



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ



НЭТИ

Кафедра прикладной математики

Курсовой проект
по дисциплине «Численные методы»



ФПМИ

ГРУППА

ПМ-92

ВАРИАНТ

21

СТУДЕНТ

ГЛУШКО ВЛАДИСЛАВ

ПРЕПОДАВАТЕЛИ

ПАТРУШЕВ ИЛЬЯ ИГОРЕВИЧ

СОЛОВЕЙЧИК ЮРИЙ ГРИГОРЬЕВИЧ

Новосибирск

1 Условие задачи

Формулировка задачи

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции линейные на треугольниках. Краевые условия всех типов. Коэффициент диффузии λ разложить по линейным базисным функциям. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать метод сопряженных градиентов (МСГ) или локально-оптимальную схему (ЛОС) с неполной факторизацией.

Постановка задачи

Эллиптическая краевая задача для функции u определяется дифференциальным уравнением:

$$-div(\lambda \operatorname{grad} u) + \gamma u = f$$

заданным в некоторой области Ω с границей $S = S_1 \cup S_2 \cup S_3$ и краевыми условиями:

$$\begin{aligned} u|_{S_1} &= u_g \\ \lambda \frac{\partial u}{\partial n} \Big|_{S_2} &= \theta \\ \lambda \frac{\partial u}{\partial n} \Big|_{S_3} + \beta(u|_{S_3} - u_\beta) &= 0 \end{aligned}$$

В декартовой системе координат x, y это уравнение может быть записано в виде

$$-\frac{\partial}{\partial x} \left(\lambda \frac{\partial u}{\partial x} \right) - \frac{\partial}{\partial y} \left(\lambda \frac{\partial u}{\partial y} \right) + \gamma u = f$$

Исходное уравнение можно переписать в виде: $Lu = f$, где $Lu = -div(\lambda \operatorname{grad} u) + \gamma u$. Тогда чтобы решить исходную задачу следует левую и правую части уравнения домножить на функцию v из пространства пробных функций и проинтегрировать по Ω . Фактически это соответствует скалярному умножению Lu и f на v в пространстве $L_2(\Omega)$:

$$\begin{aligned} (Lu, v) &= (f, v) \\ (Lu - f, v) &= 0 \end{aligned}$$

Это уравнение Галеркина в слабой форме.

В пространстве $L_2(\Omega)$ скалярное произведение вычисляется по формуле:

$$(u, v) = \int_{\Omega} uv d\Omega$$

Перепишем уравнение Галеркина в явном виде:

$$-\int_{\Omega} div(\lambda \operatorname{grad} u) v d\Omega + \int_{\Omega} \gamma uv d\Omega - \int_{\Omega} f v d\Omega = 0$$

Воспользуемся формулой Грина:

$$\int_{\Omega} (\lambda \operatorname{grad} u \operatorname{grad} v) d\Omega = - \int_{\Omega} \operatorname{div}(\lambda \operatorname{grad} u) v d\Omega + \int_S \lambda \frac{\partial u}{\partial n} v dS$$

Сделав соответствующую подстановку, получим уравнение вида:

$$\int_{\Omega} (\lambda \operatorname{grad} u \operatorname{grad} v) d\Omega - \int_S \lambda \frac{\partial u}{\partial n} v dS + \int_{\Omega} \gamma uv d\Omega - \int_{\Omega} f v d\Omega = 0$$

Учитывая краевые условия $S = S_1 \cup S_2 \cup S_3$, получим:

$$\int_{\Omega} (\lambda \operatorname{grad} u \operatorname{grad} v) d\Omega - \int_{S_1} \lambda \frac{\partial u}{\partial n} v dS - \int_{S_2} \theta v dS - \int_{S_3} \beta(u|_{\beta} - u) v dS + \int_{\Omega} \gamma uv d\Omega - \int_{\Omega} f v d\Omega = 0$$

Так как $v|_{S_1} = 0$, то

$$\int_{S_1} \lambda \frac{\partial u}{\partial n} v dS = 0$$

Уравнение примет вид:

$$\int_{\Omega} (\lambda \operatorname{grad} u \operatorname{grad} v) d\Omega + \int_{S_3} \beta uv dS + \int_{\Omega} \gamma uv d\Omega = \int_{S_2} \theta v dS + \int_{S_3} \beta u|_{\beta} v dS + \int_{\Omega} f v d\Omega = 0$$

Будем искать решение в виде:

$$u = \sum_{i=1}^n q_i \psi_i$$

где ψ_i - базисные функции. Функция v может быть представлена в таком же виде. Подставив, получим СЛАУ для компонент q_i :

$$\begin{aligned} \sum_{i=1}^n q_i \left(\int_{\Omega} (\lambda \operatorname{grad} \psi_i \operatorname{grad} \psi_j) d\Omega + \int_{S_3} \beta \psi_i \psi_j dS + \int_{\Omega} \gamma \psi_i \psi_j d\Omega \right) = \\ = \int_{S_2} \theta \psi_j dS + \int_{S_3} \beta u_{\beta} \psi_j dS + \int_{\Omega} f \psi_j d\Omega \end{aligned}$$

Поскольку исходная задача рассматривается в декартовой системе координат, то $\operatorname{grad} u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \right)$ и, соответственно: $\operatorname{grad} u \operatorname{grad} v = \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial v}{\partial y}$. Отсюда получаем уравнение в виде:

$$\begin{aligned} \sum_{j=1}^n q_j \int_{\Omega} \lambda \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) dxdy + \sum_{j=1}^n q_j \int_{\Omega} \gamma \psi_j \psi_i dxdy + \sum_{j=1}^n q_j \int_{\Omega} \beta \psi_j \psi_i dxdy = \\ = \int_{\Omega} f \psi_i dxdy + \int_{S_2} \theta \psi_i dxdy + \int_{S_3} \beta u_{\beta} \psi_i dxdy \end{aligned}$$

Конечноэлементная дискретизация

Так как для решения задачи используются линейные базисные функции, то на каждом конечном элементе Ω_k - треугольнике эти функции будут совпадать с функциями $L_1(x, y)$, $L_2(x, y)$, $L_3(x, y)$, такими, что $L_1(x, y)$ равна единице в вершине (x_1, y_1) и нулю во всех остальных вершинах, $L_2(x, y)$ равна единице в вершине (x_2, y_2) и нулю во всех остальных вершинах, $L_3(x, y)$ равна единице в вершине (x_3, y_3) и нулю во всех остальных вершинах. Любая линейная на Ω_k функция представима в виде линейной комбинации этих базисных линейных функций, коэффициентами будут значения функции в каждой из вершин треугольника Ω_k . Таким образом, на каждом конечном элементе нам понадобятся три узла – вершины треугольника.

