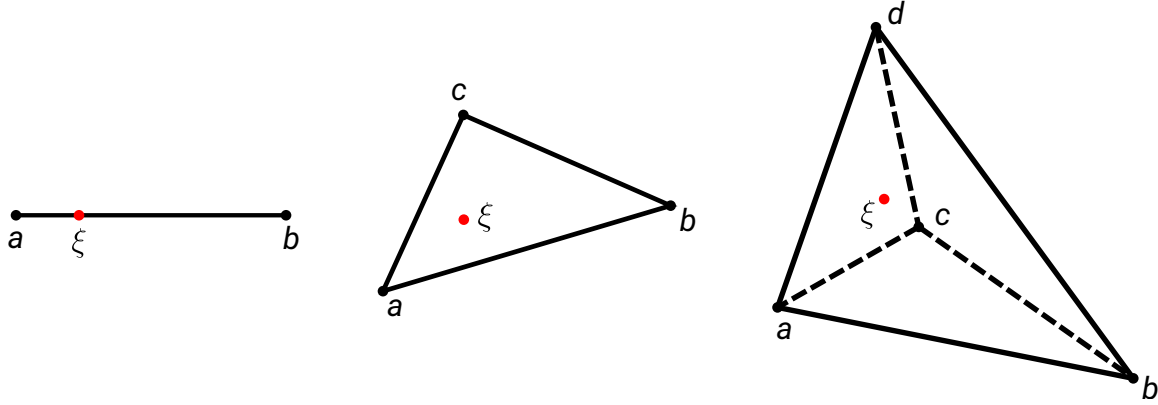


Рис. 15: Порядок нумерации точек одномерного, двумерного и трёхмерного симплекса при линейной интерполяции

Пусть функция  $u$  задана в узлах симплекса, имеющего нумерацию согласно рис. 15. Необходимо найти значение этой функции в точке  $\xi$  (эта точка вообще говоря не обязана лежать внутри симплекса).

Интерполяция в одномерном, двумерном и трёхмерном виде запишется как

$$u(\xi) = \frac{|\Delta_{\xi a}|u(b) + |\Delta_{b\xi}|u(a)}{|\Delta_{ba}|} \quad (\text{C.1})$$

$$u(\xi) = \frac{|\Delta_{ab\xi}|u(c) + |\Delta_{bc\xi}|u(a) + |\Delta_{ca\xi}|u(b)}{|\Delta_{abc}|} \quad (\text{C.2})$$

$$u(\xi) = \frac{|\Delta_{abc\xi}|u(d) + |\Delta_{cbd\xi}|u(a) + |\Delta_{cda\xi}|u(b) + |\Delta_{adb\xi}|u(c)}{|\Delta_{abcd}|}, \quad (\text{C.3})$$

где  $|\Delta|$  – знаковый объём симплекса, вычисляемый как

$$|\Delta_{ab}| = b - a,$$

$$|\Delta_{abc}| = \left( \frac{(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})}{2} \right)_z,$$

$$|\Delta_{abcd}| = \frac{(\mathbf{b} - \mathbf{a}) \cdot ((\mathbf{c} - \mathbf{a}) \times (\mathbf{d} - \mathbf{a}))}{6}.$$

### С.1.2 Преобразование координат

Рассмотрим преобразование из двумерной параметрической системы координат  $\xi$  в физическую систему  $\mathbf{x}$ . Такое преобразование полностью определяется покоординатными функциями  $\mathbf{x}(\xi)$ . Далее получим соотношения, связывающие операции дифференцирования и интегрирования в физической и параметрической областях.

### С.1.2.1 Матрица Якоби

Будем рассматривать двумерное преобразование  $(\xi, \eta) \rightarrow (x, y)$ . Линеаризуем это преобразование (разложим в ряд Фурье до линейного слагаемого)

$$\begin{aligned} x(\xi_0 + d\xi, \eta_0 + d\eta) &\approx x_0 + \left. \frac{\partial x}{\partial \xi} \right|_{\xi_0, \eta_0} d\xi + \left. \frac{\partial x}{\partial \eta} \right|_{\xi_0, \eta_0} d\eta, \\ y(\xi_0 + d\xi, \eta_0 + d\eta) &\approx y_0 + \left. \frac{\partial y}{\partial \xi} \right|_{\xi_0, \eta_0} d\xi + \left. \frac{\partial y}{\partial \eta} \right|_{\xi_0, \eta_0} d\eta, \end{aligned}$$

где  $x_0 = x(\xi_0, \eta_0)$ ,  $y_0 = y(\xi_0, \eta_0)$ . Переписывая это выражение в векторном виде, получим

$$\mathbf{x}(\xi_0 + d\xi) - \mathbf{x}_0 = J(\xi_0) d\xi. \quad (\text{C.4})$$

Матрица  $J$  (зависящая от точки приложения в параметрической плоскости) называется матрицей Якоби:

$$J = \begin{pmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix} \quad (\text{C.5})$$

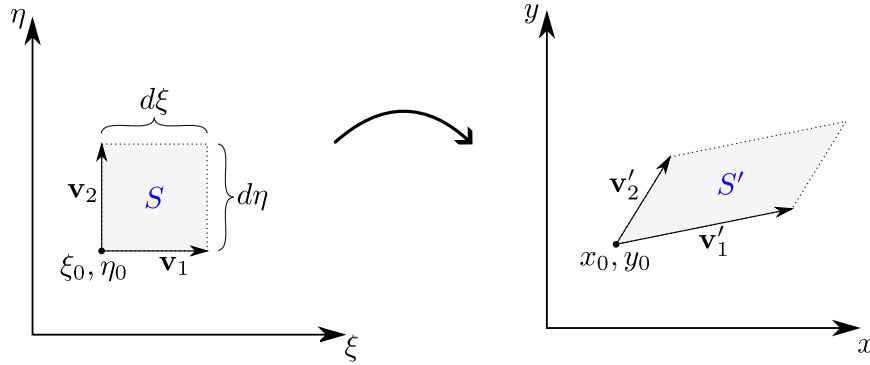


Рис. 16: Преобразование элементарного объёма

**Якобиан** Определитель матрицы Якоби (якобиан), взятый в конкретной точке параметрической плоскости  $\xi_0$ , показывает, во сколько раз увеличился элементарный объём около этой точки в результате преобразования. Действительно, рассмотрим два перпендикулярных элементарных вектора в параметрической системе координат:  $\mathbf{v}_1 = (d\xi, 0)$  и  $\mathbf{v}_2 = (0, d\eta)$  отложенных от точки  $\xi_0$  (см. рис. 16). В результате преобразования по формуле (C.4) получим следующие преобразования концевых точек и векторов:

$$\begin{aligned} (\xi_0, \eta_0) &\rightarrow (x_0, y_0), \\ (\xi_0 + d\xi, \eta_0) &\rightarrow (x_0 + J_{11}d\xi, y_0 + J_{21}d\xi) \Rightarrow \mathbf{v}_1 \rightarrow \mathbf{v}'_1 = (J_{11}d\xi, J_{21}d\xi), \\ (\xi_0, \eta_0 + d\eta) &\rightarrow (x_0 + J_{12}d\eta, y_0 + J_{22}d\eta) \Rightarrow \mathbf{v}_2 \rightarrow \mathbf{v}'_2 = (J_{12}d\eta, J_{22}d\eta). \end{aligned}$$

Элементарный объём равен площади параллелограмма, построенного на элементарных векторах. В параметрической плоскости согласно (B.4) получим

$$|S| = \mathbf{v}_1 \times \mathbf{v}_2 = d\xi d\eta,$$

и аналогично для физической плоскости:

$$|S'| = \mathbf{v}'_1 \times \mathbf{v}'_2 = (J_{11}J_{22} - J_{12}J_{21})d\xi d\eta = |J|d\xi d\eta$$

Сравнивая два последних соотношения приходим к выводу, что элементарный объём в результате преобразования увеличился в  $|J|$  раз. Тогда можно записать

$$dx dy = |J| d\xi d\eta \quad (\text{C.6})$$

Многомерным обобщением этой формулы будет

$$d\mathbf{x} = |J| d\xi \quad (\text{C.7})$$

### C.1.2.2 Дифференцирование в параметрической плоскости

Пусть задана некоторая функция  $f(x, y)$ . Распишем её производную по параметрическим координатам:

$$\begin{aligned} \frac{\partial f}{\partial \xi} &= \frac{\partial f}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial \xi}, \\ \frac{\partial f}{\partial \eta} &= \frac{\partial f}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial \eta}. \end{aligned}$$

Вспоминая определение (C.5), запишем

$$\begin{pmatrix} \frac{\partial f}{\partial \xi} \\ \frac{\partial f}{\partial \eta} \end{pmatrix} = J^T \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} J_{11} & J_{21} \\ J_{12} & J_{22} \end{pmatrix} \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

Обратная зависимость примет вид

$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = (J^T)^{-1} \begin{pmatrix} \frac{\partial f}{\partial \xi} \\ \frac{\partial f}{\partial \eta} \end{pmatrix} = \frac{1}{|J|} \begin{pmatrix} J_{22} & -J_{21} \\ -J_{12} & J_{11} \end{pmatrix} \begin{pmatrix} \frac{\partial f}{\partial \xi} \\ \frac{\partial f}{\partial \eta} \end{pmatrix}$$

В многомерном виде запишем

$$\nabla_{\mathbf{x}} f = (J^T)^{-1} \nabla_{\xi} f. \quad (\text{C.8})$$

### С.1.2.3 Интегрирование в параметрической плоскости

Пусть в физической области  $\mathbf{x}$  задана область  $D_x$ . Интеграл функции  $f(\mathbf{x})$  по этой области можно расписать, используя замену (C.7)

$$\int_{D_x} f(\mathbf{x}) d\mathbf{x} = \int_{D_\xi} f(\xi) |J(\xi)| d\xi, \quad (\text{C.9})$$

где  $f(\xi) = f(\mathbf{x}(\xi))$ , а  $D_\xi$  – образ области  $D_x$  в параметрической плоскости.

