



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

ПРЕПОДАВАТЕЛЬ

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 разложить по линейным базисным функциям. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или 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 $L_2(\Omega)$:

$$(Lu, v) = (f, v)$$

$$(Lu - f, v) = 0$$

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

$$\begin{aligned} M_{ij} &= \int_{\Omega_m} \gamma Y_i Y_j d\Omega_m = \left| \gamma = Y_1 \gamma_1 + Y_2 \gamma_2 + Y_3 \gamma_3 \right| = \int_{\Omega_m} (Y_1 \gamma_1 + Y_2 \gamma_2 + Y_3 \gamma_3) Y_i Y_j d\Omega_m = \\ &= \gamma_1 \int_{\Omega_m} Y_1 Y_i Y_j d\Omega_m + \gamma_2 \int_{\Omega_m} Y_2 Y_i Y_j d\Omega_m + \gamma_3 \int_{\Omega_m} Y_3 Y_i Y_j d\Omega_m = \\ &= \gamma_1 \int_{\Omega_m} L_1 L_i L_j d\Omega_m + \gamma_2 \int_{\Omega_m} L_2 L_i L_j d\Omega_m + \gamma_3 \int_{\Omega_m} L_3 L_i L_j d\Omega_m \end{aligned}$$

Построение матрицы жёсткости

Рассмотрим первый член в выражении для k-го конечного элемента:

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

$$B_{i,j} = (\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j) \frac{|det D|}{2} \quad i, j = \overline{0, 2}$$

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:

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

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 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:

$$I_1 = \int_{S_2} (\theta_0 \phi_0 + \theta_1 \phi_1) \phi_i dx dy$$

$$I_2 = \beta \int_{S_3} (u_{\beta 1} \phi_0 + u_{\beta 2} \phi_1) \phi_i dx dy$$

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

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

Весь проект: <https://github.com/ISTECTION/FEM>

main.cpp

```
1  #include "argparse/argparse.hpp"
2  #include "timer/cxxtimer.hpp"
3  #include "LOS/LOS.hpp"
4  #include "FEM.hpp"
5
6  #include <iostream>
7  #include <optional>
8  #include <fstream>
9
10 int main(int argc, char* argv[]) {
11     using namespace      ::Log;
12     using ::std::chrono::milliseconds;
13
14     argparse::ArgumentParser _program("FEM", "1.0.0");
15     _program.add_argument("-i", "--input")
16         .help("path to input files" )
17         .required();
18     _program.add_argument("-o", "--output")
19         .help("path to output files");
20
21     try {
22         _program.parse_args(argc, argv);
23
24         std::optional _opt          = _program.present("-o");
25         std::filesystem::path _input = _program.get<std::string>("-i");
26         std::filesystem::path _output =
27             _opt.has_value() ?
28                 _program.get<std::string>("-o") :
29                 _input / "sparse";
30
31         Function::setFunction(_input.string());
32
33         cxxtimer::Timer _timer(true);          /// start timer
34         FEM _FEM(_input);                      /// start FEM
35         _FEM.writeFile(_output, 1E-14, 10000);  /// overwriting files
36         LOS<double> _LOS(_output);             /// here is reading
37         _LOS.solve(Cond::DIAGONAL, true);
38         _timer.stop();                          /// stop timer
39
40         #if DEBUG != 0
41             _FEM.printAll();                    /// print input FEM data
42             _FEM.printSparse();                 /// print sparse format
43             _FEM.printAnalitics();              /// print analicals solve
44         #endif
45     }
```



```

45         std::cout << "SOLUTION: ";
46         print(_LOS.getX(), 14);           /// print solution vector
47         std::cout << "Milliseconds: " << _timer.count<milliseconds>();
48     } catch(const std::runtime_error& err) {
49         Logger::append(getLog("argc != 2 (FEM --input ./input)"));
50         std::cerr << err.what();
51         std::cerr << _program;
52         std::exit(1);                     /// program error
53     }
54     return 0;
55 }
56

```

Union.hpp

```

1  #ifndef _UNION_HPP_
2  #define _UNION_HPP_
3  #include <vector>
4  #include <array>
5
6  _UNION_BEGIN
7
8  struct XY {
9      double x;
10     double y;
11 };
12
13 struct Material {
14     double betta;
15     double gamma;
16 };
17
18 struct Element {
19     size_t area;
20     std::array<size_t, 3> nodeIdx;
21 };
22
23 struct Boundary {
24     size_t cond;
25     size_t type;
26     size_t area;
27     std::array<size_t, 2> nodeIdx;
28 };
29
30 struct Param {
31     size_t nodes;
32     size_t elems;
33     size_t areas;

