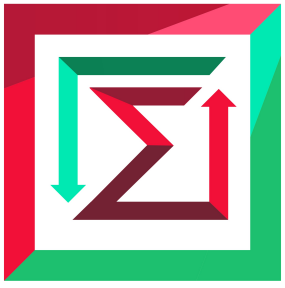
МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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 ти-пов. Коэффициент разложить по линейным базисным функциям. Матрицу СЛАУ генери-ровать 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

$$-div(\lambda gradu) + \gamma u = f$$

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

$$u|_{S_1} = u_g$$

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

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

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

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

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:

$$\begin{cases} L_1 + L_2 + L_3 = 1 \\ L_1 x_1 + L_2 x_2 + L_3 x_3 = x \\ L_1 y_1 + L_2 y_2 + L_3 y_3 = y \end{cases}$$

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 $L_1(x,y), L_2(x,y), L_3(x,y)$

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

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

$$D^{-1} = \frac{1}{|detD|} \begin{pmatrix} x_2 y_3 - x_3 y_2 & y_2 - y_3 & x_3 - x_2 \\ x_3 y_1 - x_1 y_3 & y_3 - y_1 & x_1 - x_3 \\ x_1 y_2 - x_2 y_1 & y_1 - y_2 & x_2 - x_1 \end{pmatrix}$$

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.

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

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

$$M_{ij} = \int\limits_{\Omega_m} \gamma Y_i Y_j d\Omega_m = \left| \gamma = Y_1\gamma_1 + Y_2\gamma_2 + Y_3\gamma_3 \right| = \int\limits_{\Omega_m} \left(Y_1\gamma_1 + Y_2\gamma_2 + Y_3\gamma_3\right) Y_i Y_j d\Omega_m =$$

$$= \gamma_1 \int\limits_{\Omega_m} Y_1 Y_i Y_j d\Omega_m + \gamma_2 \int\limits_{\Omega_m} Y_2 Y_i Y_j d\Omega_m + \gamma_3 \int\limits_{\Omega_m} Y_3 Y_i Y_j d\Omega_m =$$

$$= \gamma_1 \int\limits_{\Omega_m} L_1 L_i L_j d\Omega_m + \gamma_2 \int\limits_{\Omega_m} L_2 L_i L_j d\Omega_m + \gamma_3 \int\limits_{\Omega_m} L_3 L_i L_j d\Omega_m$$

Построение матрицы жёсткости

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

$$\int\limits_{\Omega_k} \lambda \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} dxdy \right)$$

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

Построение вектора правой части

Рассмотрим правую часть выражения для k-го конечного элемента:

$$\int\limits_{\Omega_k} f \psi_i dxdy$$

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

$$\int\limits_{\Omega_k} f_q L_q L_i dxdy = f_q \int\limits_{\Omega_k} L_q L_i d\Omega_k$$

Таким образом:

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

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 м атрицы 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:

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

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

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

$$\int\limits_{S_2} \theta\psi_j dxdy, \qquad \int\limits_{S_3} \beta u_\beta \psi_j dxdy, \qquad \int\limits_{S_3} \beta\psi_i\psi_j dxdy$$

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

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

где $\phi_i,\ i = \overline{0,1}$ - 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:

$$I_1 = \int\limits_{S_2} (\theta_0\phi_0 + \theta_1\phi_1)\phi_i dxdy$$

$$I_2 = \beta \int\limits_{S_3} (u_{\beta 1}\phi_0 + u_{\beta 2}\phi_1)\phi_i dxdy$$

$$I_3 = \beta \int\limits_{S_3} \phi_i\phi_j dxdy$$

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 $\phi_i,\ i = \overline{0,1}$ конечного элемента.

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

Игтегралы I_1, I_2, I_3 будем вычислять по формуле:

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

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.

$$I_1 = \begin{pmatrix} \int\limits_{S_2} L_1 L_1 dxdy & \int\limits_{S_2} L_1 L_2 dxdy \\ \int\limits_{S_2} L_2 L_1 dxdy & \int\limits_{S_2} L_2 L_2 dxdy \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = \frac{1}{6} mes S_2 \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$$

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:

$$I_3 = \beta \int\limits_{S_3} \phi_i \phi_j dxdy$$

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

main.cpp

```cpp
#include "argparse/argparse.hpp"
#include "timer/cxxtimer.hpp"
#include "LOS/LOS.hpp"
#include "FEM.hpp"

#include <iostream>
#include <optional>
#include <fstream>

int main(int argc, char* argv[]) {
    using namespace    ::Log;
    using ::std::chrono::milliseconds;

    argparse::ArgumentParser _program("FEM", "1.0.0");
    _program.add_argument("-i", "--input")
        .help("path to input files" )
        .required();

    _program.add_argument("-o", "--output")
        .help("path to output files");

    try {
```

```
        _program.parse_args(argc, argv);

        std::optional _opt            = _program.present("-o");
        std::filesystem::path _input  = _program.get<std::string>("-i");
        std::filesystem::path _output =
            _opt.has_value() ?
                _program.get<std::string>("-o") :
                _input / "sparse";

        Function::setFunction(_input.string());

        cxxtimer::Timer _timer(true);       /// start timer
        FEM _FEM(_input);                   /// start FEM
        LOS<double> _LOS (
            _FEM.takeDate(),                /// data
            _FEM.getNodes(),                /// count nodes
            1E-16, 1000);                   /// epsilon and max iteration
        _LOS.solve(Cond::HOLLESKY, true);   /// solve LOS + DIAGONAL
        _timer.stop();                      /// stop timer

        #if DEBUG != 0
        _FEM.printAll();                    /// print input FEM data
        _FEM.printSparse();                 /// print sparse format
        _LOS.printX(14);                    /// print solution vector
        _FEM.printAnalitics();              /// print analicals solve
        #endif

        std::cout << std::scientific << 0.1341234 << std::endl;
        std::cout << "Milliseconds: " << _timer.count<milliseconds>();

    } catch(const std::runtime_error& err) {
        Logger::append(getLog("argc != 2 (FEM --input ./input)"));
        std::cerr << err.what();
        std::cerr << _program;
        std::exit(1);                       /// program error
    }
    return 0;
}
```

Union.hpp

```
#ifndef _UNION_HPP_
#define _UNION_HPP_
#include <vector>
#include <array>

#define _UNION_BEGIN namespace Union {
#define _UNION_END                    }
```

```

_UNION_BEGIN

struct XY {
    double x;
    double y;
    size_t area;
};

struct Material {
    double betta;
    double gamma;
};

struct Element {
    size_t area;
    std::array<size_t, 3> nodeIdx;
};

struct Boundary {
    size_t cond;
    size_t type;
    size_t area;
    std::array<size_t, 2> nodeIdx;
};

struct Param {
    size_t nodes;
    size_t elems;
    size_t areas;
    size_t conds;
};

_UNION_END
#undef _UNION_BEGIN
#undef _UNION_END
#endif /// _UNION_HPP_
```