### С.1.2.4 Двумерное линейное преобразование. Параметрический треугольник

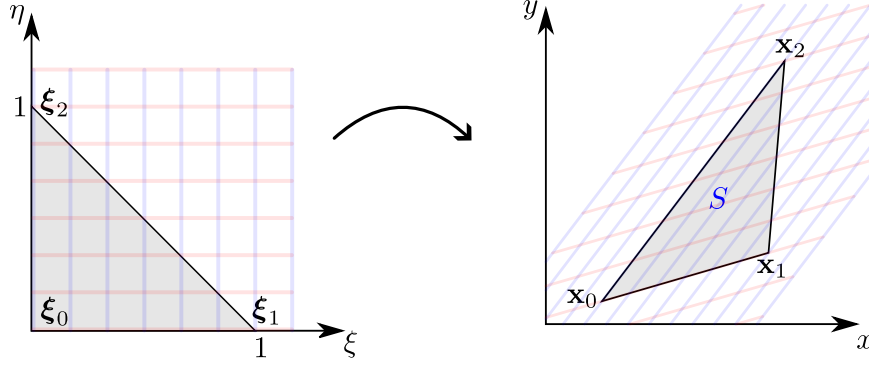


Рис. 17: Преобразование из параметрического треугольника

Рассмотрим двумерное преобразование, при котором определяющие функции являются линейными. То есть представимыми в виде

$$\begin{aligned} x(\xi, \eta) &= A_x \xi + B_x \eta + C_x, \\ y(\xi, \eta) &= A_y \xi + B_y \eta + C_y. \end{aligned}$$

Для определения шести констант, определяющих это преобразование, достаточно выбрать три любые (не лежащие на одной прямой) точки:  $(\xi_i, \eta_i) \rightarrow (x_i, y_i)$  для  $i = 0, 1, 2$ . В результате получим систему из шести линейных уравнений (три точки по две координаты), из которой находятся константы  $A_{x,y}, B_{x,y}, C_{x,y}$ . Пусть три точки в параметрической плоскости образуют единичный прямоугольный треугольник (рис. 17):

$$\xi_0, \eta_0 = (0, 0), \quad \xi_1, \eta_1 = (1, 0), \quad \xi_2, \eta_2 = (0, 1).$$

Тогда система линейных уравнений примет вид

$$\begin{aligned} x_0 &= C_x, & y_0 &= C_y, \\ x_1 &= A_x + C_x, & y_1 &= A_y + C_y, \\ x_2 &= B_x + C_x, & y_2 &= B_y + C_y. \end{aligned}$$

Определив коэффициенты преобразования из этой системы, окончательно запишем преобразование

$$\begin{aligned} x(\xi, \eta) &= (x_1 - x_0)\xi + (x_2 - x_0)\eta + x_0, \\ y(\xi, \eta) &= (y_1 - y_0)\xi + (y_2 - y_0)\eta + y_0. \end{aligned} \quad (\text{C.10})$$

Матрица Якоби этого преобразования (C.5) не будет зависеть от параметрических координат  $\xi, \eta$ :

$$J = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}. \quad (\text{C.11})$$

Якобиан преобразования будет равен удвоенной площади треугольника  $S$ , составленного из определяющих точек в физической плоскости:

$$|J| = (x_1 - x_0)(y_2 - y_0) - (y_1 - y_0)(x_2 - x_0) = (\mathbf{x}_1 - \mathbf{x}_0) \times (\mathbf{x}_2 - \mathbf{x}_0) = 2|S|. \quad (\text{C.12})$$

Распишем интеграл по треугольнику  $S$  по формуле (C.9). Вследствии линейности преобразования якобиан постоянен и, поэтому, его можно вынести его из-под интеграла:

$$\int_S f(x, y) dx dy = |J| \int_0^1 \int_0^{1-\xi} f(\xi, \eta) d\eta d\xi. \quad (\text{C.13})$$

#### С.1.2.5 Двумерное билинейное преобразование. Параметрический квадрат

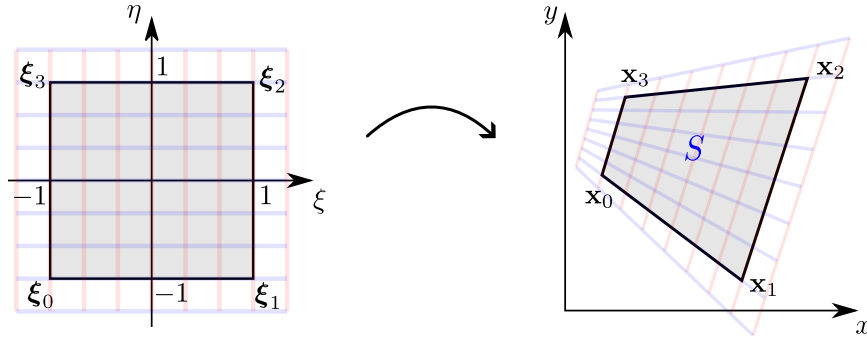


Рис. 18: Преобразование из параметрического квадрата

#### С.1.2.6 Трёхмерное линейное преобразование. Параметрический тетраэдр

TODO

#### С.1.3 Свойства многоугольника

##### С.1.3.1 Площадь многоугольника

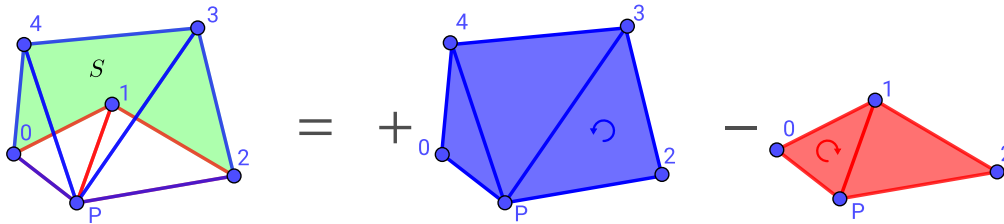


Рис. 19: Площадь произвольного многоугольника



Рассмотрим произвольный несамопересекающийся  $N$ -угольник  $S$ , заданный координатами своих узлов  $\mathbf{x}_i$ ,  $i = \overline{0, N-1}$ , пронумерованных последовательно против часовой стрелки (рис. 19). Далее введём произвольную точку  $\mathbf{p}$  и от этой точки будем строить ориентированные треугольники до граней многоугольника:

$$\Delta_i^p = (\mathbf{p}, \mathbf{x}_i, \mathbf{x}_{i+1}), \quad i = \overline{0, N-1},$$

(для корректности записи будем считать, что  $\mathbf{x}_N = \mathbf{x}_0$ ). Тогда площадь исходного многоугольника  $S$  будет равна сумме знаковых площадей треугольников  $\Delta_i^p$ :

$$|S| = \sum_{i=0}^{N-1} |\Delta_i^p|, \quad |\Delta_i^p| = \frac{(\mathbf{x}_i - \mathbf{p}) \times (\mathbf{x}_{i+1} - \mathbf{p})}{2}.$$

Знак площади ориентированного треугольника зависит от направления закрутки его узлов: она положительна для закрутки против часовой стрелки и отрицательна, если узлы пронумерованы по часовой стрелке. В частности, на рисунке 19 видно, что треугольники, отмеченные красным:  $P01, P12$ , будут иметь отрицательную площадь, а синие треугольники  $P23, P34, P40$  – положительную. Сумма этих площадей с учётом знака даст искомую площадь многоугольника.

Для сокращения вычислений воспользуемся произвольностью положения  $\mathbf{p}$  и совместим её с точкой  $\mathbf{x}_0$ . Тогда треугольники  $\Delta_0^p, \Delta_{N-1}^p$  вырождаются (будут иметь нулевую площадь). Обозначим такую последовательную триангуляцию как

$$\Delta_i = (\mathbf{x}_0, \mathbf{x}_i, \mathbf{x}_{i+1}), \quad i = \overline{1, N-2}. \quad (\text{C.14})$$

Знаковая площадь ориентированного треугольника будет равна

$$|\Delta_i| = \frac{(\mathbf{x}_i - \mathbf{x}_0) \times (\mathbf{x}_{i+1} - \mathbf{x}_0)}{2}. \quad (\text{C.15})$$

Тогда окончательно формула определения площади примет вид

$$|S| = \sum_{i=1}^{N-2} |\Delta_i|. \quad (\text{C.16})$$

**Плоский полигон в пространстве** Если плоский полигон  $S$  расположен в трёхмерном пространстве, то правая часть формулы (C.15) согласно определению векторного произведения в трёхмерном пространстве (B.3) – есть вектор. Чтобы получить скалярную площадь, нужно спроецировать этот вектор на единичную нормаль к плоскости многоугольника:

$$\mathbf{n} = \frac{\mathbf{k}}{|\mathbf{k}|}, \quad \mathbf{k} = (\mathbf{x}_1 - \mathbf{x}_0) \times (\mathbf{x}_2 - \mathbf{x}_0).$$

Эта формула записана из предположения, что узел  $\mathbf{x}_2$  не лежит на одной прямой с узлами  $\mathbf{x}_0, \mathbf{x}_1$ . Иначе вместо  $\mathbf{x}_2$  нужно выбрать любой другой узел, удовлетворяющий этому условию. Тогда площадь ориентированного треугольника, построенного в трёхмерном пространстве запишется через смешанное произведение:

$$|\Delta_i| = \frac{((\mathbf{x}_i - \mathbf{x}_0) \times (\mathbf{x}_{i+1} - \mathbf{x}_0)) \cdot \mathbf{n}}{2}. \quad (\text{C.17})$$

Формула для определения площади полигона (C.16) будет по-прежнему верна. При этом итоговый знак величины  $S$  будет положительным, если закрутка полигона положительная (против часовой стрелки) при взгляде со стороны вычисленной нормали  $\mathbf{n}$ .