```

```

34     size_t conds;
35 };
36
37 _UNION_END
38 #endif /// _UNION_HPP_

```

FEM.hpp

```

1  #ifndef _FEM_HPP_
2  #define _FEM_HPP_
3  #include "utils/lightweight.hpp"
4  #include "utils/overload.hpp"
5  #include "utils/friendly.hpp"
6  #include "Function.hpp"
7  #include "Logger.hpp"
8  #include "Union.hpp"
9
10 #include <algorithm>
11 #include <cmath>
12 #include <set>
13
14 class FEM
15 {
16 private:
17     Union::Param _size;
18
19     std::vector<Union::XY>      nodes;
20     std::vector<Union::Element> elems;
21     std::vector<Union::Boundary> boundarys;
22     std::vector<Union::Material> materials;
23
24     std::vector<double> gb;
25     std::vector<double> gg;
26     std::vector<double> di;
27     std::vector<size_t> ig;
28     std::vector<size_t> jg;
29
30 public:
31     FEM(std::filesystem::path _path) {
32         assert(readFile(_path));
33         portrait(true);
34         global();
35         boundaryCondition();
36     }
37     ~FEM() { }
38
39     void printAnalitics() {
40         std::vector<size_t> ax;

```

```

41
42     for (const auto& _elem : elems) {
43         for (size_t i = 0; i < 3; i++)
44             ax[_elem.nodeIdx[i]] =
45                 Function::analitics(
46                     nodes[_elem.nodeIdx[i]],
47                     _elem.area
48                 );
49     }
50     std::cout << "ANALITIC: "; print(ax, 14);
51 }
52 size_t    getNodes() { return _size.nodes; }
53
54 private:
55     void global();
56     void resize();
57
58     template<size_t N, typename _Struct>
59     void loc_A_to_global(
60         const std::array<std::array<double, N>, N>&,
61         const _Struct&
62     );
63
64     template<size_t N, typename _Struct>
65     void loc_b_to_global(const std::array<double, N>&, const _Struct& );
66
67     array::xxx localA(const std::array<Union::XY, 3>&, size_t) const;
68     array::x    buildF(const std::array<Union::XY, 3>&, size_t) const;
69
70     array::xxx G(const std::array<Union::XY, 3>&, size_t) const;
71     array::xxx M(const std::array<Union::XY, 3>&, size_t) const;
72
73     bool readFile(const std::filesystem::path& );
74     void portrait(const bool isWriteList = false);
75
76     void boundaryCondition();
77     void first (const Union::Boundary& bound);
78     void second(const Union::Boundary& bound);
79     void third (const Union::Boundary& bound);
80 };
81
82 void FEM::global() {
83
84     std::array<Union::XY, 3> coords;
85
86     for (size_t i = 0; i < _size.elems; i++) {
87         for (size_t j = 0; j < 3; j++) {
88             size_t point = elems[i].nodeIdx[j];
89             coords[j].x = nodes[point].x;

```

```

90         coords[j].y = nodes[point].y;
91     }
92     array::x    local_b = buildF(coords, elems[i].area);
93     array::xxx  local_A = localA(coords, elems[i].area);
94
95     loc_A_to_global<3>(local_A, elems[i]);
96     loc_b_to_global<3>(local_b, elems[i]);
97 }
98 }
99
100 void FEM::boundaryCondition() {
101     using namespace ::Log;
102
103     for (size_t _count = 0; _count < _size.conds; _count++) {
104         switch (boundaries[_count].cond)
105         {
106             case FIRST_BOUNDARY_COND:
107                 first(boundaries[_count]);
108                 break;
109             case SECOND_BOUNDARY_COND:
110                 second(boundaries[_count]);
111                 break;
112             case THIRD_BOUNDARY_COND:
113                 third(boundaries[_count]);
114                 break;
115             default:
116                 Logger::append(getLog("There is no such condition"));
117         }
118     }
119 }
120
121 void FEM::first(const Union::Boundary& bound) {
122     di[bound.nodeIdx[0]] = { 1 };
123     di[bound.nodeIdx[1]] = { 1 };
124
125     for (size_t i = 0; i < 2; i++)
126         gb[bound.nodeIdx[i]] =
127             Function::firstBound({
128                 nodes[bound.nodeIdx[i]].x,
129                 nodes[bound.nodeIdx[i]].y
130             }, bound.type);
131
132     for (size_t k = 0; k < 2; k++) {
133         size_t node = bound.nodeIdx[k];
134         for (size_t i = ig[node]; i < ig[node + 1]; i++) {
135             if(di[jg[i]] != 1)
136                 gb[jg[i]] -= gg[i] * gb[node];
137             gg[i] = 0;
138         }
139     }