FEM.hpp

```
#ifndef _FEM_HPP_
#define _FEM_HPP_
#include "utils/lightweight.hpp"
#include "utils/overload.hpp"
#include "utils/friendly.hpp"
#include "Function.hpp"
#include "Logger.hpp"
#include "Union.hpp"
```

```
9
10  #include <algorithm>
11  #include <cmath>
12  #include <set>
13
14  #define FIRST_BOUNDARY_COND  1
15  #define SECOND_BOUNDARY_COND 2
16  #define THIRD_BOUNDARY_COND  3
17
18  class FEM
19  {
20  private:
21      Union::Param _size;
22
23      std::vector<Union::XY>       nodes;
24      std::vector<Union::Element>  elems;
25      std::vector<Union::Boundary> boundarys;
26      std::vector<Union::Material> materials;
27
28      std::vector<double> gb;
29      std::vector<double> gg;
30      std::vector<double> di;
31      std::vector<size_t> ig;
32      std::vector<size_t> jg;
33
34  public:
35      FEM(std::filesystem::path _path) {
36          assert(readFile(_path));
37          portrait(true);
38          global();
39          boundaryCondition();
40      }
41      ~FEM() { }
42
43      void printAll()    const;
44      void printSparse() const;
45
46      void writeFile(
47          const std::filesystem::path&,
48          const double,
49          const size_t
50      ) const;
51
52      void printAnalitics() {
53          std::vector<size_t> ax;
54          ax.resize(_size.nodes);
55          for (size_t i = 0; i < _size.nodes; i++)
56              ax[i] = Function::analitics(
57                  nodes[i]);
```

```
58          pretty(ax);
59      }
60
61      size_t    getNodes() { return _size.nodes; }
62      Friendly* takeDate() {
63          Friendly* _friend =
64              new Friendly {
65                  gb,
66                  gg,
67                  di,
68                  ig,
69                  jg
70              };
71          return _friend;
72      }
73
74  private:
75      void global();
76      void resize();
77
78      template<size_t N, typename _Struct>
79      void loc_A_to_global(
80          const std::array<std::array<double, N>, N>&,
81          const _Struct& );
82
83      template<size_t N, typename _Struct>
84      void loc_b_to_global(
85          const std::array<double, N>&,
86          const _Struct& );
87
88      array::xxx localA(const std::array<Union::XY, 3>&, size_t) const;
89      array::x   buildF(const std::array<Union::XY, 3>&, size_t) const;
90
91      array::xxx G(const std::array<Union::XY, 3>&, size_t) const;
92      array::xxx M(const std::array<Union::XY, 3>&, size_t) const;
93
94      bool readFile(const std::filesystem::path&  );
95      void portrait(const bool isWriteList = false);
96
97      void boundaryCondition();
98      void first (const Union::Boundary& bound);
99      void second(const Union::Boundary& bound);
100     void third (const Union::Boundary& bound);
101 };
102
103 void FEM::global() {
104     std::array<Union::XY, 3> coords;
105     for (size_t i = 0; i < _size.elems; i++) {
106         for (size_t j = 0; j < 3; j++) {
```

```
107             size_t point = elems[i].nodeIdx[j];
108             coords[j].x = nodes[point].x;
109             coords[j].y = nodes[point].y;
110         }
111         array::x   local_b = buildF(coords, elems[i].area);
112         array::xxx local_A = localA(coords, elems[i].area);
113 
114         #if DEBUG != 0
115         std::cout << "Element: "
116                   << elems[i].nodeIdx[0] << ' '
117                   << elems[i].nodeIdx[1] << ' '
118                   << elems[i].nodeIdx[2] << '\n';
119         pretty(local_A);
120         pretty(local_b);
121         #endif
122 
123         loc_A_to_global<3>(local_A, elems[i]);
124         loc_b_to_global<3>(local_b, elems[i]);
125 
126         #if DEBUG != 0
127         prettyG(getSparse());
128         pretty(gb);
129         #endif
130     }
131 }
132 
133 void FEM::boundaryCondition() {
134     using namespace ::Log;
135 
136     for (size_t _count = 0; _count < _size.conds; _count++) {
137         switch (boundarys[_count].cond)
138         {
139             case FIRST_BOUNDARY_COND:
140                 first(boundarys[_count]);
141                 break;
142             case SECOND_BOUNDARY_COND:
143                 second(boundarys[_count]);
144                 break;
145             case THIRD_BOUNDARY_COND:
146                 third(boundarys[_count]);
147                 break;
148             default:
149                 Logger::append(getLog("There is no such condition"));
150         }
151     }
152 }
153 
154 void FEM::first(const Union::Boundary& bound) {
155     di[bound.nodeIdx[0]] = { 1 };
```

```
    di[bound.nodeIdx[1]] = { 1 };

    for (size_t i = 0; i < 2; i++)
        gb[bound.nodeIdx[i]] =
            Function::firstBound({
                nodes[bound.nodeIdx[i]].x,
                nodes[bound.nodeIdx[i]].y
            }, bound.type);

    for (size_t k = 0; k < 2; k++) {
        size_t node = bound.nodeIdx[k];
        for (size_t i = ig[node]; i < ig[node + 1]; i++) {
            if(di[jg[i]] != 1)
                gb[jg[i]] -= gg[i] * gb[node];
            gg[i] = 0;
        }

        for(size_t i = node + 1; i < _size.nodes; i++) {
            size_t lbeg = ig[i];
            size_t lend = ig[i + 1];
            for(size_t p = lbeg; p < lend; p++) {
                if(jg[p] == node) {
                    if(di[i] != 1)
                        gb[i] -= gg[p] * gb[node];
                    gg[p] = 0;
                }
            }
        }
    }
}

void FEM::second(const Union::Boundary& bound) {

    std::array<Union::XY, 2>
        coord_borders = {
            nodes[bound.nodeIdx[0]],
            nodes[bound.nodeIdx[1]]
        };

    double _koef = edgeLength(coord_borders) / 6;

    std::array<double, 2> corr_b;
    for (size_t i = 0; i < 2; i++)
        corr_b[i] = _koef * (
            2 * Function::secondBound({
                    nodes[bound.nodeIdx[i]].x,
                    nodes[bound.nodeIdx[i]].y
                }, bound.type) +
                Function::secondBound({
```

```
                    nodes[bound.nodeIdx[1 - i]].x,
                    nodes[bound.nodeIdx[1 - i]].y
                }, bound.type)
            );

    loc_b_to_global<2>(corr_b, bound);
}

void FEM::third(const Union::Boundary& bound) {

    std::array<Union::XY, 2> coord_borders = {
        nodes[bound.nodeIdx[0]],
        nodes[bound.nodeIdx[1]]
    };

    double _koef =
        materials[bound.area].betta *
        edgeLength(coord_borders) / 6;

    std::array<std::array<double, 2>, 2> corr_a;

    std::array<double, 2> corr_b;
    for (size_t i = 0; i < 2; i++) {

        corr_b[i] = _koef * (
            2 * Function::thirdBound({
                    nodes[bound.nodeIdx[i]].x,
                    nodes[bound.nodeIdx[i]].y
                }, bound.type) +
                Function::thirdBound({
                    nodes[bound.nodeIdx[1 - i]].x,
                    nodes[bound.nodeIdx[1 - i]].y
                }, bound.type)
            );

        for (size_t j = 0; j < 2; j++) {
            corr_a[i][j] =
                (i == j) ? (2 * _koef) :
                           (    _koef) ;
        }
    }
    loc_b_to_global<2>(corr_b, bound);
    loc_A_to_global<2>(corr_a, bound);
}
template<size_t N, typename _Struct>
void FEM::loc_A_to_global(
        const std::array<std::array<double, N>, N>& locA,
        const _Struct& elem) {