### C.1.3.2 Интеграл по многоугольнику

Рассмотрим интеграл функции  $f(x, y)$  по  $N$ -угольнику  $S$ , заданному последовательными координатами своих узлов  $\mathbf{x}_i$ . Введём последовательную триангуляцию согласно (C.14). Тогда интеграл по многоугольнику можно расписать как сумму интегралов по ориентированным треугольникам:

$$\int_S f(x, y) dx dy = \sum_{i=1}^{N-2} \int_{\Delta_i} f(x, y) dx dy. \quad (\text{C.18})$$

Далее для вычисления интегралов в правой части воспользуемся преобразованием к параметрическому треугольнику (п. C.1.2.4). Следуя формуле интегрирования (C.13), распишем интеграл по  $i$ -ому треугольнику:

$$\int_{\Delta_i} f(x, y) dx dy = |J_i| \int_0^1 \int_0^{1-\xi} f_i(\xi, \eta) d\eta d\xi,$$

где якобиан  $|J_i|$  согласно (C.12) есть удвоенная площадь ориентированного треугольника  $\Delta_i$  (положительная при закрутке против часовой стрелки и отрицательная иначе):

$$|J_i| = 2|\Delta_i| = (\mathbf{x}_i - \mathbf{x}_0) \times (\mathbf{x}_{i+1} - \mathbf{x}_0),$$

а функция  $f_i(\xi, \eta)$  есть функция от преобразованных согласно (C.10) переменных:

$$f_i(\xi, \eta) = f((\mathbf{x}_i - \mathbf{x}_0)\xi + (\mathbf{x}_{i+1} - \mathbf{x}_0)\eta + \mathbf{x}_0).$$

Окончательно запишем

$$\int_S f(x, y) dx dy = 2 \sum_{i=1}^{N-2} |\Delta_i| \int_0^1 \int_0^{1-\xi} f_i(\xi, \eta) d\eta d\xi. \quad (\text{C.19})$$

Отметим, что эта формула работает и в том случае, когда полигон расположен в трёхмерном пространстве (знаковую площадь при этом следует вычислять по (C.17)).

### C.1.3.3 Центр масс многоугольника

По определению, координаты центра масс  $\mathbf{c}$  области  $S$  равны среднеинтегральным значениям координатных функций. То есть

$$c_x = \frac{1}{|S|} \int_S x dx dy, \quad c_y = \frac{1}{|S|} \int_S y dx dy.$$

Далее распишем интеграл в правой части через последовательную триангуляцию согласно (C.18) с учётом линейного преобразования (C.10):

$$\begin{aligned}
\int_S x \, dx dy &= \sum_{i=1}^{N-2} \int_{\Delta_i} x \, dx dy \\
&= \sum_{i=1}^{N-2} |J_i| \int_0^1 \int_0^{1-\xi} ((x_i - x_0)\xi + (x_{i+1} - x_0)\eta + x_0) \, d\eta d\xi \\
&= \sum_{i=1}^{N-2} \frac{|J_i|}{2} \frac{x_0 + x_i + x_{i+1}}{3} \\
&= \sum_{i=1}^{N-2} |\Delta_i| \frac{x_0 + x_i + x_{i+1}}{3}.
\end{aligned}$$

Итого, с учётом (C.16), координаты центра масс примут вид

$$\mathbf{c} = \frac{\sum_{i=1}^{N-2} \frac{\mathbf{x}_0 + \mathbf{x}_i + \mathbf{x}_{i+1}}{3} |\Delta_i|}{\sum_{i=1}^{N-2} |\Delta_i|}.$$

Если полигон расположен в двумерном пространстве  $xy$ , то знаковая площадь треугольников вычисляется по формуле (C.15). В случае трёхмерного пространства должна использоваться формула (C.17).

## C.1.4 Свойства многогранника

### C.1.4.1 Объём многогранника

TODO

### C.1.4.2 Интеграл по многограннику

TODO

### C.1.4.3 Центр масс многогранника

TODO

## C.1.5 Поиск многоугольника, содержащего заданную точку

TODO

## С.2 Форматы хранения разреженных матриц

### С.2.1 CSR-формат

При реализации решателей систем сеточных уравнений важно учитывать разреженный характер используемых в левой части. То есть избегать хранения и ненужных операций с нулевыми элементами матрицы.

Хотя рассмотренные ранее алгоритмы конечноразностных аппроксимаций на структурированных сетках давали трёх- (для одномерных задач) или пятидиагональную (для двумерных) сеточную матрицу, здесь будем рассматривать общие форматы хранения, не привязанные к конкретному шаблону.

Любой общий формат хранения должен хранить информацию о шаблоне матрице (адресах ненулевых элементов) и значениях матричных коэффициентов в этом шаблоне.

В CSR (Compressed sparse rows) формате все ненулевые элементы хранятся в линейном массиве `vals`. А шаблон матрицы – в двух массивах

- массиве колонок `cols` – значений колонок для соответствующих ему значений из массива `vals`,
- массиве адресов `addr` – индексах массива `vals`, с которых начинается описание соответствующей строки.

В конце массива `addr` добавляется общая длина массива `vals`.

Таким образом, длины массивов `vals`, `cols` равны количеству ненулевых элементов матрицы, а длина массива `addr` равна количеству строк в матрице плюс один.

Для облегчения процедур поиска описание каждой строки должно идти последовательно с увеличением индекса колонки.

Для примера рассмотрим следующую матрицу

$$\begin{pmatrix} 2.0 & 0 & 0 & 1.0 \\ 0 & 3.0 & 5.0 & 4.0 \\ 0 & 0 & 6.0 & 0 \\ 0 & 7.0 & 0 & 8.0 \end{pmatrix} \quad (C.20)$$

Массивы, описывающие матрицу в формате CSR примут вид

	<i>row</i> = 0	<i>row</i> = 1	<i>row</i> = 2	<i>row</i> = 3	
<i>vals</i> =	2.0, 1.0,	3.0, 5.0, 4.0,	6.0,	7.0, 8.0	
<i>cols</i> =	0, 3,	1, 2, 3,	2,	1, 3	
<i>addr</i> =	0,	2,	5,	6,	8

Рассмотрим реализацию базовых алгоритмов для матриц, заданных в этом формате.

Пусть матрица задана следующими массивами:

```
std::vector<double> vals; // массив значений
std::vector<size_t> cols; // массив столбцов
std::vector<size_t> addr; // массив адресов
```

Число строк в матрице:

```
size_t nrows = addr.size()-1;
```

Число элементов в шаблоне (ненулевых элементов)

```
size_t n_nonzeros = vals.size();
```

Число ненулевых элементов в заданной строке 'irow'

```
size_t n_nonzeros_in_row = addr[irow+1] - addr[irow];
```

Умножение матрицы на вектор 'v' (длина этого вектора должна быть равна числу строк в матрице). Здесь реализуется суммирование вида

$$r_i = \sum_{j=0}^{N-1} A_{ij}v_j,$$

при этом избегаются лишние операции с нулями

```
// число строк в матрице и длина вектора v
size_t nrows = addr.size() - 1;
// массив ответов. Инициализируем нулями
std::vector<double> r(nrows, 0);
// цикл по строкам
for (size_t irow=0; irow < nrows; ++irow){
    // цикл по ненулевым элементам строки irow
    for (size_t a = addr[irow]; a < addr[irow+1]; ++a){
        // получаем индекс колонки
        size_t icol = cols[a];
        // значение матрицы на позиции [irow, icol]
        double val = vals[a];
        // добавляем к ответу
        r[irow] += val * v[icol];
    }
}
```

Поиск значения элемента матрицы по адресу (irow, icol) с учётом локально сортированного вектора cols

```

using iter_t = std::vector<size_t>::const_iterator;
// указатели на начало и конец описания строки в массиве cols
iter_t it_start = cols.begin() + addr[irow];
iter_t it_end = cols.begin() + addr[irow+1];
// поиск значения icol в отсортированной последовательности [it_start, it_end)
iter_t fnd = std::lower_bound(it_start, it_end, icol);
if (fnd != it_end && *fnd == icol){
    // если нашли, то определяем индекс найденного элемента в массиве cols
    size_t a = fnd - cols.begin();
    // и возвращаем значение из vals по этому индексу
    return vals[a];
} else {
    // если не нашли, значит элемент [irow, icol] находится вне шаблона. Возвращаем 0
    return 0;
}

```

Формат CSR обеспечивает максимальную компактность хранения разреженной матрицы и при этом удобен для последовательной итерации по элементам матрицы (операции умножения матрицы на вектор), но его существенным недостатком является высокая сложность добавления нового элемента в шаблон.

### C.2.2 Массив словарей

При реализации сеточных методов решения дифференциальных уравнений работу с матрицами можно разбить на два этапа: сборка матриц и их непосредственное использование. Сборка матрицы в свою очередь может быть разделена на этап вычисление шаблона матрицы (символьная сборка) и непосредственное вычисление коэффициентов матрицы (числовая сборка).

На этапе использования матрицы основной операцией является умножение матрицы на вектор, где наиболее эффективным является CSR-формат.

В случае использования неструктурированных сеток этап символьной сборки является нетривиальной операцией и сводится к неупорядоченному добавлению новых элементов в шаблон матрицы. Как было отмечено ранее, такая операция в случае использования CSR формата неэффективна.

Поэтому часто для этапов сборки и расчёта используют разные форматы хранения матриц, первый из которых оптимизирован для операции вставки, а второй – для операции умножения на вектор. В качестве формата, оптимизированного для вставки, можно представить формат массива словарей (List of dictionaries), где каждая строка матрицы описывается словарём, ключём которого является индекс колонки, а значением – величина соответствующего матричного коэффициента.

С использованием синтаксиса C++ такой формат может быть описан следующим образом:

```

std::vector<std::map<size_t, double>> data;

```

Матрица вида (C.20) в таком формате примет вид

```
data = {
    {0: 2.0, 3: 1.0},
    {1: 3.0, 2: 5.0, 3: 4.0},
    {2: 6.0},
    {1: 7.0, 3: 8.0}
};
```

Добавление нового матричного коэффициента сведётся к вставке элемента в словарь:

```
data[i][j] = value;
```

А основной операцией для такого формата будет служить конверсия в CSR:

```
std::vector<size_t> addr{0};
std::vector<size_t> cols;
std::vector<double> vals;
for (size_t irow=0; irow < data.size(); ++irow){
    for (auto it: data[irow]){
        cols.push_back(it.first);
        vals.push_back(it.second);
    }
    addr.push_back(addr.back() + data[irow].size());
}
```

Поскольку данные в контейнере типа

`std::map` итерируются в отсортированном по ключам порядке, то полученный в результате массив `cols` также является локально отсортированным.

## D Работа с инфраструктурой проекта CFDCourse



В настоящем параграфе будут даны инструкции для разворачивания инфраструктуры сборки проекта для операционных систем Linux(Ubuntu), MacOS и Windows.

В процессе настройки будет необходимо установить систему контроля версий **Git**, систему контейнеризации **Docker** и (опционально) интегрированную среду разработки. Проект позволяет работать в любой среде разработки. Ниже будут приведены инструкции для настройки **vscode**.