```

```

139
140     for(size_t i = node + 1; i < _size.nodes; i++) {
141         size_t lbeg = ig[i];
142         size_t lend = ig[i + 1];
143         for(size_t p = lbeg; p < lend; p++) {
144             if(jg[p] == node) {
145                 if(di[i] != 1)
146                     gb[i] -= gg[p] * gb[node];
147                 gg[p] = 0;
148             }
149         }
150     }
151 }
152 }
153
154 void FEM::second(const Union::Boundary& bound) {
155
156     std::array<Union::XY, 2>
157         coord_borders = {
158         nodes[bound.nodeIdx[0]],
159         nodes[bound.nodeIdx[1]]
160     };
161
162     double _koef = edgeLength(coord_borders) / 6;
163
164     std::array<double, 2> corr_b;
165     for (size_t i = 0; i < 2; i++)
166         corr_b[i] = _koef * (
167             2 * Function::secondBound({
168                 nodes[bound.nodeIdx[i]].x,
169                 nodes[bound.nodeIdx[i]].y
170             }, bound.type) +
171             Function::secondBound({
172                 nodes[bound.nodeIdx[1 - i]].x,
173                 nodes[bound.nodeIdx[1 - i]].y
174             }, bound.type)
175         );
176
177     loc_b_to_global<2>(corr_b, bound);
178 }
179
180 void FEM::third(const Union::Boundary& bound) {
181
182     std::array<Union::XY, 2> coord_borders = {
183         nodes[bound.nodeIdx[0]],
184         nodes[bound.nodeIdx[1]]
185     };
186
187     double _koef =

```

```

188         materials[bound.area].betta *
189         edgeLength(coord_borders) / 6;
190
191     std::array<std::array<double, 2>, 2> corr_a;
192
193     std::array<double, 2> corr_b;
194     for (size_t i = 0; i < 2; i++) {
195
196         corr_b[i] = _koef * (
197             2 * Function::thirdBound({
198                 nodes[bound.nodeIdx[i]].x,
199                 nodes[bound.nodeIdx[i]].y
200             }, bound.type) +
201             Function::thirdBound({
202                 nodes[bound.nodeIdx[1 - i]].x,
203                 nodes[bound.nodeIdx[1 - i]].y
204             }, bound.type)
205         );
206
207         for (size_t j = 0; j < 2; j++) {
208             corr_a[i][j] =
209                 (i == j) ? (2 * _koef) :
210                 (_koef);
211         }
212     }
213     loc_b_to_global<2>(corr_b, bound);
214     loc_A_to_global<2>(corr_a, bound);
215 }
216
217 template<size_t N, typename _Struct>
218 void FEM::loc_A_to_global(
219     const std::array<std::array<double, N>, N>& locA,
220     const _Struct& elem) {
221
222     using ::std::vector;
223     using iterator = ::std::vector<size_t>::iterator;
224
225     for (size_t i = 0; i < N; i++) {
226         di[elem.nodeIdx[i]] += locA[i][i];
227
228         for (size_t j = 0; j < i; j++) {
229             size_t a = elem.nodeIdx[i];
230             size_t b = elem.nodeIdx[j];
231             if (a < b) std::swap(a, b);
232
233             if (ig[a + 1] > ig[a]) {
234                 iterator _beg = jg.begin() + ig[a];
235                 iterator _end = jg.begin() + ig[a + 1] - ig[0];
236

```

```

237         auto _itr = std::lower_bound(_beg, _end, b);
238         auto _idx = _itr - jg.begin();
239         gg[_idx] += locA[i][j];
240     }
241 }
242 }
243 }
244
245 template<size_t N, typename _Struct>
246 void FEM::loc_b_to_global(
247     const std::array<double, N>& loc_b,
248     const _Struct& elem) {
249
250     for (size_t i = 0; i < N; i++)
251         gb[elem.nodeIdx[i]] += loc_b[i];
252 }
253
254 array::x FEM::buildF(const std::array<Union::XY, 3>& elem, size_t area)
255 ↪ const {
256     std::array<double, 3> function {
257         Function::f(elem[0], area),
258         Function::f(elem[1], area),
259         Function::f(elem[2], area)
260     };
261
262     double det_D = fabs(determinant(elem)) / 24;
263     return {
264         det_D * (2 * function[0] + function[1] + function[2]),
265         det_D * (2 * function[1] + function[0] + function[2]),
266         det_D * (2 * function[2] + function[0] + function[1]),
267     };
268 }
269
270 array::xxx FEM::localA(const std::array<Union::XY, 3>& elem, size_t area)
271 ↪ const {
272     std::array<std::array<double, 3>, 3> G = FEM::G(elem, area);
273     std::array<std::array<double, 3>, 3> M = FEM::M(elem, area);
274     std::array<std::array<double, 3>, 3> A = G + M;
275     return A;
276 }
277
278 array::xxx FEM::G(const std::array<Union::XY, 3>& elem, size_t area) const
279 ↪ {
280     double det = fabs(determinant(elem));
281     double _koef = Function::lambda(area) / (2 * det);
282
283     std::array<std::array<double, 3>, 3> G;
284     std::array<std::array<double, 2>, 3> a {