$$\psi_1 = L_1(x, y)$$

$$\psi_2 = L_2(x, y)$$

$$\psi_3 = L_3(x, y)$$

Учитывая построение *L-функций*, получаем следующие соотношения:

$$\begin{cases} L_1 + L_2 + L_3 = 1 \\ L_1x_1 + L_2x_2 + L_3x_3 = x \\ L_1y_1 + L_2y_2 + L_3y_3 = y \end{cases}$$

Т.е. имеем систему:

$$\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} \cdot \begin{pmatrix} L_1 \\ L_2 \\ L_3 \end{pmatrix} = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix}$$

Отсюда находим коэффициенты линейных функций $L_1(x, y)$, $L_2(x, y)$, $L_3(x, y)$

$$L_i = a_0^i + a_1^i x + a_2^i y, i = \overline{1, 3}$$

$$\begin{pmatrix} \alpha_0^1 & \alpha_1^1 & \alpha_2^1 \\ \alpha_0^2 & \alpha_1^2 & \alpha_2^2 \\ \alpha_0^3 & \alpha_1^3 & \alpha_2^3 \end{pmatrix} = D^{-1} = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1}$$

$$D^{-1} = \frac{1}{|\det D|} \begin{pmatrix} x_2y_3 - x_3y_2 & y_2 - y_3 & x_3 - x_2 \\ x_3y_1 - x_1y_3 & y_3 - y_1 & x_1 - x_3 \\ x_1y_2 - x_2y_1 & y_1 - y_2 & x_2 - x_1 \end{pmatrix}$$

Переход к локальным матрицам

Чтобы получить выражения для локальных матриц жёсткости G и массы M каждого конечного элемента Ω_K , перейдём к решению локальной задачи на каждом конечном элементе. Полученное уравнение для области Ω представим в виде суммы интегралов по областям Ω_k без учёта краевых условий. Тогда на каждом конечном элементе будем решать локальную задачу построения матриц жёсткости, массы и вектора правой части.

$$\int_{\Omega_k} \lambda \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) dx dy + \int_{\Omega_k} \gamma \psi_j \psi_i ds dy = \int_{\Omega_k} f \psi_i dx dy$$

Локальная матрица будет представлять собой сумму матриц жёсткости и массы и будет иметь размерность 3×3 (по числу узлов на конечном элементе)

Построение матрицы массы

$$M_{ii} = \int_{\Omega_k} \gamma (\psi_i)^2 d\Omega_k = \gamma \int_{\Omega_k} (L_i)^2 d\Omega_k = \gamma \frac{2!0!0!}{(2+0+0+2)!} |det D| = \gamma \frac{det D}{12}, i = 1, \dots, 3$$

$$M_{ij} = \int_{\Omega_k} \gamma \psi_i \psi_j d\Omega_k = \gamma \int_{\Omega_k} L_i L_j d\Omega_k = \gamma \frac{1!1!0!}{(1+1+0+2)!} |det D| = \gamma \frac{det D}{24}, i \neq j$$

$$M = \gamma \frac{|det D|}{24} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Построение матрицы жёсткости

$$G_{ij} = \int_{\Omega_k} \lambda \left(\frac{\partial \psi_i}{\partial x} \frac{\partial \psi_j}{\partial x} + \frac{\partial \psi_i}{\partial y} \frac{\partial \psi_j}{\partial y} \right) d\Omega_k = \left| \lambda = \sum_{k=1}^3 \lambda_k \psi_k \right| =$$

$$= \lambda_1 \int_{\Omega_k} \psi_1 (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) d\Omega_k + \lambda_2 \int_{\Omega_k} \psi_2 (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) d\Omega_k + \lambda_3 \int_{\Omega_k} \psi_3 (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) d\Omega_k =$$

$$= (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) \left(\lambda_1 \int_{\Omega_k} \psi_1 d\Omega_k + \lambda_2 \int_{\Omega_k} \psi_2 d\Omega_k + \lambda_3 \int_{\Omega_k} \psi_3 d\Omega_k \right) =$$

$$= (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) \left(\lambda_1 \int_{\Omega_k} L_1 dx dy + \lambda_2 \int_{\Omega_k} L_2 dx dy + \lambda_3 \int_{\Omega_k} L_3 dx dy \right) =$$

$$= (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) (\lambda_1 + \lambda_2 + \lambda_3) |det D| \left(\frac{1!0!0!}{(1+0+0+2)!} + \frac{0!1!0!}{(0+1+0+2)!} + \frac{0!0!1!}{(0+0+1+2)!} \right) =$$

$$= (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) \left(\frac{(\lambda_1 + \lambda_2 + \lambda_3) |det D|}{6} \right)$$

Построение вектора правой части

Рассмотрим правую часть выражения для k -го конечного элемента:

$$\int_{\Omega_k} f \psi_i dx dy$$

представим f в виде $f_1 L_1 + f_2 L_2 + f_3 L_3$, где f_i - значения в вершинах треугольника. Получим:

$$\int_{\Omega_k} f_q L_q L_i dx dy = f_q \int_{\Omega_k} L_q L_i d\Omega_k$$

Таким образом:

$$B_i = \sum_{q=1}^3 f_q \int_{\Omega_k} L_q L_i d\Omega_k \quad i = \overline{0, 2}$$

Сборка глобальной матрицы и глобального вектора

При формировании глобальной матрицы из локальных, полученных суммированием соответствующих матриц массы и жесткости, учитываем соответствие локальной и глобальной нумераций каждого конечного элемента. Глобальная нумерация каждого конечного элемента однозначно определяет позиции вклада его локальной матрицы в глобальную. Поэтому, зная глобальные номера соответствующих узлов конечного элемента, определяем и то, какие элементы глобальной матрицы изменятся при учете текущего конечного элемента. Аналогичным образом определяется вклад локального вектора правой части в глобальный. При учете текущего локального вектора изменятся те элементы глобального вектора правой части, номера которых совпадают с глобальными номерами узлов, присутствующих в этом конечном элементе.

Учёт первых краевых условий

Для учета первых краевых условий, в глобальной матрице и глобальном векторе находим соответствующую глобальному номеру краевого узла строку и зануляем всё кроме диагонального элемента, которому присваиваем 1, а вместо элемента с таким номером в векторе правой части - значение краевого условия, заданное в исходной задаче.