Процесс сборки и запуска программ будет осуществляться в системе, развёрнутой в докере на основе Ubuntu 24.04. В дальнейшем систему, установленную непосредственно на компьютере, будем называть хост системой. А систему, развёрнутую в докере – контейнером.

Для успешной установке на хосте должно быть около 5Гб свободного места.

## D.1 Клонирование

Для клонирования проекта на локальный компьютер необходимо установить систему контроля версий Git и открыть терминал на хост-системе в папке, в которой планируется хранить папку с репозиторием. В нижеследующих инструкциях в качестве такой папки будет использоваться домашняя папка пользователя.

### Ubuntu

Откройте терминал и в нём

```
sudo apt install git # установка гита
cd ~                 # переходим в папку,
                     # где будет храниться папка с репозиторием
```

### Windows

В Windows необходимо скачать и установить дистрибутив <https://github.com/git-for-windows/git/releases/download/v2.51.0.windows.1/Git-2.51.0-64-bit.exe>.

Далее откройте командную строку `cmd`. И в ней перейдите в целевую папку

```
cd %USERPROFILE%
```

### MacOs

Откройте терминал и в нём

```
# Установите Homebrew, если у вас его ещё нет
/bin/bash -c \
"$ (curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh) "
# установка гита
brew install git
# переходим в папку где будет храниться папка с репозиторием
cd ~
```

Далее необходимо клонировать репозиторий

```
git clone https://github.com/kalininei/CFDCourse26
```

В результате в домашней папке должна появиться папка с `CFDCourse26`.

## D.2 Разворачивание контейнера

Установим докер

### Ubuntu+apt

Запустить скрипт, написанный на основе инструкций с официального сайта <https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>:

```
cd ~/CFDCourse26      # перейдём в репозиторий
./scripts/ubuntu_docker_install.sh
```

### Windows/MacOs+DockerDesktop

Скачайте и установите дистрибутив с официального сайта

- <https://docs.docker.com/desktop/setup/install/windows-install/> для Windows
- <https://docs.docker.com/desktop/setup/install/mac-install/> для MacOS

Далее следуйте процедуре установки десктопного приложения. После установки запустите `DockerDesktop`. Этап регистрации при запуске опционален и может быть пропущен.

Далее в терминале находясь в директории `CFDCourse26` развернём контейнер:

```
docker compose up --build -d
```

На этом этапе будут скачаны необходимые образы и запущен контейнер. По окончании можно убедиться, что контейнер работает

```
docker ps
```

В случае работы с `DockerDesktop` запущенный контейнер будет виден в графическом интерфейсе во вкладке `Containers`.

## D.3 Базовая разработка

Чтобы скомпилировать проект необходимо войти в терминал контейнера. Далее

```
docker ps                # Убедимся, что контейнер cfd26 запущен
docker exec -it cfd26 bash # Войдём в него
```

Если вы работаете на Unix-системе с несколькими пользователями и ваш пользователь не является дефолтным, то лучше войти в контейнер под своим пользовательским id. Для этого нужно определить необходимые переменные окружения выполнив вместо последней команды из предыдущего листинга:

```
USER_ID=$(id -u) GROUP_ID=$(id -g) docker exec -it cfd26 bash
```

Папка хоста с репозиторием `CFDCourse` примонтирована к папке контейнера `/app`. Войдём туда

```
cd /app    # заходим в директорию
ls -alh    # смотрим список файлов
```

Для удобства в контейнере установлен консольный файловый менеджер `mc`, который можно использовать для операций с файлами. Так же есть его псевдоним `mcs`, который отличается от базового `mc` тем, что запоминает текущую директорию при выходе. Благодаря чему его можно использовать для навигации вместо `cd`.

### D.3.1 Особенности проекта

- Проект состоит из статически линкуемой библиотеки `libcfd26.a` и исполняемого файла `cfd26_test` с тестовыми программами для этой библиотеки. Исходники библиотеки лежат в директории `src/cfd`, исходники тестов – `src/test`,
- Используется 20-ый стандарт C++,
- Сборка осуществляется в системе `smake`,
- Для написания тестов используется фреймворк `Catch2`,
- В проекте установлены жёсткие правила работы с предупреждениями компиляции, из-за которых они как обрабатываются ошибки,
- В проекте установлены форматтеры для исходных кодов на C++, python, `smake`. При сохранении любого исходного файла этих форматов в `vscode` форматтер будет вызван автоматически. Если исходники модифицировались иначе, то запустить форматтер для всех файлов проекта можно скриптом `scripts/formatall.sh` из папки `/app`.

### D.3.2 Сборка в отладочном режиме

Из папки `/app` контейнера:

```
mkdir build          # создаём папку, в которую будут строиться программа
cd build             # Заходим
smake .. -DCMAKE_BUILD_TYPE=Debug # Строим сборочные скрипты
make -j4              # Компилируем на 4-х потоках
```

Сама папка

`build` является временной папкой для построения. Она не подключена к системе контроля версий. И может быть удалена и создана заново (например для полной гарантированной очистки кэша построения). Бинарные файлы будут построены в папку `/app/build/bin`. Для запуска тестов зайдём в эту папку:

```
cd bin
./cfd26_test          # запуск всех тестов
./cfd26_test [grid1]  # запуск единственного тест кейса
```

### D.3.3 Сборка в релизном режиме

Отличается от сборки в отладочном режиме только флагом `cmake`. Будем строить в папку `build_release`. Также от папки `/app`

```
mkdir build_release      # создаём папку
cd build_release         # Заходим
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo # Или просто Release,
                                     # если отладочная информация не нужна
make -j4                 # Компилируем на 4-х потоках
cd bin                   # заходим в папку с исполняемым файлом
./cfd26_test             # запуск всех тестов
./cfd26_test [grid1]     # запуск единственного тест кейса
```

### D.3.4 Работа с кодом

Работать с исходными файлами можно находясь на хосте и используя любой удобный текстовый редактор. (например

`notepad++` на Windows). Порядок работы будет выглядеть следующим образом

- Редактировать исходные файлы на хосте в папке `CFDCourse26/src`
- Для сборки переключиться на терминал и выполнить вышеописанную процедуру сборки
- Если сборка не прошла из-за ошибок в коде, эти ошибки будут распечатаны в терминал с указанием файла и номера строки
- Для отладки можно использовать консольный отладчик `gdb` из терминала

## D.4 Разработка в vscode

### D.4.1 Подключение к контейнеру

Необходимо установить vscode на хосте следуя инструкциям с официального сайта <https://code.visualstudio.com/Download>. В самом vscode нужно установить расширение

`Remote Explorer, Dev Containers` для удалённой разработки в контейнере. Далее нужно переключиться на вкладку `RemoteExplorer` (см. рис. 20).

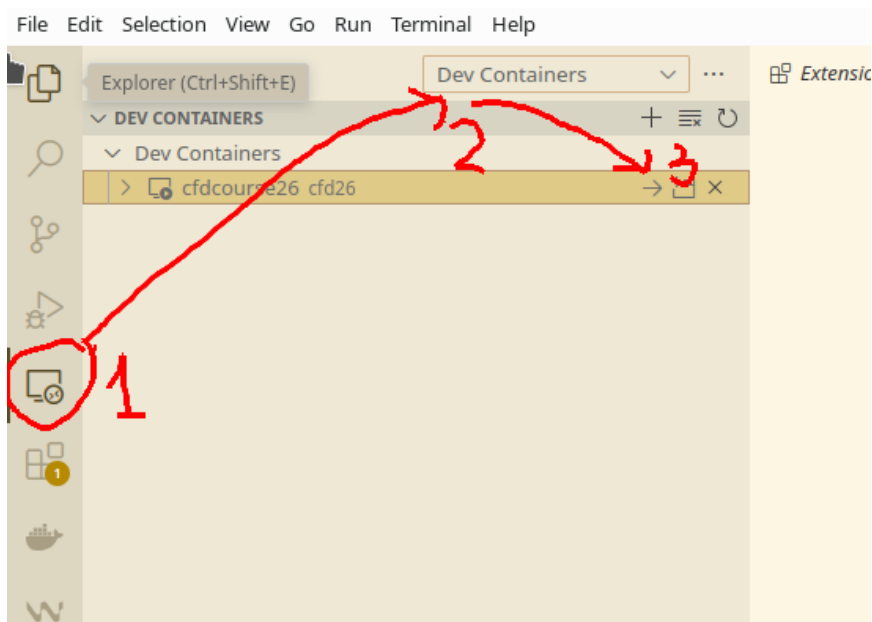


Рис. 20: Подключения vscode к контейнеру

## D.4.2 Настройки vscode

Далее необходимо открыть папку `/app`:

**File->Open Folder...** Контейнер содержит в себе базовые настройки vscode, которые при сборке контейнера копируются в папки `.vscode`, `.vscode-server`.

Следует отметить, что удалённо подключённый vscode не может пользоваться расширениями, установленными на хосте. Все необходимые расширения нужно устанавливать в контейнер заново. Список базовых расширений, необходимых для работы, уже содержится в настроечных файлах. Когда вы откроете папку `app` в vscode, вам будет предложено эти расширения установить. Нужно согласиться.

### Настроечные папки

`.vscode`, `.vscode-server` игнорируются системой контроля версий и расположены на хосте. То есть вы можете дополнительно установить туда любые свои расширения и донастроить vscode как вам удобно. Эти настройки не будут зависеть от ветки гита и не будут затираться при пересборке контейнера.

Если всё же понадобится обнулить настройки vscode, нужно

1. удалить папки `.vscode`, `.vscode_server`
2. удалить все настройки из конфигурационного файла контейнера (`ctrl+shift+p`, `open container configuration file`)
3. выйти из контейнера на vscode: **File->Close Remote Connection**
4. остановить и пересобрать контейнер на хосте

```
docker stop cfd26
docker compose up --build -d
```

### D.4.3 Сборка и отладка

В папке `.vscode` лежат базовые таски и лаунчи для компиляции и запуска программы. В случае необходимости можете дополнить базовый набор своими командами. Согласно настройкам по умолчанию по нажатию `F5` происходит сборка программы в отладочном режиме и запуск отладки тестовой программы с прогоном всех тестов. Посмотреть результаты можно на вкладке `TERMINAL`. В случае ошибок компиляции, список этих ошибок будет виден на вкладке `PROBLEMS`.