```

```

283         elem[1].y - elem[2].y,
284         elem[2].x - elem[1].x,
285
286         elem[2].y - elem[0].y,
287         elem[0].x - elem[2].x,
288
289         elem[0].y - elem[1].y,
290         elem[1].x - elem[0].x
291     };
292
293     for (int i = 0; i < 3; i++)
294     for (int j = 0; j < 3; j++)
295         G[i][j] = _koef * (
296             a[i][0] * a[j][0] +
297             a[i][1] * a[j][1]
298         );
299
300     return G;
301 }
302
303 array::xxx FEM::M(const std::array<Union::XY, 3>& elem, size_t area) const
304 ↪ {
305     double det = fabs(determinant(elem));
306     double gammaKcoef = materials[area].gamma * det / 24;
307     std::array<std::array<double, 3>, 3> M;
308     for (size_t i = 0; i < 3; i++)
309     for (size_t j = 0; j < 3; j++) {
310         M[i][j] =
311             (i == j) ? (2 * gammaKcoef) :
312             (gammaKcoef);
313     }
314     return M;
315 }
316
317 void FEM::portrait(const bool isWriteList) {
318
319     const size_t N { _size.nodes };
320     std::vector<std::set<size_t>> list(N);
321
322     for (size_t el = 0; el < _size.elems; el++)
323     for (size_t point = 0; point < 3; point++) {
324         for (size_t i = point + 1; i < 3; i++) {
325             size_t idx1 = { elems[el].nodeIdx[point] };
326             size_t idx2 = { elems[el].nodeIdx[ i ] };
327             idx1 > idx2 ?
328                 list[idx1].insert(idx2) :
329                 list[idx2].insert(idx1) ;
330         }
331     }

```



```

331
332     for (size_t i = 2; i < ig.size(); i++)
333         ig[i] = ig[i - 1] + list[i - 1].size();
334
335     jg.resize(ig[N] - ig[0]);
336     gg.resize(ig[N] - ig[0]);
337
338     for (size_t index = 0, i = 1; i < list.size(); i++)
339     for (size_t value : list[i])
340         jg[index++] = value;
341
342     #if DEBUG != 0
343     if (isWriteList) {
344         std::cout << "list: " << '\n';
345         for (size_t i = 0; i < list.size(); i++) {
346             std::cout << i << ':' << ' ';
347             for (size_t j : list[i])
348                 std::cout << j << ' ';
349             std::cout << std::endl;
350         }
351     }
352     #endif
353 }
354
355 bool FEM::readFile(const std::filesystem::path& path) {
356     using namespace ::Log;
357     bool isError { true };
358
359     std::ifstream fin(path / "params.txt");
360     isError &= is_open(fin, getLog("Error - params.txt"));
361     fin >> _size.nodes
362         >> _size.elems
363         >> _size.areas
364         >> _size.conds;
365     fin.close();
366
367     resize();
368     std::fill_n(ig.begin(), 2, 0);
369
370     fin.open(path / "nodes.txt");
371     isError &= is_open(fin, getLog("Error - nodes.txt"));
372     for (size_t i = 0; i < _size.nodes; i++)
373         fin >> nodes[i].x >> nodes[i].y;
374     fin.close();
375
376     fin.open(path / "elems.txt");
377     isError &= is_open(fin, getLog("Error - elems.txt"));
378     for (size_t i = 0; i < _size.elems; i++) {
379         fin >> elems[i].nodeIdx[0]

```

```

380         >> elems[i].nodeIdx[1]
381         >> elems[i].nodeIdx[2];
382     }
383     fin.close();
384
385     fin.open(path / "areas.txt");
386     isError &= is_open(fin, getLog("Error - areas.txt"));
387     for (size_t i = 0; i < _size.areas; i++)
388         fin >> materials[i].gamma
389         >> materials[i].betta;
390
391     for (size_t i = 0; i < _size.elems; i++)
392         fin >> elems[i].area;
393     fin.close();
394
395     fin.open(path / "bords.txt");
396     isError &= is_open(fin, getLog("Error - bords.txt"));
397     for (size_t i = 0; i < _size.conds; i++)
398         fin >> boundarys[i].area
399         >> boundarys[i].nodeIdx[0]
400         >> boundarys[i].nodeIdx[1]
401         >> boundarys[i].cond
402         >> boundarys[i].type;
403     fin.close();
404
405     std::sort(
406         boundarys.begin(),
407         boundarys.end(),
408         [](Union::Boundary& _left, Union::Boundary& _right){
409             return _left.cond > _right.cond;
410         }
411     );
412
413     return isError;
414 }
415
416 void FEM::resize() {
417     nodes.resize( _size.nodes );
418     elems.resize( _size.elems );
419     boundarys.resize( _size.conds );
420     materials.resize( _size.areas );
421
422     gb.resize( _size.nodes );
423     di.resize( _size.nodes );
424     ig.resize(_size.nodes + 1);
425 }
426 #endif /// _FEM_HPP_