Учёт вторых и третьих краевых условий

Рассмотрим краевые условия второго и третьего рода:

$$\lambda \frac{\partial u}{\partial n} \Big|_{S_2} = \theta$$

$$\lambda \frac{\partial u}{\partial n} \Big|_{S_3} + \beta(u|_{S_3} - u_\beta) = 0$$

Отсюда получаем, что для учёта краевых условий необходимо вычислить интегралы:

$$\int_{S_2} \theta \psi_j dx dy, \quad \int_{S_3} \beta u_\beta \psi_j dx dy, \quad \int_{S_3} \beta \psi_i \psi_j dx dy$$

Краевые условия второго и третьего рода задаются на рёбрах, т.е. определяются двумя узлами, лежащими на ребре. Будем считать, что параметр β на S_3 постоянен, тогда параметр u_β будем раскладывать по двум базисным функциям, определённым на этом ребре:

$$u_\beta = u_{\beta 1} \phi_1 + u_{\beta 2} \phi_2$$

где ϕ_i , $i = \overline{0, 1}$ - локально занумерованные линейные базисные функции, которые имеют также свои глобальные номера во всей расчетной области, а $u_{\beta i}$ - значение функции u_β в узлах ребра.

Аналогично поступаем и при учете вторых краевых условий, раскладывая по базису ребра функцию:

$$\theta = \theta_0 \phi_0 + \theta_1 \phi_1$$

Тогда приведенные выше интегралы примут вид:

$$\begin{aligned} I_1 &= \int_{S_2} (\theta_0 \phi_0 + \theta_1 \phi_1) \phi_i dx dy \\ I_2 &= \beta \int_{S_3} (u_{\beta 1} \phi_0 + u_{\beta 2} \phi_1) \phi_i dx dy \\ I_3 &= \beta \int_{S_3} \phi_i \phi_j dx dy \end{aligned}$$

Фактически, решая задачу учета краевых условий второго и третьего рода, мы переходим к решению одномерной задачи на ребре для того, чтобы занести соответствующие результаты в глобальную матрицу и вектор.

Базисными функциями ребра являются две ненулевые на данном ребре базисные функции из ϕ_i , $i = \overline{0, 1}$ конечного элемента.

Для учёта вклада вторых и третьих краевых условий рассчитываются 2 матрицы 2×2 .

Интегралы I_1, I_2, I_3 будем вычислять по формуле:

$$\int (L_i)^{v_i} (L_j)^{v_j} dS = \frac{v_i! v_j!}{(v_i + v_j + 1)!} mes \Gamma, \quad i \neq j$$

где $mes \Gamma$ длина ребра. При этом независимо от того, что на каждом из ребер присутствуют свои функции, интегралы, посчитанные по приведенным выше формулам, будут равны:

$$I_1 = \begin{pmatrix} \int_{S_2} L_1 L_1 dx dy & \int_{S_2} L_1 L_2 dx dy \\ \int_{S_2} L_2 L_1 dx dy & \int_{S_2} L_2 L_2 dx dy \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = \frac{1}{6} mes S_2 \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$$

Этот вектор поправок в правую часть позволяет учесть не только вторые краевые условия, но и часть βu_β из третьих. Осталось рассмотреть матрицу поправок в левую часть:

$$I_3 = \beta \int_{S_3} \phi_i \phi_j dx dy$$

Очевидно, что получится та же матрица, только не умноженная на вектор констант:

$$I_3 = \frac{1}{6} mes S_3 \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Добавляя эту матрицу в левую часть, на места соответствующие номерам узлов, получаем учет третьих краевых условий.

2 Текст программы

Ссылка к полному проэкту: <https://github.com/ISTECTION/FEM>

main.cpp

```
1  #include "argparse/argparse.hpp"
2  #include "timer/cxxtimer.hpp"
3  #include "FEM.hpp"
4
5  #include <iostream>
6  #include <optional>
7  #include <fstream>
8
9  int main(int argc, char* argv[]) {
10     std::ios_base::sync_with_stdio(false);
11
12     using namespace      ::Log;
13     using ::std::chrono::milliseconds;
14
15     argparse::ArgumentParser _prs("FEM", "1.0.0");
16     _prs.add_argument("-i", "--input")
17         .help( "path to input files" )
18         .required();
19
20     _prs.add_argument("-o", "--output")
21         .help( "path to output files" );
22
23     _prs.add_argument("-d", "--debug")
24         .help("debugging information")
25         .default_value(false)
26         .implicit_value(true);
27
28     try {
29         using std::string;
30
31         _prs.parse_args(argc, argv);
32         std::optional      _opt = _prs.present      ("-o");
33         std::filesystem::path _inp = _prs.get<string>("-i");
34         std::filesystem::path _out = _opt.has_value()
35             ? _prs.get<string>("-o")
36             : _inp / "sparse";
37
38         Function::setFunction(_inp.string());
39         cxxtimer::Timer _timer(true);
40         FEM _FEM(_inp, _prs.get<bool>("-d"));
41         _FEM.startLOS(_out);
42         _timer.stop();
43
44         if (_prs["-d"] == true) {
```

```

45         _FEM.printAll();
46         _FEM.printSparse();
47         _FEM.createTable();
48     } else
49         _FEM.printZ();
50
51     std::cout << _timer;
52 } catch(const std::runtime_error& err) {
53     #define ARGUMENTS_NO_RECEIVED 2
54     Logger::append(getLog(
55         "argc != 2 (FEM --input ./input)"
56     ));
57     std::cerr << err.what() << std::endl;
58     std::cerr << _prs << std::endl;
59     std::exit( ARGUMENTS_NO_RECEIVED );
60 }
61 return 0;
62 }

```

Union.hpp

```

1  #include <vector>
2  #include <array>
3
4  namespace Union {
5      struct XY      { double x, y;          };
6      struct Material { double betta, gamma; };
7
8      struct Element {
9          size_t area;
10         std::array<size_t, 3> nodeIdx;
11     };
12
13     struct Boundary {
14         size_t cond, type, area;
15         std::array<size_t, 2> nodeIdx;
16     };
17
18     struct Param {
19         size_t nodes;
20         size_t elems;
21         size_t areas;
22         size_t conds;
23     };
24 }