```

```
    using            ::std::vector;
    using iterator = ::std::vector<size_t>::iterator;

    for (size_t i = 0; i < N; i++) {
        di[elem.nodeIdx[i]] += locA[i][i];

        for (size_t j = 0; j < i; j++) {
            size_t a = elem.nodeIdx[i];
            size_t b = elem.nodeIdx[j];
            if (a < b) std::swap(a, b);

            if (ig[a + 1] > ig[a]) {
                iterator _beg = jg.begin() + ig[a];
                iterator _end = jg.begin() + ig[a + 1] - ig[0];

                auto _itr = std::lower_bound(_beg, _end, b);
                auto _idx = _itr - jg.begin();
                gg[_idx] += locA[i][j];
            }
        }
    }
}

template<size_t N, typename _Struct>
void FEM::loc_b_to_global(
        const std::array<double, N>& loc_b,
        const _Struct& elem) {

    for (size_t i = 0; i < N; i++)
        gb[elem.nodeIdx[i]] += loc_b[i];
}

array::x FEM::buildF(const std::array<Union::XY, 3>& elem, size_t area)
 ↪  const {
    std::array<double, 3> function {
        Function::f(elem[0], area),
        Function::f(elem[1], area),
        Function::f(elem[2], area)
    };

    double det_D = fabs(determinant(elem)) / 24;
    return {
        det_D * (2 * function[0] + function[1] + function[2]),
        det_D * (2 * function[1] + function[0] + function[2]),
        det_D * (2 * function[2] + function[0] + function[1]),
    };
}

array::xxx FEM::localA(const std::array<Union::XY, 3>& elem, size_t area)
 ↪  const {
```

```
302     std::array<std::array<double, 3>, 3> G = FEM::G(elem, area);
303     std::array<std::array<double, 3>, 3> M = FEM::M(elem, area);
304     std::array<std::array<double, 3>, 3> A = G + M;
305     return A;
306 }
307 
308 array::xxx FEM::G(const std::array<Union::XY, 3>& elem, size_t area) const
    ↪ {
309     double det   = fabs(determinant(elem));
310     double _koef = Function::lambda(area) / (2 * det);
311 
312     std::array<std::array<double, 3>, 3> G;
313     std::array<std::array<double, 2>, 3> a {
314 
315             elem[1].y - elem[2].y,
316             elem[2].x - elem[1].x,
317 
318             elem[2].y - elem[0].y,
319             elem[0].x - elem[2].x,
320 
321             elem[0].y - elem[1].y,
322             elem[1].x - elem[0].x
323     };
324 
325     for (int i = 0; i < 3; i++)
326     for (int j = 0; j < 3; j++)
327         G[i][j] = _koef * (
328             a[i][0] * a[j][0] +
329             a[i][1] * a[j][1]
330         );
331 
332     return G;
333 }
334 
335 array::xxx FEM::M(const std::array<Union::XY, 3>& elem, size_t area) const
    ↪ {
336     double det = fabs(determinant(elem));
337     double gammaKoef = materials[area].gamma * det / 24;
338     std::array<std::array<double, 3>, 3> M;
339     for (size_t i = 0; i < 3; i++)
340     for (size_t j = 0; j < 3; j++) {
341         M[i][j] =
342             (i == j) ? (2 * gammaKoef) :
343                        (    gammaKoef) ;
344     }
345     return  M;
346 }
347 
348 void FEM::portrait(const bool isWriteList) {
```

```
349
350     const size_t N {    _size.nodes     };
351     std::vector<std::set<size_t>> list(N);
352
353     for (size_t el = 0; el < _size.elems; el++)
354     for (size_t point = 0; point < 3; point++) {
355         for (size_t i = point + 1; i < 3; i++) {
356             size_t idx1 = { elems[el].nodeIdx[point] };
357             size_t idx2 = { elems[el].nodeIdx[  i  ] };
358             idx1 > idx2 ?
359                 list[idx1].insert(idx2) :
360                 list[idx2].insert(idx1) ;
361         }
362     }
363
364     for (size_t i = 2; i < ig.size(); i++)
365         ig[i] = ig[i - 1] + list[i - 1].size();
366
367     jg.resize(ig[N] - ig[0]);
368     gg.resize(ig[N] - ig[0]);
369
370     for (size_t index = 0, i = 1; i < list.size(); i++)
371     for (size_t value : list[i])
372         jg[index++] = value;
373
374     #if DEBUG != 0
375     if (isWriteList) {
376         std::cout << "list: " << '\n';
377         for (size_t i = 0; i < list.size(); i++) {
378             std::cout << i << ':' << ' ';
379             for (size_t j : list[i])
380                 std::cout << j << ' ';
381             std::cout << std::endl;
382         }
383     }
384     #endif
385 }
386
387 void FEM::printAll() const {
388     #define PRINTLINE \
389         for (size_t i = 0; i < 20; std::cout << '-', i++);
390     #define ENDLINE std::cout << '\n';
391     SetConsoleOutputCP(65001);
392     PRINTLINE ENDLINE
393     std::cout << "PARAMS:         "                << '\n';
394     std::cout << "Size nodes:     " << _size.nodes  << '\n';
395     std::cout << "Size element:   " << _size.elems  << '\n';
396     std::cout << "Size areas:     " << _size.areas  << '\n';
397     std::cout << "Size condition: " << _size.conds  << '\n';
```

```
398     PRINTLINE ENDLINE
399     std::cout << std::setw(4) << "X" << std::setw(4) << "Y" << '\n';
400     for (size_t i = 0; i < _size.nodes; i++)
401         std::cout << std::setw(4) << nodes[i].x
402                   << std::setw(4) << nodes[i].y << '\n';
403     PRINTLINE ENDLINE
404     std::cout << "Elements: " << '\n';
405     for (size_t i = 0; i < _size.elems; i++)
406         std::cout << elems[i].nodeIdx[0] << ' '
407                   << elems[i].nodeIdx[1] << ' '
408                   << elems[i].nodeIdx[2] << " -> area "
409                   << elems[i].area       << '\n';
410     PRINTLINE ENDLINE
411     std::cout << "Areas: " << '\n';
412     for (size_t i = 0; i < _size.areas; i++) {
413         std::cout << "\u03B3 = " << materials[i].gamma  << ',' << ' '
414                   << "\u03B2 = " << materials[i].betta
415                   << " -> area " << i << '\n';
416     }
417     PRINTLINE ENDLINE
418     std::cout << "Borders: " << '\n';
419     for (size_t i = 0; i < _size.conds; i++)
420         std::cout << boundarys[i].area       << ' '
421                   << boundarys[i].nodeIdx[0] << ' '
422                   << boundarys[i].nodeIdx[1] << ' '
423                   << boundarys[i].cond       << ' '
424                   << boundarys[i].type       << ' ' << '\n';
425     PRINTLINE ENDLINE
426     #undef PRINTLINE
427     #undef ENDLINE
428 }
429
430 void FEM::printSparse() const {
431     #define PRINTLINE \
432         for (size_t i = 0; i < 20; std::cout << '-', i++); \
433         std::cout << '\n';
434     PRINTLINE
435     std::cout << "ig: "; print(ig);
436     std::cout << "jg: "; print(jg);
437     std::cout << "di: "; print(di);
438     std::cout << "gg: "; print(gg);
439     PRINTLINE
440     #undef PRINTLINE
441 }
442
443 bool FEM::readFile(const std::filesystem::path& path) {
444     using namespace ::Log;
445     bool isError { true };
446
```

```
    std::ifstream fin(path / "params.txt");
    isError &= is_open(fin, getLog("Error - params.txt"));
    fin >> _size.nodes
        >> _size.elems
        >> _size.areas
        >> _size.conds;
    fin.close();

    resize();
    std::fill_n(ig.begin(), 2, 0);

    fin.open(path / "nodes.txt");
    isError &= is_open(fin, getLog("Error - nodes.txt"));
    for (size_t i = 0; i < _size.nodes; i++)
        fin >> nodes[i].x >> nodes[i].y;
    fin.close();

    fin.open(path / "nodes_area.txt");
    isError &= is_open(fin, getLog("Error - nodes_area.txt"));
    for (size_t i = 0; i < _size.nodes; i++)
        fin >> nodes[i].area;
    fin.close();

    fin.open(path / "elems.txt");
    isError &= is_open(fin, getLog("Error - elems.txt"));
    for (size_t i = 0; i < _size.elems; i++) {
        fin >> elems[i].nodeIdx[0]
            >> elems[i].nodeIdx[1]
            >> elems[i].nodeIdx[2];
    }
    fin.close();

    fin.open(path / "areas.txt");
    isError &= is_open(fin, getLog("Error - areas.txt"));
    for (size_t i = 0; i < _size.areas; i++)
        fin >> materials[i].gamma
            >> materials[i].betta;

    for (size_t i = 0; i < _size.elems; i++)
        fin >> elems[i].area;
    fin.close();

    fin.open(path / "bords.txt");
    isError &= is_open(fin, getLog("Error - bords.txt"));
    for (size_t i = 0; i < _size.conds; i++)
        fin >> boundarys[i].area
            >> boundarys[i].nodeIdx[0]
            >> boundarys[i].nodeIdx[1]
            >> boundarys[i].cond
```

```
            >> boundarys[i].type;
    fin.close();

    std::sort(
        boundarys.begin(),
        boundarys.end(),
        [](Union::Boundary& _left, Union::Boundary& _right){
            return _left.cond > _right.cond;
        }
    );

    return isError;
}

void FEM::writeFile(
        const std::filesystem::path& _path,
        const double _eps,
        const size_t _max_iter) const {

    std::filesystem::create_directories(_path);
    bool is_dir = std::filesystem::is_directory(_path);

    using namespace ::Log;
    if (not is_dir) assert(
        Logger::append(getLog("Error - create directory"))
    );

    std::ofstream fout(_path / "kuslau.txt");
    fout << _size.nodes             << '\n';
    fout << std::scientific << _eps << '\n';
    fout << _max_iter;
    fout.close();

    Output::write(_path / "gg.txt", gg, { 14, ' ' });
    Output::write(_path / "di.txt", di, { 14, ' ' });
    Output::write(_path / "jg.txt", jg);
    Output::write(_path / "ig.txt", ig);
    Output::write(_path / "gb.txt", gb);
}

void FEM::resize() {
    nodes.    resize(  _size.nodes  );
    elems.    resize(  _size.elems  );
    boundarys.resize(  _size.conds  );
    materials.resize(  _size.areas  );

    gb.resize(  _size.nodes  );
    di.resize(  _size.nodes  );
    ig.resize(_size.nodes + 1);
```

```
}
#undef FIRST_BOUNDARY_COND
#undef SECOND_BOUNDARY_COND
#undef THIRD_BOUNDARY_COND
#endif /// _FEM_HPP_
```