```

Data.hpp

```
1  #ifndef _DATA_HPP_
2  #define _DATA_HPP_
3  #include "../utils/friendly.hpp"
4  #include "../Logger.hpp"
5
6  #include <filesystem>
7  #include <cassert>
8  #include <fstream>
9  #include <sstream>
10 #include <vector>
11 #include <cmath>
12
13 _SYMMETRIC_BEG
14
15 struct Param {
16     size_t n;
17     double epsilon;
18     size_t max_iter;
19 };
20
21 enum class Cond {
22     NONE,
23     DIAGONAL,
24     HOLLESKY
25 };
26
27 template <class T>
28 class Data
29 {
30 protected:
31     Param param;
32     std::vector<size_t> ig;
33     std::vector<size_t> jg;
34     std::vector<T> di;
35     std::vector<T> gg;
36     std::vector<T> b;
37     std::vector<T> x;
38
39     std::vector<T> di_l;
40     std::vector<T> gg_l;
41     std::vector<T> y;
42     size_t iter{ 0 };
43
44 public:
45     Data(std::filesystem::path _path) { assert(loadData(_path)); }
46     Data(Friendly* _friend, size_t _n, T _eps, size_t _max_iter) {
47         param.n = _n;
48         param.epsilon = _eps;
```

```

49         param.max_iter = _max_iter;
50
51         ig = _friend->ig;
52         jg = _friend->jg;
53         gg = _friend->gg;
54         di = _friend->di;
55         b = _friend->gb;
56
57         x.resize(_n);
58         delete _friend;
59     }
60     ~Data() { }
61
62     std::vector<T>& getX() const { return x; }
63     size_t getIteration() const { return iter; }
64
65     void convertToLU();
66     std::vector<T> normal (std::vector<T> b);
67     std::vector<T> reverse(std::vector<T> y);
68     std::vector<T> mult(const std::vector<T>& _vec);
69
70 private:
71     bool loadData(std::filesystem::path _path);
72 };
73
74 template <class T>
75 void Data<T>::convertToLU() {
76     di_l = di;
77     gg_l = gg;
78
79     for (size_t i = 0; i < param.n; i++) {
80         T sum_diag = 0;
81         for (size_t j = ig[i]; j < ig[i + 1] ; j++) {
82             T sum = 0;
83             size_t jk = ig[jg[j]];
84             size_t ik = ig[i];
85             while ((ik < j) && (jk < ig[jg[j] + 1]))
86             {
87                 size_t l = jg[jk] - jg[ik];
88                 if (l == 0) {
89                     sum += gg_l[jk] * gg_l[ik];
90                     ik++; jk++;
91                 }
92                 jk += (l < 0);
93                 ik += (l > 0);
94             }
95             gg_l[j] -= sum;
96             gg_l[j] /= di_l[jg[j]];
97             sum_diag += gg_l[j] * gg_l[j];

```

```

98     }
99     di_l[i] -= sum_diag;
100    di_l[i] = sqrt(fabs(di_l[i]));
101    }
102 }
103
104 template <class T>
105 std::vector<T> Data<T>::normal(std::vector<T> b) {
106     for (size_t i = 0; i < param.n; i++) {
107         for (size_t j = ig[i]; j < ig[i + 1]; j++)
108             b[i] -= gg_l[j] * b[jg[j]];
109
110         b[i] = b[i] / di_l[i];
111     }
112     return b;
113 }
114
115 template <class T>
116 std::vector<T> Data<T>::reverse(std::vector<T> x) {
117     for (int j = param.n - 1; j >= 0; j--) {
118         x[j] = x[j] / di_l[j];
119
120         for (size_t i = ig[j]; i < ig[j + 1]; i++)
121             x[jg[i]] -= gg_l[i] * x[j];
122     }
123     return x;
124 }
125
126 template <class T>
127 std::vector<T> Data<T>::mult(const std::vector<T>& _vec) {
128     std::vector<T> pr(_vec.size());
129
130     int jj = 0;
131     for (size_t i = 0; i < _vec.size(); i++) {
132         pr[i] = di[i] * _vec[i];
133
134         for (size_t j = ig[i]; j < ig[i + 1]; j++, jj++) {
135             pr[i] += gg[jj] * _vec[jg[jj]];
136             pr[jg[jj]] += gg[jj] * _vec[i];
137         }
138     }
139     return pr;
140 }
141
142 template <typename T>
143 bool read(std::filesystem::path _path, std::vector<T>& _vec) {
144     using namespace ::Log;
145     std::ifstream fin(_path);
146     if (not