```

FEM.hpp

```
1  #ifndef _FEM_HPP_
2  #define _FEM_HPP_
3  #include "indicators/indicators.hpp"
4  #include "utils/lightweight.hpp"
5  #include "utils/overload.hpp"
6  #include "utils/friendly.hpp"
7  #include "nlohmann/json.hpp"
8  #include "Function.hpp"
9  #include "LOS/LOS.hpp"
10 #include "Logger.hpp"
11 #include "Union.hpp"
12
13 #include <algorithm>
14 #include <chrono>
15 #include <thread>
16 #include <cmath>
17 #include <set>
18
19 class FEM
20 {
21 private:
22     Union::Param _size;
23
24     std::vector<Union::XY>      nodes;
25     std::vector<Union::Element> elems;
26     std::vector<Union::Boundary> boundarys;
27     std::vector<Union::Material> materials;
28
29     std::vector<double> gb;
30     std::vector<double> gg;
31     std::vector<double> di;
32     std::vector<size_t> ig;
33     std::vector<size_t> jg;
34
35     bool _debugging;
36 public:
37     FEM(std::filesystem::path _path, bool debugging)
38         : _debugging(debugging) {
39
40         assert(readJson(_path));
41         portrait(true);
42         global();
43         boundaryCondition();
44     }
45
46     void startLOS(const std::filesystem::path& _out) {
47         using ::Cond::HOLLESKY;
48         using ::Cond::DIAGONAL;
```

```

49     using ::Cond::NONE;
50
51     const double _eps = 1E-20;
52     const double _itr = 10000;
53     Cond _cond = DIAGONAL;
54
55     LOS<double> _LOS (getDate(), getNodes(), _eps, _itr);
56     _LOS.solve(_cond, true);
57     _z = std::move(_LOS.getX());
58 }
59
60 double getValue    (double, double) const;
61 private:
62     void global();
63     void resize();
64
65     array::x    localB(const std::array<Union::XY, 3>&, size_t) const;
66     array::xxx localA(const std::array<Union::XY, 3>&, size_t) const;
67
68     array::xxx G(const std::array<Union::XY, 3>&, size_t) const;
69     array::xxx M(const std::array<Union::XY, 3>&, size_t) const;
70
71     bool readFile(const std::filesystem::path& );
72     bool readJson(const std::filesystem::path& );
73     void portrait(const bool isWriteList = false);
74
75     void boundaryCondition();
76     void first (const Union::Boundary& bound);
77     void second(const Union::Boundary& bound);
78     void third (const Union::Boundary& bound);
79
80     template<size_t N,
81             typename _Struct>
82     void loc_A_to_global(
83         const std::array<std::array<double, N>, N>&,
84         const _Struct&
85     );
86
87     template<size_t N,
88             typename _Struct>
89     void loc_b_to_global(
90         const std::array<double, N>&,
91         const _Struct&
92     );
93 };
94
95 void FEM::portrait(const bool isWriteList) {
96
97     const size_t N {    _size.nodes    };

```

```

98     std::vector<std::set<size_t>> list(N);
99
100    for (size_t el = 0; el < _size.elems; el++)
101    for (size_t point = 0; point < 3; point++) {
102        for (size_t i = point + 1; i < 3; i++) {
103            size_t idx1 = { elems[el].nodeIdx[point] };
104            size_t idx2 = { elems[el].nodeIdx[ i ] };
105            idx1 > idx2 ?
106                list[idx1].insert(idx2) :
107                list[idx2].insert(idx1) ;
108        }
109    }
110
111    for (size_t i = 2; i < ig.size(); i++)
112        ig[i] = ig[i - 1] + list[i - 1].size();
113
114    jg.resize(ig[N]);
115    gg.resize(ig[N]);
116
117    for (size_t index = 0, i = 1; i < list.size(); i++)
118    for (size_t value : list[i])
119        jg[index++] = value;
120 }
121
122 void FEM::global() {
123     std::array<Union::XY, 3> coords;
124     for (size_t i = 0; i < _size.elems; i++) {
125         for (size_t j = 0; j < 3; j++) {
126             size_t point = elems[i].nodeIdx[j];
127             coords[j].x = nodes[point].x;
128             coords[j].y = nodes[point].y;
129         }
130         array::x    local_b = localB(coords, elems[i].area);
131         array::xxx  local_A = localA(coords, elems[i].area);
132
133         loc_A_to_global<3>(local_A, elems[i]);
134         loc_b_to_global<3>(local_b, elems[i]);
135     }
136 }
137
138 template<size_t N, typename _Struct>
139 void FEM::loc_A_to_global(
140     const std::array<std::array<double, N>, N>& locA,
141     const _Struct& elem) {
142
143     using          ::std::vector;
144     using iterator = ::std::vector<size_t>::iterator;
145
146     for (size_t i = 0; i < N; i++) {

```

```

147         di[elem.nodeIdx[i]] += locA[i][i];
148
149         for (size_t j = 0; j < i; j++) {
150             size_t a = elem.nodeIdx[i];
151             size_t b = elem.nodeIdx[j];
152             if (a < b) std::swap(a, b);
153
154             iterator _beg = jg.begin() + ig[a];
155             iterator _end = jg.begin() + ig[a + 1];
156
157             auto _itr = std::lower_bound(_beg, _end, b);
158             auto _idx = _itr - jg.begin();
159             gg[_idx] += locA[i][j];
160         }
161     }
162 }
163
164 template<size_t N, typename _Struct>
165 void FEM::loc_b_to_global(
166     const std::array<double, N>& loc_b,
167     const _Struct& elem) {
168
169     for (size_t i = 0; i < N; i++)
170         gb[elem.nodeIdx[i]] += loc_b[i];
171 }
172
173 array::x
174 FEM::localB(const std::array<Union::XY, 3>& elem, size_t area) const {
175     std::array<double, 3> function {
176         Function::f(elem[0], area),
177         Function::f(elem[1], area),
178         Function::f(elem[2], area)
179     };
180
181     double det_D = fabs(determinant(elem));
182     double _koef = det_D / 24;
183
184     return {
185         _koef * (2 * function[0] + function[1] + function[2]),
186         _koef * (2 * function[1] + function[0] + function[2]),
187         _koef * (2 * function[2] + function[0] + function[1])
188     };
189 }
190
191 array::xxx
192 FEM::localA(const std::array<Union::XY, 3>& elem, size_t area) const {
193     std::array<std::array<double, 3>, 3> G = FEM::G(elem, area);
194     std::array<std::array<double, 3>, 3> M = FEM::M(elem, area);
195     std::array<std::array<double, 3>, 3> A = G + M;

```

```

196     return A;
197 }
198
199 array::xxx
200 FEM::G(const std::array<Union::XY, 3>& elem, size_t area) const {
201     double det = fabs(determinant(elem));
202     double _koef = (
203         Function::lambda(elem[0], area) +
204         Function::lambda(elem[1], area) +
205         Function::lambda(elem[2], area)
206     ) / (det * 6);
207
208     std::array<std::array<double, 3>, 3> G;
209     std::array<std::array<double, 2>, 3> a {
210         elem[1].y - elem[2].y,
211         elem[2].x - elem[1].x,
212
213         elem[2].y - elem[0].y,
214         elem[0].x - elem[2].x,
215
216         elem[0].y - elem[1].y,
217         elem[1].x - elem[0].x
218     };
219
220     for (int i = 0; i < 3; i++)
221     for (int j = 0; j < 3; j++)
222         G[i][j] = _koef * (
223             a[i][0] * a[j][0] +
224             a[i][1] * a[j][1]
225         );
226
227     return G;
228 }
229
230 array::xxx
231 FEM::M(const std::array<Union::XY, 3>& elem, size_t area) const {
232     double det = fabs(determinant(elem));
233     double gammaKcoef = materials[area].gamma * det / 24;
234     std::array<std::array<double, 3>, 3> M;
235     for (size_t i = 0; i < 3; i++)
236     for (size_t j = 0; j < 3; j++) {
237         M[i][j] =
238             (i == j) ? (2 * gammaKcoef) :
239             (gammaKcoef);
240     }
241     return M;
242 }
243
244 void FEM::boundaryCondition() {