Для отладки конкретного теста необходимо запустить тестовую программу с аргументом (например `cf2d26_test [grid1]`). Чтобы передать программе аргумент нужно этот аргумент прописать в файле `.vscode/launch.json` в поле `args`. Либо создать ещё одну конфигурацию запуска с вашими аргументами и указать эту конфигурацию в настройке `Run And Debug`.

Чтобы собрать программу без запуска нужно выполнить таск (`ctrl+shift+b`):

`cmake: build debug, cmake: build release` для отладочного и релизного режима соответственно.

В целом сборка на `vscode` представляет из себя автоматизированный алгоритм, представленный в п. D.3. То есть исполняемая программа в дебаговой версии кладётся в папку `build`, в релизной – в `build_release` и её можно запустить из терминала. Удаление этих папок ведёт к полной очистке кэша построения.

## D.5 Работа с системой контроля версий

Работать с гитом можно как с хоста (из папки `CFDCourse26`), так и из контейнера (из папки `\app`). Ниже будут даны инструкции для работы с гитом в консоли. Альтернативно, можно установить графический интерфейс (например `GitExtensions` для Windows) или командами `vscode` на вкладке `Source Control`.

Из системы контроля версий исключены следующие каталоги:

- `build*/` – папки со сборками,
- `.vscode`, `.vscode-server` – настройки и расширения `vscode`,
- `local_data` – папка для хранения любых пользовательских данных.

Изменения из этих папок не будут отслежены и скоммичены.

### D.5.1 Порядок работы с репозиторием CFDCourse

Основная ветка проекта – `master`. После каждой лекции в эту ветку будет отправлен коммит с сообщением `lect{index}`. В этом коммите будет дополнен pdf документ с содержанием лекции, задание по итогам лекции и необходимые для этого задания изменения в коде.

#### D.5.1.1 Получение последнего коммита

Таким образом, **после лекции**, после того, как изменение `lect{index}` придёт на сервер, необходимо выполнить следующие команды

```
git checkout master # перейти на основную ветку
git pull            # получить изменения
```

Если изменения не содержали никаких изменений в настройках контейнера `Dockerfile`, `docker-compose.yaml`, то для сборки проекта рестарт контейнера не требуется. Иначе требуется пересобрать контейнер:

```
docker stop cfd26          # остановить текущий контейнер
docker compose up --build -d # пересобрать новый
```

#### D.5.1.2 Создание коммита с текущим дз

**Перед началом лекции**, если была сделана какая то работа по заданиям,

```
git checkout -b hw-lect{index} # создать локальную ветку, содержащую задание
git add .
git commit -m "{свой комментарий}" # скоммитить свои изменения в эту ветку
```

Даже если задание выполнено не до конца, вы в любой момент можете переключиться на ветку с заданием и его доделать

```
git checkout hw-lect{index}
```

### D.5.1.3 Создание коммита с прошлым дз

Если вы не сделали задание вовремя и решили вернуться к нему позже, то нужно

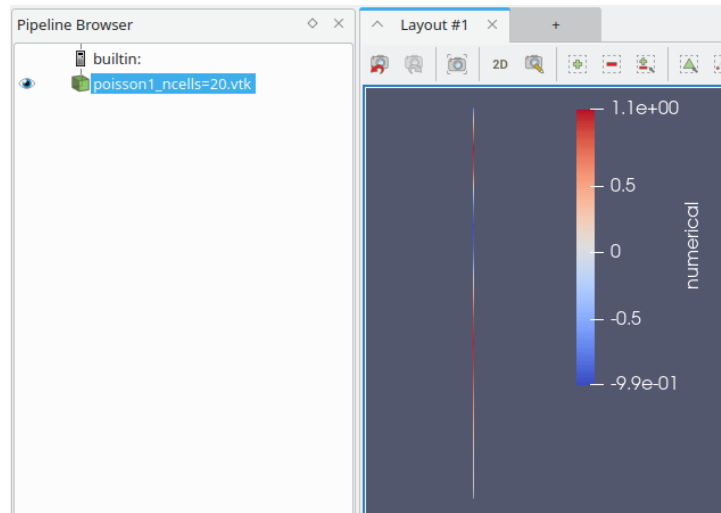
```
git checkout master          # перейти на основную ветку
git log --oneline            # в списке всех коммитов найти хэш коммита
                             # lect{index} той лекции которую нужно сделать
git checkout <...>           # переключиться на этот коммит по его хэшу
git checkout -b hw-lect{index} # создать ветку от этого коммита и работать в этой ветке
...                           # делаем работу
git add .
git commit -m "comment"      # по окончании работы скоммитить изменения
git checkout master          # и вернуться в основную ветку
```



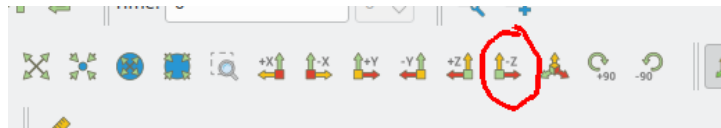
## D.6 Paraview

### D.6.1 Данные на одномерных сетках

Заданные на сетке данные паравью показывает цветом. Поэтому при загрузке одномерных сеток можно видеть картинку типа

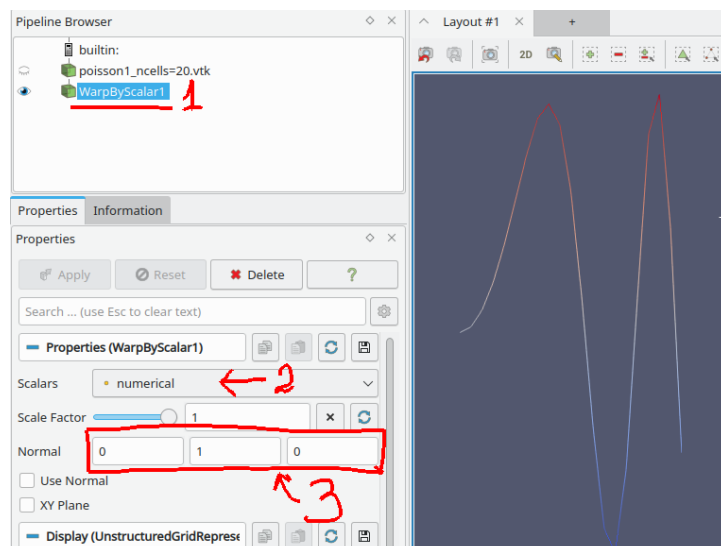


Развернуть изображение в плоскость ху



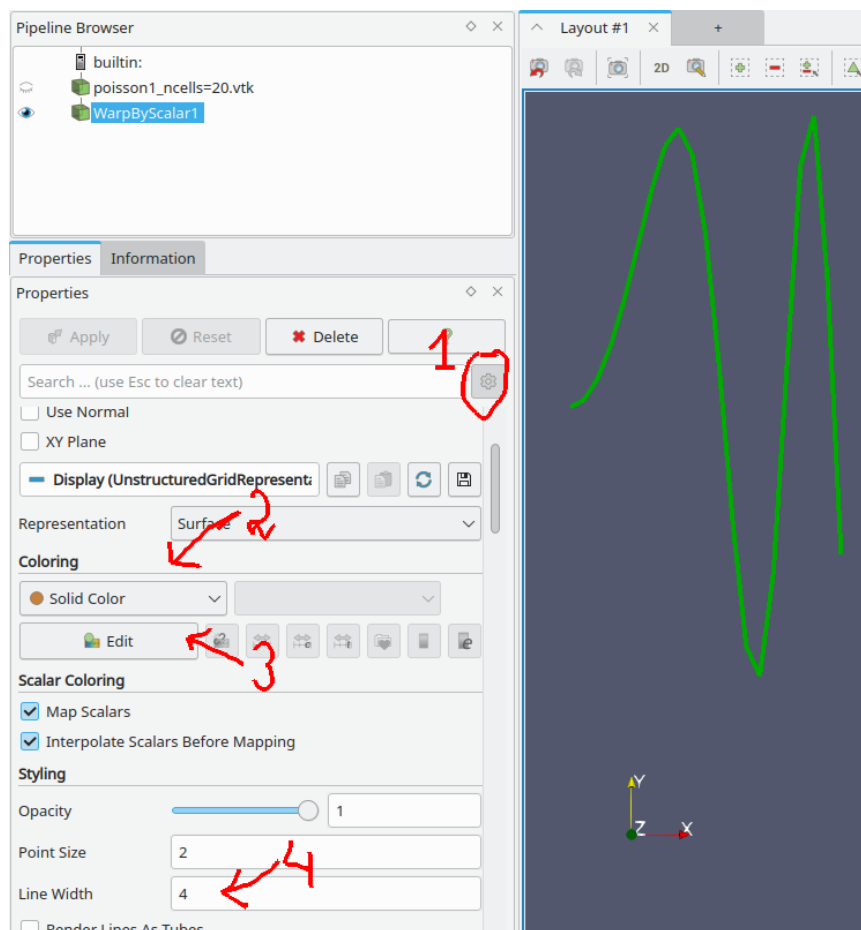
**Отобразить данные в виде у-координаты** Для того, что бы данные отображались в качестве значения по оси ординат, к загруженному файлу необходимо

1. применить фильтр **WarpByScalar** (В меню **Filters->Alphabetical->Warp By Scalar**)
2. в меню настройки фильтра указать поле данных, для отображения (numerical в примере ниже)
3. И настроить нормаль, вдоль которой будут проецироваться данные (в нашем случае ось у)