```

```

147         is_open(fin, "Error - " + _path.filename().string()))
148         return false;
149     for (size_t i = 0; i < _vec.size(); i++)
150         fin >> _vec[i];
151     fin.close(); return true;
152 }
153
154 template <class T>
155 bool Data<T>::loadData(std::filesystem::path _path) {
156     using namespace ::Log;
157     std::ifstream fin(_path / "kuslau.txt");
158     if (not is_open(fin, "Error - kuslau.txt"))
159         return false;
160     fin >> param.n
161         >> param.epsilon
162         >> param.max_iter;
163     fin.close();
164
165     bool is_cor { true };
166     ig.resize(param.n + 1);
167
168     is_cor &= read(_path / "ig.txt", ig);
169
170     gg.resize(ig.back());
171     jg.resize(ig.back());
172     di.resize( param.n );
173
174     b.resize (param.n);
175     x.resize (param.n);
176
177     is_cor &= read(_path / "gg.txt", gg);
178     is_cor &= read(_path / "di.txt", di);
179     is_cor &= read(_path / "jg.txt", jg);
180     is_cor &= read(_path / "gb.txt", b);
181     return is_cor;
182 }
183 _SYMMETRIC_END
184 #endif /// _DATA_HPP_

```

LOS.hpp

```

1  #ifndef _LOS_HPP_
2  #define _LOS_HPP_
3  #include "Data.hpp"
4  #include "LOS_Function.hpp"
5
6  using namespace Symmetric;
7

```

```

8  #define LOGGER if (isLog) \
9      printLog(this->iter, eps);
10
11  template <class T>
12  class LOS : public Data<T>
13  {
14  public:
15      LOS(std::filesystem::path _path) : Data<T>(_path) { }
16      LOS(Friendly* _friend, size_t _n, T _eps, size_t _max_iter)
17          : Data<T>(_friend, _n, _eps, _max_iter) { }
18
19      ~LOS() { }
20
21      void solve(Cond _cond, bool isLog = true);
22  private:
23      void none      (bool);
24      void diagonal(bool);
25      void hollesky (bool);
26  };
27
28  template <class T>
29  void LOS<T>::solve(Cond _cond, bool isLog) {
30      using namespace ::Log;
31      std::streamsize p = std::cout.precision();
32      std::cout.precision(2);
33      std::cout.setf(std::ios::uppercase);
34      switch (_cond) {
35          case Cond::NONE:      none( isLog ); break;
36          case Cond::DIAGONAL: diagonal(isLog); break;
37          case Cond::HOLLESKY: hollesky(isLog); break;
38          default:
39              Logger::append(getLog("this conditional non exist"));
40              std::exit(1);
41      }
42      std::cout.unsetf(std::ios::scientific);
43      std::cout.unsetf(std::ios::uppercase);
44      std::cout.precision(p);
45  }
46
47  template <class T>
48  void LOS<T>::none(bool isLog) {
49      std::vector<T> r (this->param.n),
50                    z (this->param.n),
51                    p (this->param.n),
52                    Ar(this->param.n);
53
54      r = this->b - this->mult(this->x);
55      z = r;
56      p = this->mult(z);

```

```

57
58     T alpha, betta, eps;
59     do {
60         betta    = scalar(p, p);
61         alpha    = scalar(p, r) / betta;
62         this->x = this->x + alpha * z;
63         r        = r - alpha * p;
64         Ar       = this->mult(r);
65         betta    = scalar(p, Ar) / betta;
66         z        = r - betta * z;
67         p        = Ar - betta * p;
68         eps      = scalar(r, r);
69
70         this->iter++;
71         LOGGER
72
73     } while(
74         this->iter < this->param.max_iter
75         && eps > this->param.epsilon);
76 }
77
78 template <class T>
79 void LOS<T>::diagonal(bool isLog) {
80     std::vector<T> r (this->param.n),
81                   z (this->param.n),
82                   p (this->param.n),
83                   Ar(this->param.n);
84
85     std::vector<T> L(this->param.n, 1);
86     for (size_t i = 0; i < L.size(); i++)
87         L[i] /= sqrt(this->di[i]);
88
89     r = L * (this->b - this->mult(this->x));
90     z = L * r;
91     p = L * this->mult(z);
92
93     T alpha, betta, eps;
94     do {
95         betta    = scalar(p, p);
96         alpha    = scalar(p, r) / betta;
97         this->x = this->x + alpha * z;
98         r        = r - alpha * p;
99         Ar       = L * this->mult(L * r);
100        betta    = scalar(p, Ar) / betta;
101        z        = L * r - betta * z;
102        p        = Ar - betta * p;
103        eps      = scalar(r, r);
104
105        this->iter++;