```

```

245     using namespace ::Log;
246
247     std::sort(
248         boundarys.begin(),
249         boundarys.end(),
250         [](Union::Boundary& _left, Union::Boundary& _right){
251             return _left.cond > _right.cond;
252         }
253     );
254
255     for (size_t _count = 0; _count < _size.conds; _count++) {
256         switch (boundarys[_count].cond)
257         {
258             case FIRST_BOUNDARY_COND:
259                 first(boundarys[_count]);
260                 break;
261             case SECOND_BOUNDARY_COND:
262                 second(boundarys[_count]);
263                 break;
264             case THIRD_BOUNDARY_COND:
265                 third(boundarys[_count]);
266                 break;
267             default:
268                 Logger::append(getLog("There is no such condition"));
269         }
270     }
271 }
272
273 void FEM::first(const Union::Boundary& bound) {
274     di[bound.nodeIdx[0]] = { 1 };
275     di[bound.nodeIdx[1]] = { 1 };
276
277     for (size_t i = 0; i < 2; i++)
278         gb[bound.nodeIdx[i]] =
279             Function::firstBound({
280                 nodes[bound.nodeIdx[i]].x,
281                 nodes[bound.nodeIdx[i]].y
282             }, bound.type);
283
284     for (size_t k = 0; k < 2; k++) {
285         size_t node = bound.nodeIdx[k];
286         for (size_t i = ig[node]; i < ig[node + 1]; i++) {
287             if(di[jg[i]] != 1)
288                 gb[jg[i]] -= gg[i] * gb[node];
289             gg[i] = 0;
290         }
291
292         for(size_t i = node + 1; i < _size.nodes; i++) {
293             size_t lbeg = ig[i];

```



```

294         size_t lend = ig[i + 1];
295         for(size_t p = lbeg; p < lend; p++) {
296             if(jg[p] == node) {
297                 if(di[i] != 1)
298                     gb[i] -= gg[p] * gb[node];
299                 gg[p] = 0;
300             }
301         }
302     }
303 }
304 }
305
306 void FEM::second(const Union::Boundary& bound) {
307     std::array<Union::XY, 2>
308         coord_borders = {
309         nodes[bound.nodeIdx[0]],
310         nodes[bound.nodeIdx[1]]
311     };
312
313     double _koef =
314         edgeLength(coord_borders) / 6;
315
316     std::array<double, 2> corr_b;
317     for (size_t i = 0; i < 2; i++)
318         corr_b[i] = _koef * (
319             2 * Function::secondBound({
320                 nodes[bound.nodeIdx[i]].x,
321                 nodes[bound.nodeIdx[i]].y
322             }, bound.type) +
323             Function::secondBound({
324                 nodes[bound.nodeIdx[1 - i]].x,
325                 nodes[bound.nodeIdx[1 - i]].y
326             }, bound.type)
327         );
328
329     loc_b_to_global<2>(corr_b, bound);
330 }
331
332 void FEM::third(const Union::Boundary& bound) {
333     std::array<Union::XY, 2> coord_borders = {
334         nodes[bound.nodeIdx[0]],
335         nodes[bound.nodeIdx[1]]
336     };
337
338     double _koef =
339         materials[bound.area].betta *
340         edgeLength(coord_borders) / 6;
341
342     std::array<double, 2> corr_b;

```

```

343     std::array<std::array<double, 2>, 2> corr_a;
344
345     for (size_t i = 0; i < 2; i++) {
346         corr_b[i] = _koef * (
347             2 * Function::thirdBound({
348                 nodes[bound.nodeIdx[i]].x,
349                 nodes[bound.nodeIdx[i]].y
350             }, bound.type) +
351             Function::thirdBound({
352                 nodes[bound.nodeIdx[1 - i]].x,
353                 nodes[bound.nodeIdx[1 - i]].y
354             }, bound.type)
355         );
356
357         for (size_t j = 0; j < 2; j++) {
358             corr_a[i][j] =
359                 (i == j) ? (2 * _koef) :
360                 (_koef);
361         }
362     }
363     loc_b_to_global<2>(corr_b, bound);
364     loc_A_to_global<2>(corr_a, bound);
365 }
366
367 bool FEM::readFile(const std::filesystem::path& path) {
368     using namespace ::Log;
369     bool isError { true };
370     std::ifstream fin(path / "params.txt");
371     isError &= is_open(fin, getLog("Error - params.txt"));
372     fin >> _size.nodes
373         >> _size.elems
374         >> _size.areas
375         >> _size.conds;
376     fin.close();
377     resize();
378     fin.open(path / "nodes.txt");
379     isError &= is_open(fin, getLog("Error - nodes.txt"));
380     for (size_t i = 0; i < _size.nodes; i++)
381         fin >> nodes[i].x >> nodes[i].y;
382     fin.close();
383     fin.open(path / "elems.txt");
384     isError &= is_open(fin, getLog("Error - elems.txt"));
385     for (size_t i = 0; i < _size.elems; i++) {
386         fin >> elems[i].nodeIdx[0]
387             >> elems[i].nodeIdx[1]
388             >> elems[i].nodeIdx[2];
389     }
390     fin.close();
391     fin.open(path / "areas.txt");

```

```

392     isError &= is_open(fin, getLog("Error - areas.txt"));
393     for (size_t i = 0; i < _size.areas; i++)
394         fin >> materials[i].gamma
395         >> materials[i].betta;
396
397     for (size_t i = 0; i < _size.elems; i++)
398         fin >> elems[i].area;
399     fin.close();
400     fin.open(path / "bords.txt");
401     isError &= is_open(fin, getLog("Error - bords.txt"));
402     for (size_t i = 0; i < _size.conds; i++)
403         fin >> boundarys[i].area
404         >> boundarys[i].nodeIdx[0]
405         >> boundarys[i].nodeIdx[1]
406         >> boundarys[i].cond
407         >> boundarys[i].type;
408     fin.close();
409     return isError;
410 }
411 #endif /// _FEM_HPP_

```

lightweight.hpp

```

1  double
2  determinant(const std::array<Union::XY, 3>& elem) {
3      return (
4          (elem[1].x - elem[0].x) * (elem[2].y - elem[0].y) -
5          (elem[1].y - elem[0].y) * (elem[2].x - elem[0].x)
6      );
7  }
8
9  double
10 edgeLength(const std::array<Union::XY, 2>& elem) {
11     return (
12         sqrt (
13             pow(elem[1].x - elem[0].x ,2) +
14             pow(elem[1].y - elem[0].y, 2)
15         )
16     );
17 }