## Цвет и толщина линии

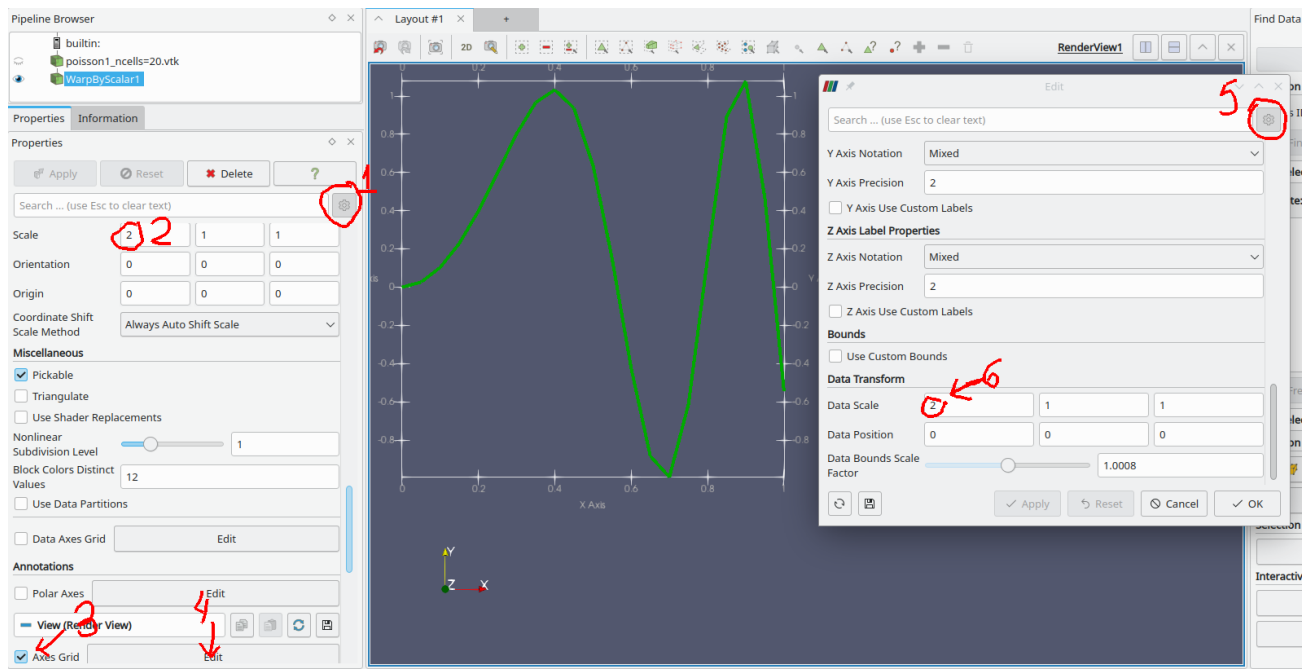
1. Включить подробные опции фильтра
2. Сменить стиль на **Solid Color**
3. В меню **Edit** выбрать желаемый цвет
4. В строке **Line Width** указать толщину линии



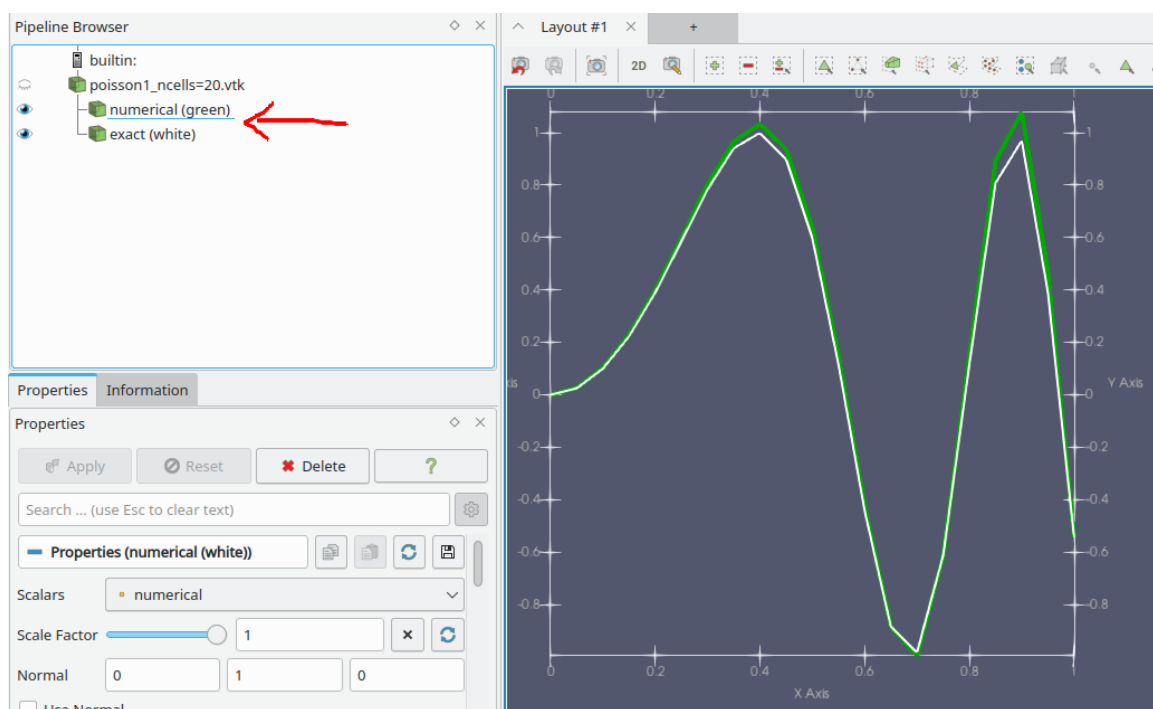
## Настройка масштабов и отображение осей координат

1. Отметьте подробные настройки фильтра
2. В поле **Transforming/Scale** Установите желаемые масштабы (в нашем случае растянуть в два раза по оси x)
3. Установите галку на отображение осей
4. откройте меню натройки осей
5. В нём включите подробные настройки
6. И также поставьте растяжение осей

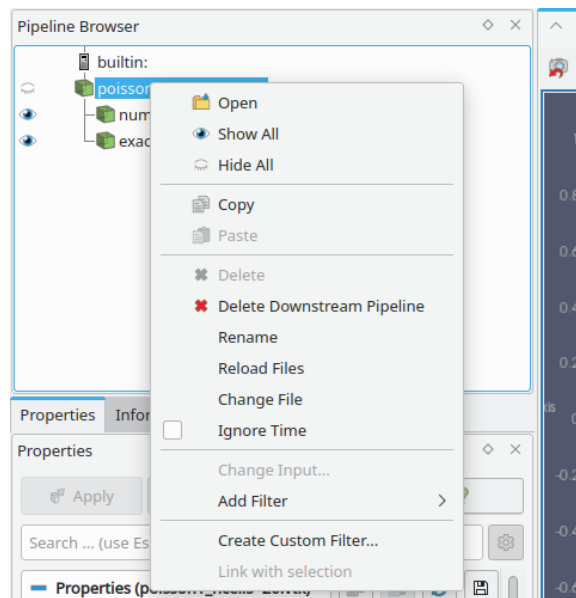
В случае, если масштабировать график не нужно, достаточно выполнить шаг 3.



**Построение графиков для нескольких данных** Если требуется нарисовать рядом несколько графиков для разных данных из одного файла, примените фильтр **Warp By Scalar** для этого файла ещё раз, изменив поле **Scalars** в настройке фильтра. Для наглядности измените имя узла в Pipeline Browser на осмысленные

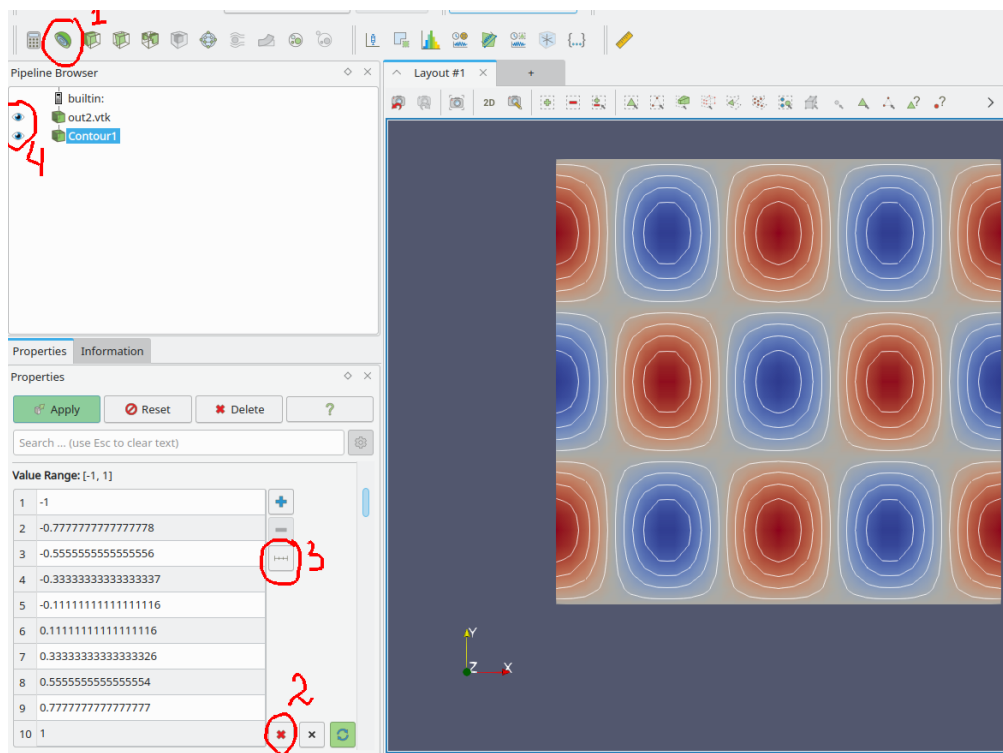


**Обновление данных при изменении исходного файла** В случае, если исходный файл был изменён, нужно в контекстном меню узла соответствующего файла выбрать **Reload Files** (или нажать F5). Если те же самые фильтры нужно применить для просмотра другого файла нужно в этом меню нажать **Change File**.