```



```

106         LOGGER
107
108     } while(
109         this->iter < this->param.max_iter
110         && eps > this->param.epsilon);
111 }
112
113 template <class T>
114 void LOS<T>::hollesky(bool isLog) {
115     std::vector<T> r (this->param.n),
116                   z (this->param.n),
117                   p (this->param.n),
118                   Ar (this->param.n),
119                   LAU(this->param.n);
120
121     this->convertToLU();
122     r = this->normal(this->b - this->mult(this->x));
123     z = this->reverse(r);
124     p = this->normal(this->mult(z));
125
126     T alpha, betta, eps;
127     do {
128         betta = scalar(p, p);
129         alpha = scalar(p, r) / betta;
130         this->x = this->x + alpha * z;
131         r = r - alpha * p;
132         LAU = this->normal(this->mult(this->reverse(r)));
133         betta = scalar(p, LAU) / betta;
134         z = this->reverse(r) - betta * z;
135         p = LAU - betta * p;
136         eps = scalar(r, r);
137
138         this->iter++;
139         LOGGER
140     }
141     while(
142         this->iter < this->param.max_iter
143         && eps > this->param.epsilon);
144 }
145 #undef LOGGER
146 #endif /// _LOS_HPP_

```

LOS_Function.hpp

```

1  #ifndef _LOS_FUNCTION_HPP_
2  #define _LOS_FUNCTION_HPP_
3
4  #include <numeric>

```

```

5  #include <vector>
6  #include <cmath>
7
8  template <typename T>
9  inline void printLog(size_t _iter, T _eps) {
10     std::cout << "Iteration = " << std::fixed << _iter << "\t\t" <<
11         "Discrepancy = " << std::scientific << _eps << std::endl;
12 }
13
14 template <typename T>
15 T scalar(const std::vector<T>& _v1, const std::vector<T>& _v2) {
16     T _res = 0;
17     for (size_t i = 0; i < _v1.size(); i++)
18         _res += _v1[i] * _v2[i];
19     return _res;
20 }
21
22 template <typename T>
23 T norm (const std::vector<T>& _v) {
24     return sqrt(std::accumulate(_v.begin(), _v.end(), 0.0,
25         [](double _S, const double &_E1) { return _S + _E1 * _E1; }));
26 }
27
28 template <typename T>
29 std::vector<T> operator* (std::vector<T> _v1, const std::vector<T>& _v2) {
30     for (size_t i = 0; i < _v1.size(); i++)
31         _v1[i] *= _v2[i];
32     return _v1;
33 };
34
35 template <typename T>
36 std::vector<T> operator- (std::vector<T> _v1, const std::vector<T>& _v2) {
37     for (size_t i = 0; i < _v1.size(); i++)
38         _v1[i] -= _v2[i];
39     return _v1;
40 };
41
42 template <typename T>
43 std::vector<T> operator+ (std::vector<T> _v1, const std::vector<T>& _v2) {
44     for (size_t i = 0; i < _v1.size(); i++)
45         _v1[i] += _v2[i];
46     return _v1;
47 };
48
49 template <typename T>
50 std::vector<T> operator* (T _alpha, std::vector<T> _v1) {
51     for (size_t i = 0; i < _v1.size(); i++)
52         _v1[i] *= _alpha;
53     return _v1;

```

```

54 };
55 #endif // _LOS_FUNCTION_HPP_

```

lightweight.hpp

```

1  #ifndef _LIGHTWEIGHT_HPP_
2  #define _LIGHTWEIGHT_HPP_
3  #include "../Union.hpp"
4
5  #include <array>
6  #include <cmath>
7
8  double
9  determinant(const std::array<Union::XY, 3>& elem) {
10     return (
11         (elem[1].x - elem[0].x) * (elem[2].y - elem[0].y) -
12         (elem[1].y - elem[0].y) * (elem[2].x - elem[0].x)
13     );
14 }
15
16 double
17 edgeLength(const std::array<Union::XY, 2>& elem) {
18     return (
19         sqrt (
20             pow(elem[1].x - elem[0].x ,2) +
21             pow(elem[1].y - elem[0].y, 2)
22         )
23     );
24 }
25 #endif // _LIGHTWEIGHT_HPP_