```

3 Тестирование

Тест №1

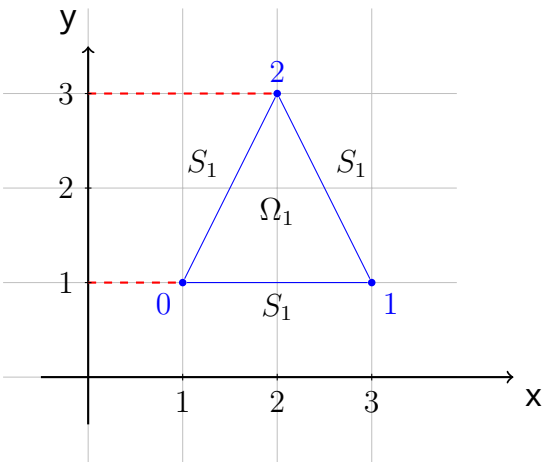
$$u(x,y) = 1$$

$$f(x,y) = 0$$

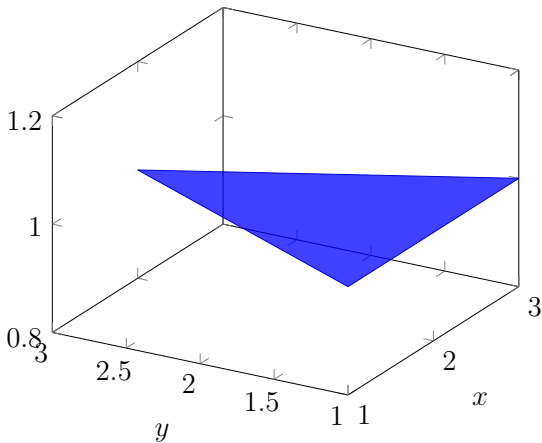
$$\lambda = 1$$

$$\gamma = 0$$

$$\beta = 0$$



$$S_1 = 1$$

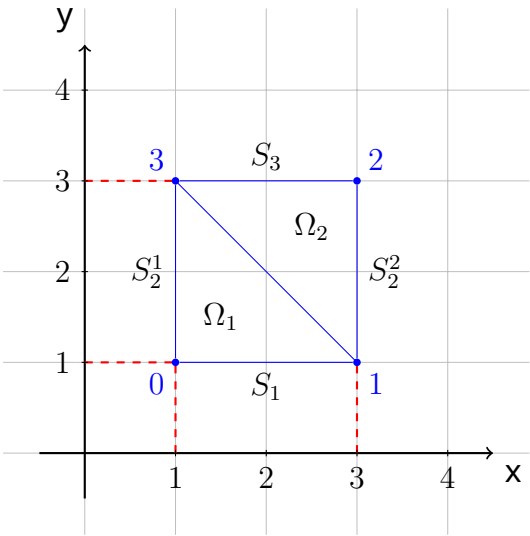


nodes	elems	area	bords
1 1	0 1 2	0	0 0 1 1 0
3 1			0 1 2 1 0
2 3			0 2 0 1 0

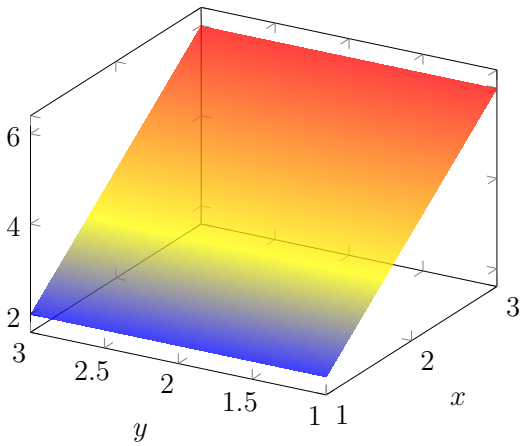
x	x^*	$x^* - x$	$\ x^* - x\ $
1,000	1,000	0,0e+00	
1,000	1,000	0,0e+00	0,0e+00
1,000	1,000	0,0e+00	

Тест №2

$u(x,y) = 2x$
 $f(x,y) = 4x - 2$
 $\lambda = x$
 $\gamma = 2$
 $\beta = 2$



$S_1 = 2x$
 $S_2^1 = -2x$
 $S_2^2 = 2x$
 $S_3 = 2x$



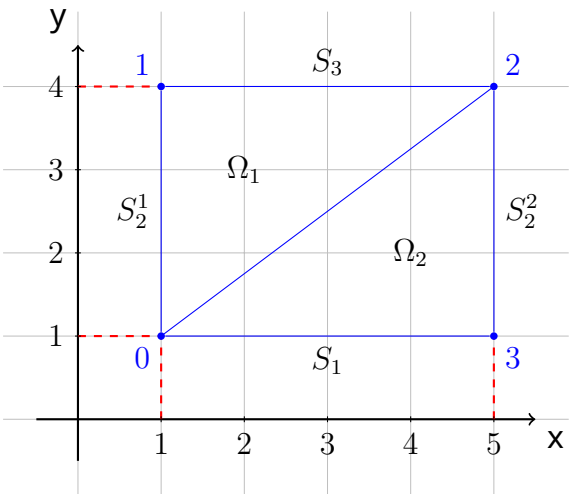
nodes	elems	area	bords
1 1	0 1 3	0	0 0 1 1 0
3 1	1 2 3	0	0 0 3 2 0
3 3			0 1 2 2 1
1 3			0 2 3 3 0

x	x^*	$x^* - x$	$\ x^* - x\ $
2,000	2,000	0,00e+00	
6,000	6,000	0,00e+00	
6,000	6,000	1,23e-13	1,20e-13
2,000	2,000	-1,29e-14	

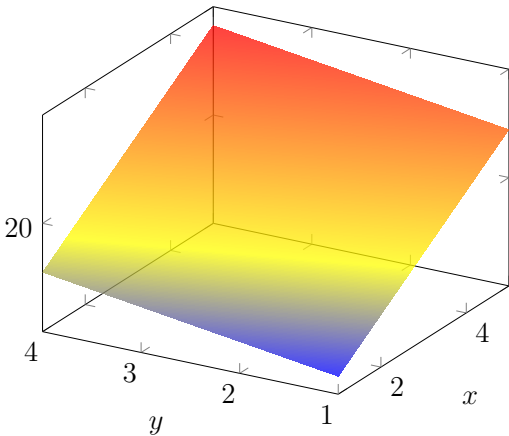
Тест №3

Количество элементов - 2

$u(x,y) = 5x + 2y$
 $f(x,y) = 10x + 4y$
 $\lambda = 2$
 $\gamma = 2$
 $\beta = 5$