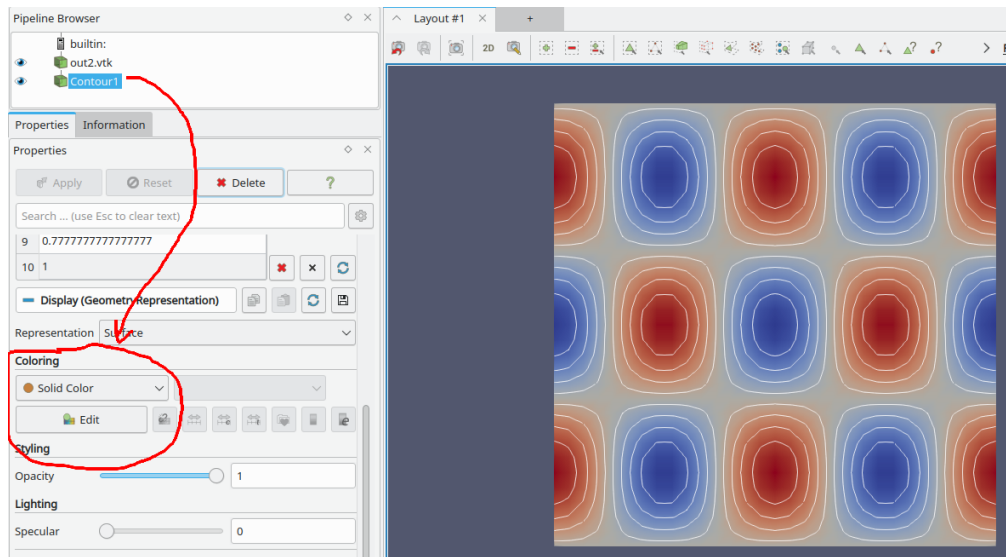


## D.6.2 Изолинии для двумерного поля

1. Нажмите иконку **Contour** (или **Filters/Contour**) В настройках фильтра Contour by выберите данные, по которым нужно строить изолинии.
2. В настройках фильтра удалите все существующие записи о значениях для изолиний
3. Добавьте равномерные значения. В появившемся меню установите необходимое количество изолиний и их диапазон.
4. Если необходимо, включите одновременное отображения цветного поля и изолиний.



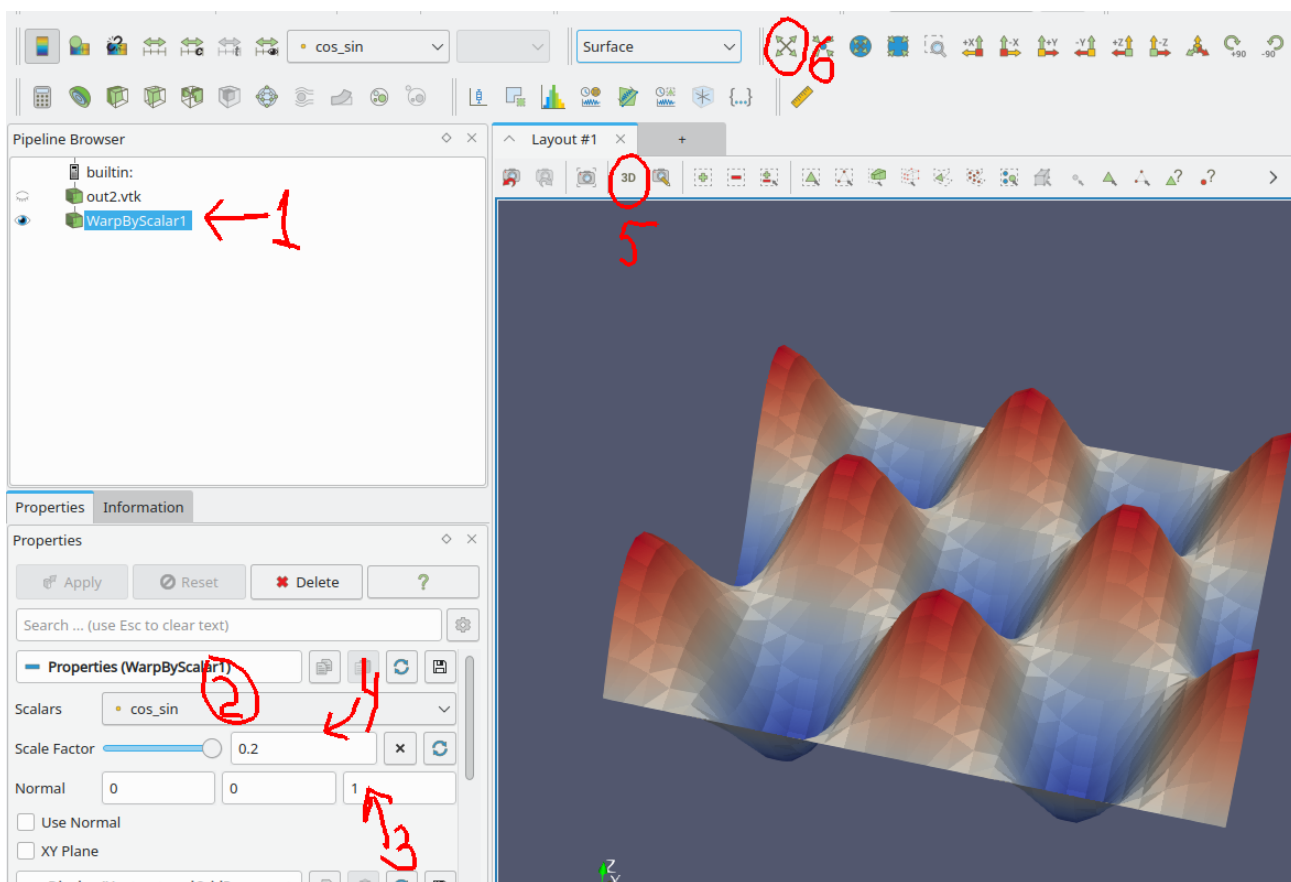
**Задание цвета и толщины изолинии** В случае, если нужно сделать изолинии одного цвета, установите поле **Coloring/Solid color** в настройках фильтра. Там же в меню **Edit** можно выбрать цвет. Для установления толщины линии включите подробные настройки и найдите там опцию **Styling/Line Width**.



### D.6.3 Данные на двумерных сетках в виде поверхности

По аналогии с одномерным графиком (п. D.6.1), двумерные поля так же можно отобразить, проектируя данные на геометрическую координату для получения объёмного графика. Для этого

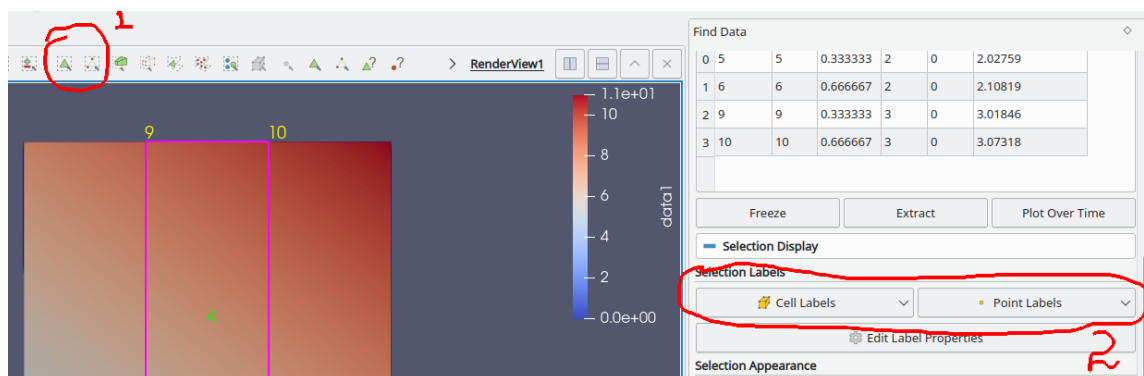
1. Включите фильтр **Filters/Warp By Scalar**
2. В настройках фильтра установите данные, которые будут проектироваться на координату  $z$
3. Установите нормаль для проецирования (ось  $z$ )
4. Если нужно, выберите масштабирования для этой координаты
5. После нажатия **Apply** включите трёхмерное отображение
6. Если данные не видно, обновите экран.



#### D.6.4 Числовых значения в точках и ячейках

Иногда в процессе отладки или анализа результатов расчёта требуется знать точное значение поля в заданном узле или ячейке сетки. Для этого

1. Включить режим выделения точек или ячеек (иконка (1 на рисунке) или горячие клавиши **s**, **d**). Выделить мышкой интересующую область
2. В окне **Find data** (или **Selection Inspector** для старых версий Paraview) отметить поле, которое должно отображаться в центрах ячеек и в точках (2 на рисунке). Если такого окна нет, включить его из основного меню **View**.

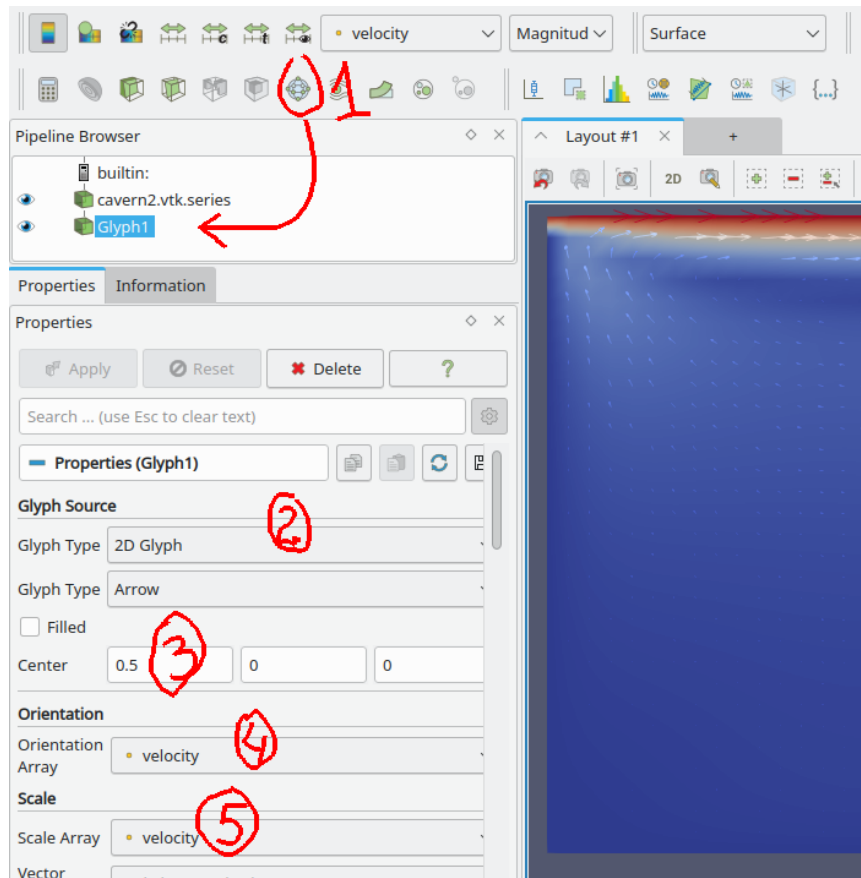


#### D.6.5 Векторные поля

Открыть файл **vtk** или **vtk.series**, который содержит векторное поле. Далее

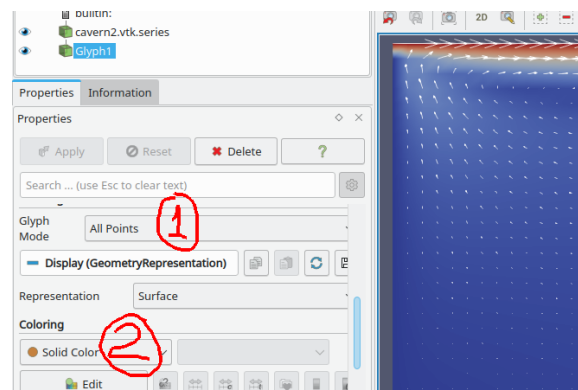
1. Создать фильтр **Glyph**

2. Задать двумерный тип стрелки
3. Сместить центр стрелки, чтобы она исходила из точки, к которой приписана
4. Отметить необходимое векторное поле в качестве ориентации
5. Отметить необходимое векторное поле для масштабирования Нажать **Apply**.