Data.hpp

```
#ifndef _DATA_HPP_
#define _DATA_HPP_
#include "../utils/friendly.hpp"
#include "../Logger.hpp"

#include <filesystem>
#include <iostream>
#include <cassert>
#include <fstream>
#include <sstream>
#include <vector>
#include <cmath>

#define _SYMMETRIC_BEG namespace Symmetric {
#define _SYMMETRIC_END                     }

_SYMMETRIC_BEG

struct Param {
    size_t n;
    double epsilon;
    size_t max_iter;
};

enum class Cond {
    NONE,
    DIAGONAL,
    HOLLESKY
};

template <class T>
class Data
{
protected:
    Param param;
    std::vector<size_t> ig;
    std::vector<size_t> jg;
    std::vector<T> di;
    std::vector<T> gg;
    std::vector<T> b;
```

```
41      std::vector<T> x;
42
43      std::vector<T> di_l;
44      std::vector<T> gg_l;
45      std::vector<T> y;
46      size_t iter{ 0 };
47
48  public:
49      Data(std::filesystem::path _path) { assert(loadData(_path)); }
50      Data(Friendly* _friend, size_t _n, T _eps, size_t _max_iter) {
51          param.n        = _n;
52          param.epsilon  = _eps;
53          param.max_iter = _max_iter;
54
55          ig = _friend->ig;
56          jg = _friend->jg;
57          gg = _friend->gg;
58          di = _friend->di;
59          b  = _friend->gb;
60
61          x.resize(_n);
62          delete _friend;
63      }
64      ~Data() { }
65
66      std::vector<T>& getX() const { return x;    }
67      size_t  getIteration() const { return iter; }
68      void printX(std::streamsize count = 0) const;
69
70      void convertToLU();
71      std::vector<T> normal (std::vector<T> b);
72      std::vector<T> reverse(std::vector<T> y);
73      std::vector<T> mult(const std::vector<T>& _vec);
74
75  private:
76      bool loadData(std::filesystem::path _path);
77  };
78
79  template <class T>
80  void Data<T>::convertToLU() {
81      di_l = di;
82      gg_l = gg;
83
84      for (size_t i = 0; i < param.n; i++) {
85          T sum_diag = 0;
86          for (size_t j = ig[i]; j < ig[i + 1] ; j++) {
87              T sum = 0;
88              size_t jk = ig[jg[j]];
89              size_t ik = ig[i];
```

```
            while ((ik < j) && (jk < ig[jg[j] + 1]))
            {
                size_t l = jg[jk] - jg[ik];
                if (l == 0) {
                    sum += gg_l[jk] * gg_l[ik];
                    ik++; jk++;
                }
                jk += (l < 0);
                ik += (l > 0);
            }
            gg_l[j]  -= sum;
            gg_l[j]  /= di_l[jg[j]];
            sum_diag += gg_l[j] * gg_l[j];
        }
        di_l[i] -= sum_diag;
        di_l[i] = sqrt(fabs(di_l[i]));
    }
}

template <class T>
std::vector<T> Data<T>::normal(std::vector<T> b) {
    for (size_t i = 0; i < param.n; i++) {
        for (size_t j = ig[i]; j < ig[i + 1]; j++)
            b[i] -= gg_l[j] * b[jg[j]];

        b[i] = b[i] / di_l[i];
    }
    return b;
}

template <class T>
std::vector<T> Data<T>::reverse(std::vector<T> x) {
    for (int j = param.n - 1; j >= 0; j--) {
        x[j] = x[j] / di_l[j];

        for (size_t i = ig[j]; i < ig[j + 1]; i++)
            x[jg[i]] -= gg_l[i] * x[j];
    }
    return x;
}

template <class T>
std::vector<T> Data<T>::mult(const std::vector<T>& _vec) {
    std::vector<T> pr(_vec.size());

    int jj = 0;
    for (size_t i = 0; i < _vec.size(); i++) {
        pr[i] = di[i] * _vec[i];