```

overload.hpp

```

1  #ifndef _OVERLOAD_HPP_
2  #define _OVERLOAD_HPP_
3  #include <iostream>
4  #include <vector>
5  #include <array>
6
7  namespace array {
8      using x    = std::array<double, 3>;
9      using xxx = std::array<std::array<double, 3>, 3>;
10 }
11
12 array::xxx
13 operator+ (const array::xxx& G, const array::xxx& M) {

```

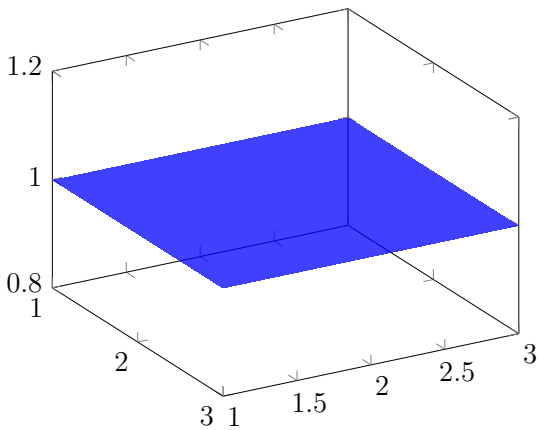
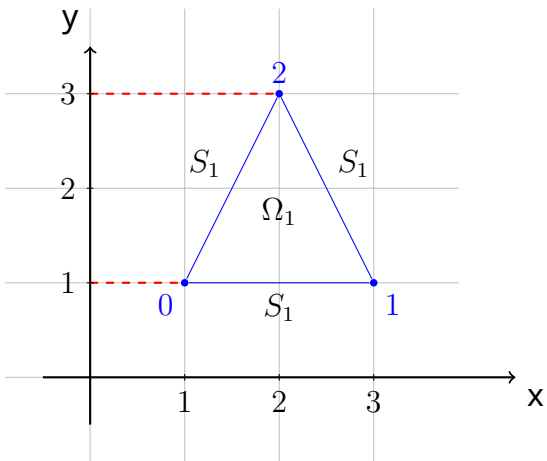
```
14     array::xxx _res;
15     for (size_t i = 0; i < G.size(); i++)
16     for (size_t j = 0; j < G.size(); j++)
17         _res[i][j] = G[i][j] + M[i][j];
18     return _res;
19 }
20 #endif // _OVERLOAD_HPP_
```

3 Тестирование

Тест №1

$u(x,y) = 1$
 $f(x,y) = 0$
 $\lambda = 1$
 $\gamma = 0$
 $\beta = 0$

$I_0 = 1$



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	0.00E+00	
1.000	1.000	0.00E+00	0.00E+00
1.000	1.000	0.00E+00	

Тест №2

Количество элементов - 2

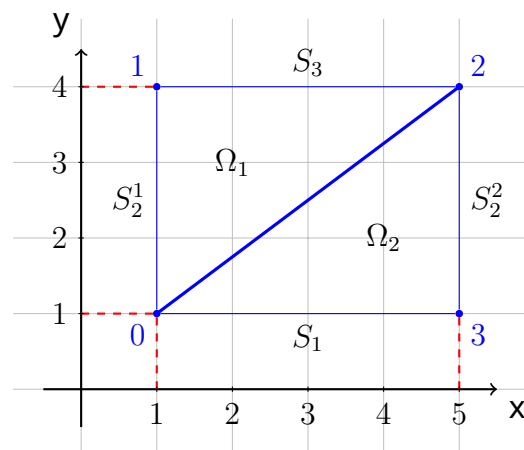
$$u(x, y) = 5x + 2y$$

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

$$\lambda = 2$$

$$\gamma = 2$$

$$\beta = 5$$

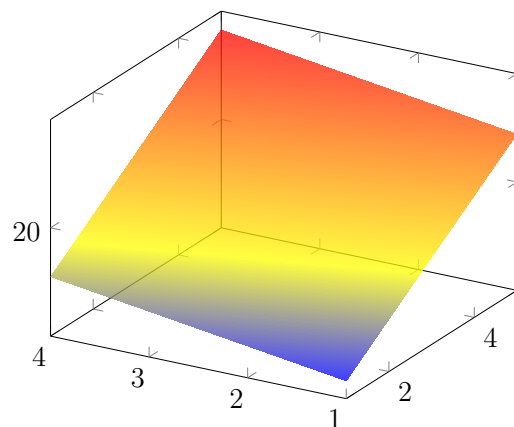


$$I_0 = 5x + 2$$

$$II_0 = -10$$

$$II_1 = 10$$

$$III_0 = 5x + 8,8$$



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	0.00E+00	
13.000	13.000	-9.61E-08	3.04E-07
33.000	32.999	2.88E-07	
27.000	27.000	0.00E+00	

Количество элементов - 4

Количество элементов - 8

Количество элементов - 16

Тест №3

4 Выводы

Выводы