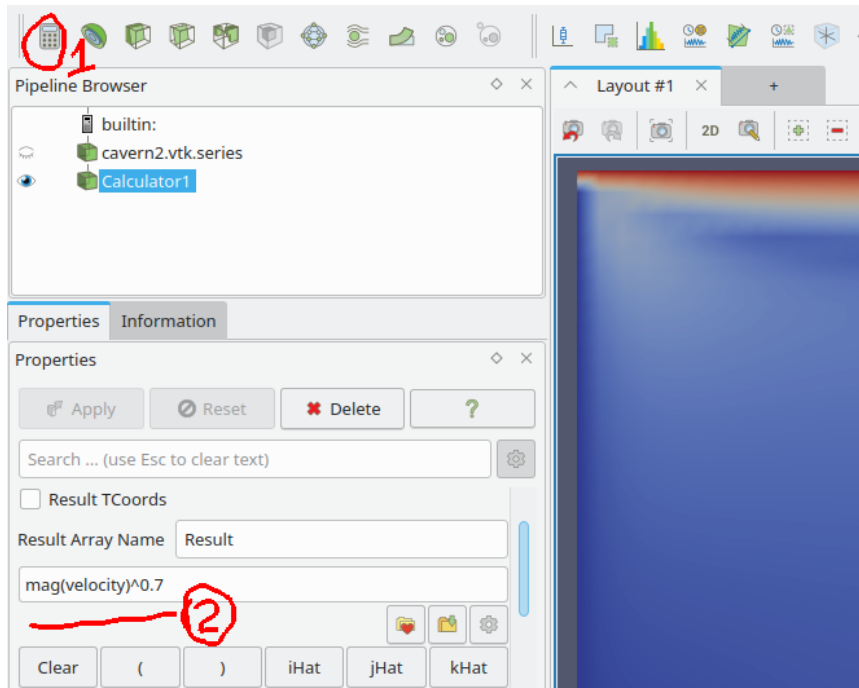
## Настройка отображения стрелок

1. Выбрать необходимый **Glyph-mode**. Если сетка небольшая, то можно **All Points**.
2. Установить белый цвет для стрелок. Нажать Apply.



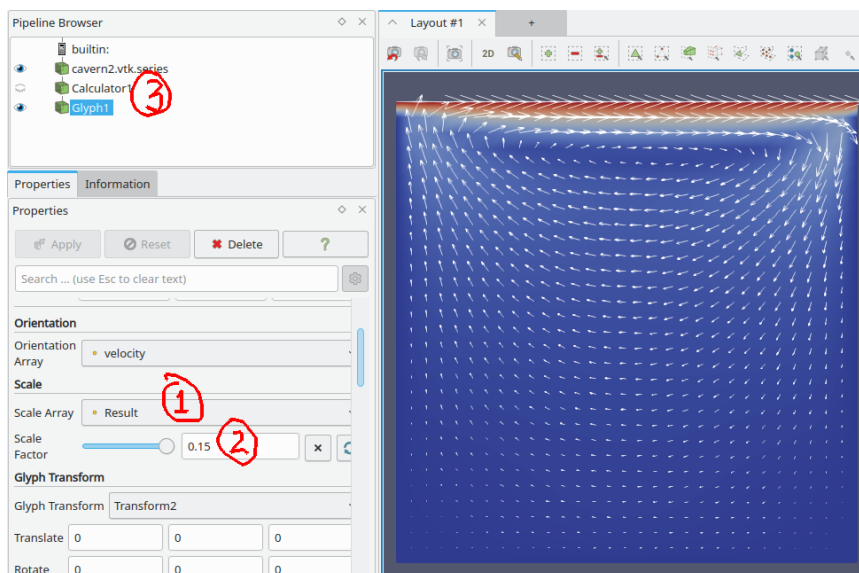
**Уменьшения разброса по длине стрелок** Если разброс по длинам стрелок слишком велик, его можно подравнять, введя новую функцию  $|\mathbf{v}|^\alpha$  – длина вектора в степени меньше единицы (например,  $\alpha = 0.7$ ). Такую функцию можно создать через калькулятор

1. Начиная от загруженного файла создать фильтр **Calculator**
2. Там вбить необходимую формулу



Созданную функцию нужно прокинуть в **Glyph** в качестве коэффициента масштабирования

1. В **Scale Array** фильтра **Glyph** указать уже результат работы **Calculator**-а (**Result** по умолчанию),
2. Подтянуть значение **Scale Factor** до приемлимого
3. Не забыть отключить вспомогательное поле **Calculator** из отображения

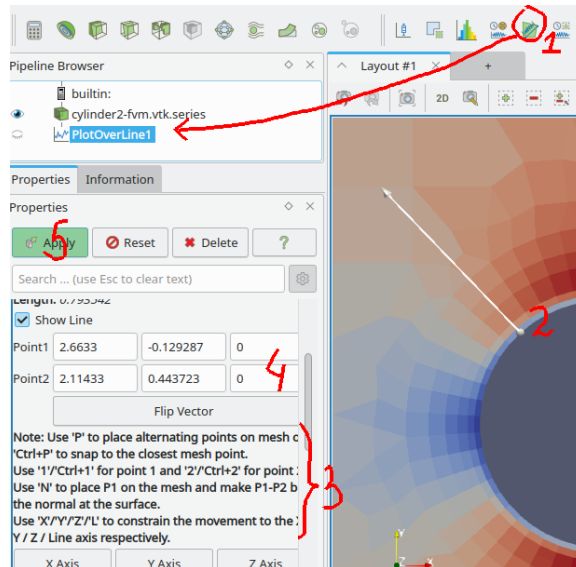


## D.6.6 Значение функции вдоль линии

1. Выбрать фильтр **Plot Over Line** иконкой или в меню **Filters**
2. Установить начальную и конечную точку сечения

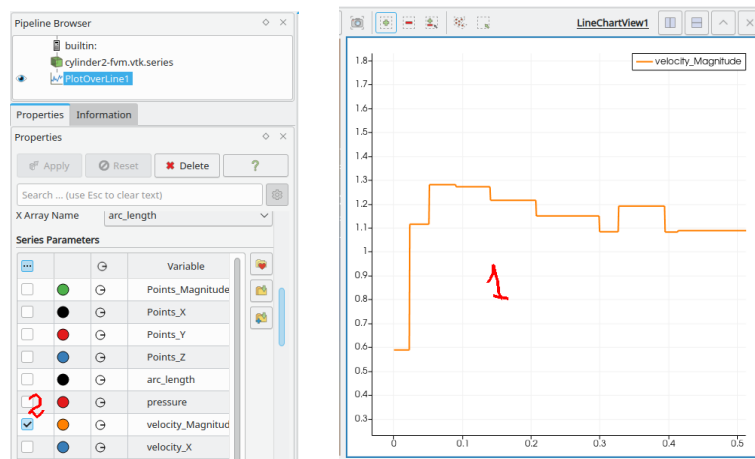


3. Можно использовать привязку к узлам сетки с помощью горячих клавиш (в подсказках написано)
4. Можно установить координаты руками в соответствующем поле. Для двумерных задач проследить, что координата Z равна нулю
5. Нажать **Apply**



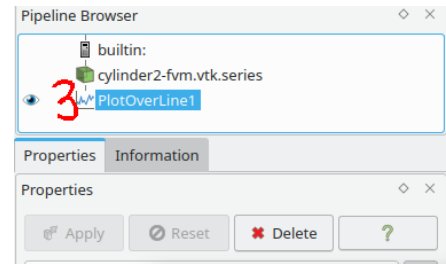
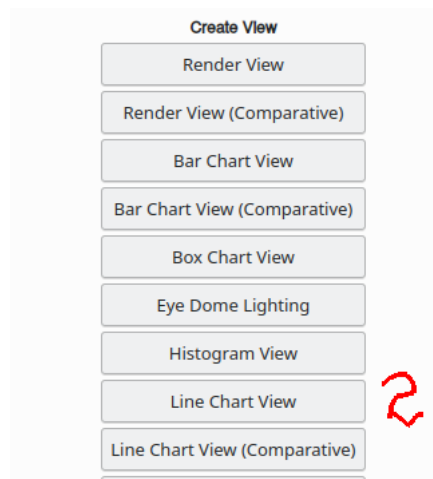
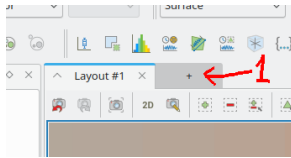
## Настройка графика

1. После установок появится дополнительное окно типа **Line Chart View** с нарисованным графиком.
2. Сделав это окно активным в настройках фильтра **PlotOverLine** можно выбрать, какие поля рисовать (**Series Parameters**)



## Отрисовка в отдельном окне

1. Открыть новую вкладку
2. Выбрать **Line Chart View**
3. Выбрать предварительно созданный фильтр с одномерным графиком



## D.7 Hybmesh

Генератор сеток на основе композитного подхода. Работает на основе python-скриптов. Полная документация <http://kalininei.github.io/HybMesh/index.html>

### D.7.1 Построение сеток

Скрипты построения сетки лежат в папке `/app/test-data`. Для запуска скрипта построения сетки следует находясь в контейнере перейти в эту папку и запустить (например для `trigrid.py`)

```
cd /app/test-data
hybmesh -sx trigrid.py
```

При первом запуске система предложит собрать и установить `hybmesh` из исходников. Следует согласиться, введя `y`. Первое построение может занять несколько минут.