```

```
        for (size_t j = ig[i]; j < ig[i + 1]; j++, jj++) {
            pr[i] += gg[jj] * _vec[jg[jj]];
            pr[jg[jj]] += gg[jj] * _vec[i];
        }
    }
    return pr;
}

template <class T>
void Data<T>::printX(std::streamsize count) const {
    std::ostringstream ostream;
    ostream << '\n';
    if (count) {
        ostream.setf(std::ios::fixed);
        ostream.precision(count);
    }
    ostream << "[ ";
    for (size_t i = 0; i < x.size(); i++)
        ostream << x[i] << " ";
    ostream << "]\n";
    std::cout << ostream.str();
}

template <typename T>
bool read(std::filesystem::path _path, std::vector<T>& _vec) {
    using namespace ::Log;
    std::ifstream fin(_path);
    if (not
        is_open(fin, "Error - " + _path.filename().string()))
        return false;
    for (size_t i = 0; i < _vec.size(); i++)
            fin >> _vec[i];
    fin.close(); return true;
}

template <class T>
bool Data<T>::loadData(std::filesystem::path _path) {
    using namespace ::Log;
    std::ifstream fin(_path / "kuslau.txt");
    if (not is_open(fin, "Error - kuslau.txt"))
        return false;
    fin >> param.n
        >> param.epsilon
        >> param.max_iter;
    fin.close();

    bool is_cor {  true  };
    ig.resize(param.n + 1);