$S_1 = 5x + 2$
 $S_2^1 = -10$
 $S_2^2 = 10$
 $S_3 = 5x + 8,8$



nodes	elems	area	bords
1 1	0 1 2	0	0 0 3 1 0
1 4	0 2 3	0	0 0 1 2 0
5 4			0 3 2 2 1
5 1			0 1 2 3 0

x	x^*	$x^* - x$	$\ x^* - x\ $
7,000	7,000	0,0e+00	
13,000	13,000	1,8e-13	
33,000	32,000	-3,2e-13	3,6e-13
27,000	27,000	0,0e+00	

Количество элементов - 4

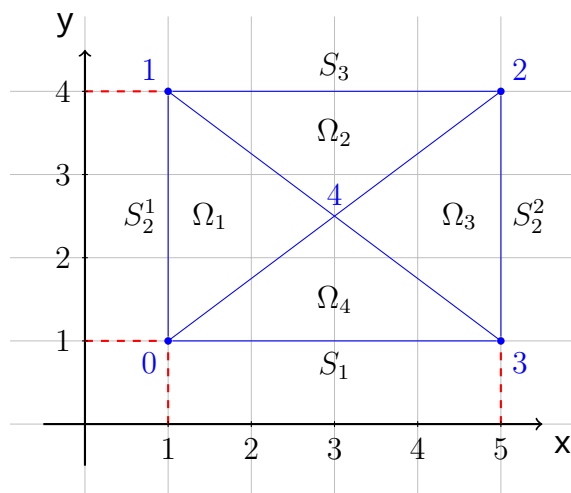
$$u(x, y) = 5x + 2y$$

$$f(x, y) = 10x + 4y$$

$$\lambda = 2$$

$$\gamma = 2$$

$$\beta = 5$$



nodes	elems	area	bords
1 1	0 1 4	0	0 0 3 1 0
1 4	1 2 4	0	0 0 1 2 0
5 4	2 3 4	0	0 3 2 2 1
5 1	0 3 4	0	0 1 2 3 0
3 2,5			

x	x^*	$x^* - x$	$\ x^* - x\ $
7,000	7,000	0,0e+00	
13,000	13,000	-2,5e-13	
33,000	33,000	-2,6e-13	3,7e-13
27,000	27,000	0,0e+00	
20,000	20,000	8,8e-14	

Количество элементов - 8

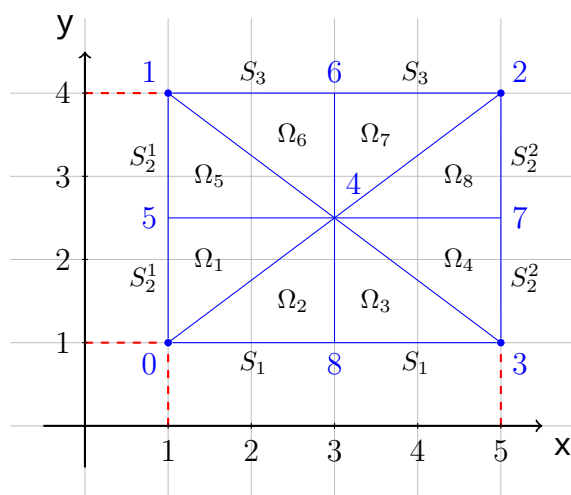
$$u(x, y) = 5x + 2y$$

$$f(x, y) = 10x + 4y$$

$$\lambda = 2$$

$$\gamma = 2$$

$$\beta = 5$$



nodes	elems	area	bords
1 1	0 4 5	0	0 0 8 1 0
1 4	0 4 8	0	0 8 3 1 0
5 4	3 4 8	0	0 0 5 2 0
5 1	3 4 7	0	0 5 1 2 0
3 2,5	1 4 5	0	0 3 7 2 1
1 2,5	1 4 6	0	0 7 2 2 1
3 4	2 4 6	0	0 1 6 3 0
5 2,5	2 4 7	0	0 6 2 3 0
3 1			

x	x^*	$x^* - x$	$\ x^* - x\ $
7,000	7,000	0,0e+00	
13,000	13,000	9,1e-14	
33,000	33,000	-1,2e-12	
27,000	27,000	0,0e+00	
20,000	20,000	4,6e-13	1,7e-12
10,000	10,000	-1,1e-13	
23,000	23,000	7,8e-13	
30,000	30,000	-5,8e-13	
17,000	17,000	0,0e+00	

Количество элементов - 16

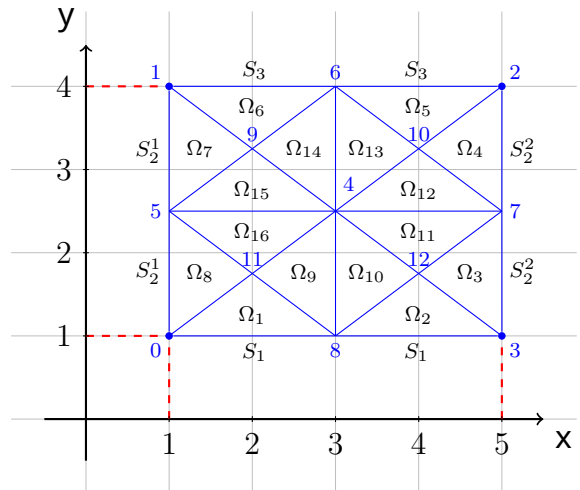
$$u(x, y) = 5x + 2y$$

$$f(x, y) = 10x + 4y$$

$$\lambda = 2$$

$$\gamma = 2$$

$$\beta = 5$$



nodes	elems	area	bords
1 1	0 8 11	0	0 0 8 1 0
1 4	3 8 12	0	0 8 3 1 0
5 4	3 7 12	0	0 0 5 2 0
5 1	2 7 10	0	0 5 1 2 0
3 2,5	2 6 10	0	0 3 7 2 1
1 2,5	1 6 9	0	0 7 2 2 1
3 4	1 5 9	0	0 1 6 3 0
5 2,5	0 5 11	0	0 6 2 3 0
3 1	4 8 11	0	
2 3,25	4 8 12	0	
4 3,25	4 7 12	0	
2 1,75	4 7 10	0	
4 1,75	4 6 10	0	
	4 6 9	0	
	4 5 9	0	
	4 5 11	0	

x	x^*	$x^* - x$	$\ x^* - x\ $
7,000	7,000	0,0e+00	
13,000	13,000	-2,8e-13	
33,000	33,000	-1,2e-12	
27,000	27,000	0,0e+00	
20,000	20,000	5,1e-13	
10,000	10,000	-2,7e-13	
23,000	23,000	8,9e-13	1,8e-12
30,000	30,000	-6,4e-13	
17,000	17,000	0,0e+00	
16,500	16,500	-5,0e-14	
26,500	26,500	-4,1e-13	
13,500	13,500	-2,0e-14	
23,500	23,500	2,2e-13	