```

```
    is_cor &= read(_path / "ig.txt", ig);

    gg.resize(ig.back());
    jg.resize(ig.back());
    di.resize( param.n );

    b.resize (param.n);
    x.resize (param.n);

    is_cor &= read(_path / "gg.txt", gg);
    is_cor &= read(_path / "di.txt", di);
    is_cor &= read(_path / "jg.txt", jg);
    is_cor &= read(_path / "gb.txt", b);
    return is_cor;
}
_SYMMETRIC_END
#undef _SYMMETRIC_DEGIN
#undef _SYMMETRIC_END
#endif /// _DATA_HPP_
```

LOS.hpp

```
#ifndef _LOS_HPP_
#define _LOS_HPP_
#include "Data.hpp"

#include "LOS_Function.hpp"

using namespace Symmetric;

#define LOGGER if (isLog)   \
    printLog(this->iter, eps);

template <class T>
class LOS : public Data<T>
{
public:
    LOS(std::filesystem::path _path) : Data<T>(_path) { }
    LOS(Friendly* _friend, size_t _n,  T _eps, size_t _max_iter)
        : Data<T>(_friend, _n, _eps, _max_iter) { }

    ~LOS() { }

    void solve(Cond _cond, bool isLog = true);
private:
    void none    (bool);
    void diagonal(bool);
    void hollesky(bool);
```

```
};

template <class T>
void LOS<T>::solve(Cond _cond, bool isLog) {
    using namespace ::Log;
    std::streamsize p = std::cout.precision();
    std::cout.precision(2);
    std::cout.setf(std::ios::uppercase);
    switch (_cond) {
        case Cond::NONE:     none(  isLog  ); break;
        case Cond::DIAGONAL: diagonal(isLog); break;
        case Cond::HOLLESKY: hollesky(isLog); break;
        default:
            Logger::append(getLog("this conditional non exist"));
            std::exit(1);
    }
    std::cout.unsetf(std::ios::scientific);
    std::cout.unsetf(std::ios::uppercase);
    std::cout.precision(p);
}

template <class T>
void LOS<T>::none(bool isLog) {
    std::vector<T> r (this->param.n),
                   z (this->param.n),
                   p (this->param.n),
                   Ar(this->param.n);

    r = this->b - this->mult(this->x);
    z = r;
    p = this->mult(z);

    T alpha, betta, eps;
    do {
        betta   = scalar(p, p);
        alpha   = scalar(p, r) / betta;
        this->x = this->x + alpha * z;
        r       = r - alpha * p;
        Ar      = this->mult(r);
        betta   = scalar(p, Ar) / betta;
        z       = r  - betta * z;
        p       = Ar - betta * p;
        eps     = scalar(r, r);

        this->iter++;
        LOGGER

    } while(
        this->iter < this->param.max_iter
```

```
76              && eps > this->param.epsilon);
77  }
78
79  template <class T>
80  void LOS<T>::diagonal(bool isLog) {
81      std::vector<T> r (this->param.n),
82                     z (this->param.n),
83                     p (this->param.n),
84                     Ar(this->param.n);
85
86      std::vector<T> L(this->param.n, 1);
87      for (size_t i = 0; i < L.size(); i++)
88          L[i] /= sqrt(this->di[i]);
89
90      r = L * (this->b - this->mult(this->x));
91      z = L * r;
92      p = L * this->mult(z);
93
94      T alpha, betta, eps;
95      do {
96          betta   = scalar(p, p);
97          alpha   = scalar(p, r) / betta;
98          this->x = this->x + alpha * z;
99          r       = r - alpha * p;
100         Ar      = L * this->mult(L * r);
101         betta   = scalar(p, Ar) / betta;
102         z       = L * r - betta * z;
103         p       = Ar - betta * p;
104         eps     = scalar(r, r);
105
106         this->iter++;
107         LOGGER
108
109     } while(
110         this->iter < this->param.max_iter
111             && eps > this->param.epsilon);
112 }
113
114 template <class T>
115 void LOS<T>::hollesky(bool isLog) {
116     std::vector<T> r  (this->param.n),
117                    z  (this->param.n),
118                    p  (this->param.n),
119                    Ar (this->param.n),
120                    LAU(this->param.n);
121
122     this->convertToLU();
123     r = this->normal(this->b - this->mult(this->x));
124     z = this->reverse(r);
```

```
    p = this->normal(this->mult(z));

    T alpha, betta, eps;
    do {
        betta   = scalar(p, p);
        alpha   = scalar(p, r) / betta;
        this->x = this->x + alpha * z;
        r       = r - alpha * p;
        LAU     = this->normal(this->mult(this->reverse(r)));
        betta   = scalar(p, LAU) / betta;
        z       = this->reverse(r) - betta * z;
        p       = LAU - betta * p;
        eps     = scalar(r, r);

        this->iter++;
        LOGGER
    }
    while(
        this->iter < this->param.max_iter
            && eps > this->param.epsilon);
}
#undef LOGGER
#endif /// _LOS_HPP_
```