Тест №4

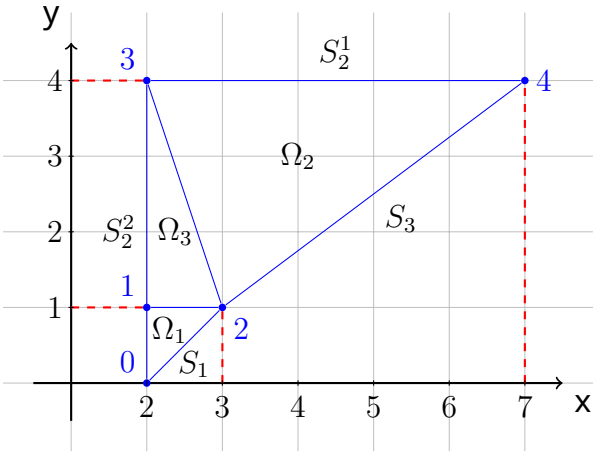
$$u(x,y)=\begin{cases} y^2 \\ 20y-19 \end{cases}$$

$$f(x,y)=\begin{cases} -20 \\ 0 \end{cases}$$

$$\lambda=\begin{cases} 10 \\ 1 \end{cases}$$

$$\gamma=0$$

$$\beta=2$$

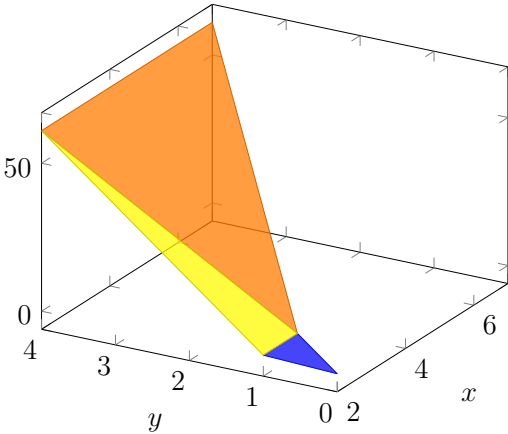


$$S_1=y^2x$$

$$S_2^1=20$$

$$S_2^2=0$$

$$S_3=20y-27$$



nodes	elems	area	bords
2 0	0 1 2	0	0 0 1 2 1
2 1	1 2 3	1	1 1 3 2 1
3 1	2 3 4	1	1 3 4 2 0
2 4			1 2 4 3 0
7 4			0 0 2 1 0

x	x^*	$x^* - x$	$\ x^* - x\ $
0,000	0,000	0,00e+00	
1,000	1,134	1,34e-01	
1,000	1,000	-1,11e-16	6,63e-01
61,000	60,350	-6,49e-01	
61,000	60,970	-2,95e-02	

Тест №5

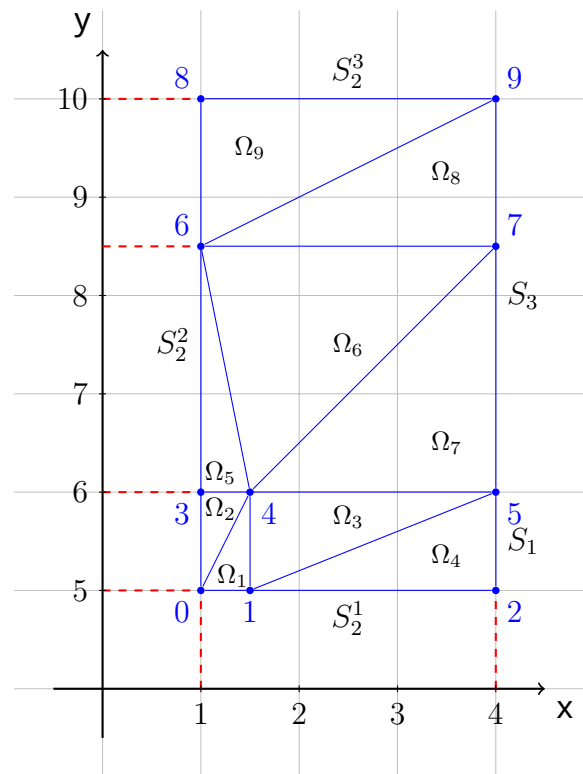
$$u(x, y) = x + 6y - 2$$

$$f(x, y) = \begin{cases} 5x + 30y - 10 \\ 0 \end{cases}$$

$$\lambda = 1$$

$$\gamma = \begin{cases} 5 \\ 0 \end{cases}$$

$$\beta = 10$$



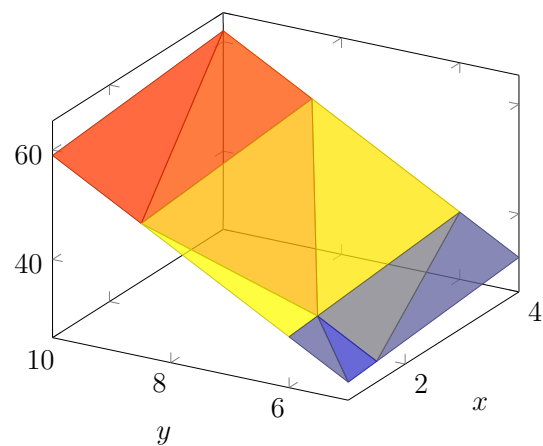
$$S_1 = 6y + 2$$

$$S_2^1 = -6$$

$$S_2^2 = -1$$

$$S_2^3 = 6$$

$$S_3 = 6y + 2.1$$



nodes	elems	area	bords
1 5	0 1 4	0	0 0 1 2 0
1.5 5	0 3 4	0	0 1 2 2 0
4 5	1 4 5	0	0 2 5 1 0
1 6	1 2 5	0	0 0 3 2 1
1.5 6	3 4 6	1	1 5 7 3 0
4 6	4 6 7	1	1 7 9 3 0
1 8.5	4 5 7	1	1 9 8 2 2
4 8.5	6 7 9	1	1 8 6 2 1
1 10	6 8 9	1	1 3 6 2 1
4 10			

x	x^*	$x^* - x$	$\ x^* - x\ $
29,000	29,000	-6,43e-13	
29,500	29,500	-4,48e-13	
32,000	32,000	0,00e+00	
35,000	35,000	-5,12e-13	
35,500	35,500	-6,47e-13	1,60e-12
38,000	38,000	-7,11e-15	
50,000	50,000	4,41e-13	
53,000	53,000	-9,38e-13	
59,000	59,000	3,98e-13	
62,000	62,000	2,34e-13	