LOS_Function.hpp

```
#ifndef _LOS_FUNCTION_HPP_
#define _LOS_FUNCTION_HPP_

#include <numeric>
#include <vector>
#include <cmath>

template <typename T>
inline void printLog(size_t _iter, T _eps) {
    std::cout << "Iteration = "   << std::fixed      << _iter << "\t\t" <<
                 "Discrepancy = " << std::scientific << _eps  << std::endl;
}

template <typename T>
T scalar(const std::vector<T>& _v1, const std::vector<T>& _v2) {
    T _res = 0;
    for (size_t i = 0; i < _v1.size(); i++)
        _res += _v1[i] * _v2[i];
    return _res;
}

template <typename T>
```

```cpp
T norm (const std::vector<T>& _v) {
    return sqrt(std::accumulate(_v.begin(), _v.end(), 0.0,
        [] (double _S, const double &_El) { return _S + _El * _El; }));
}

template <typename T>
std::vector<T> operator* (std::vector<T> _v1, const std::vector<T>& _v2) {
    for (size_t i = 0; i < _v1.size(); i++)
        _v1[i] *= _v2[i];
    return _v1;
};

template <typename T>
std::vector<T> operator- (std::vector<T> _v1, const std::vector<T>& _v2) {
    for (size_t i = 0; i < _v1.size(); i++)
        _v1[i] -= _v2[i];
    return _v1;
};

template <typename T>
std::vector<T> operator+ (std::vector<T> _v1, const std::vector<T>& _v2) {
    for (size_t i = 0; i < _v1.size(); i++)
        _v1[i] += _v2[i];
    return _v1;
};

template <typename T>
std::vector<T> operator* (T _alpha, std::vector<T> _v1) {
    for (size_t i = 0; i < _v1.size(); i++)
        _v1[i] *= _alpha;
    return _v1;
};
#endif // _LOS_FUNCTION_HPP_
```

lightweight.hpp

```cpp
#ifndef _LIGHTWEIGHT_HPP_
#define _LIGHTWEIGHT_HPP_
#include "../Union.hpp"

#include <array>
#include <cmath>

double
determinant(const std::array<Union::XY, 3>& elem) {
    return (
        (elem[1].x - elem[0].x) * (elem[2].y - elem[0].y) -
        (elem[1].y - elem[0].y) * (elem[2].x - elem[0].x)
```

```
    );
}

double
edgeLength(const std::array<Union::XY, 2>& elem) {
    return (
        sqrt (
            pow(elem[1].x - elem[0].x ,2) +
            pow(elem[1].y - elem[0].y, 2)
        )
    );
}
#endif // _LIGHTWEIGHT_HPP_
```

overload.hpp

```
#ifndef _OVERLOAD_HPP_
#define _OVERLOAD_HPP_
#include <iostream>
#include <vector>
#include <array>

namespace array {
    using x   = std::array<double, 3>;
    using xxx = std::array<std::array<double, 3>, 3>;
}

array::xxx
operator+ (const array::xxx& G, const array::xxx& M) {
    array::xxx _res;
    for (size_t i = 0; i < G.size(); i++)
    for (size_t j = 0; j < G.size(); j++)
        _res[i][j] = G[i][j] + M[i][j];
    return _res;
}
#endif // _OVERLOAD_HPP_
```

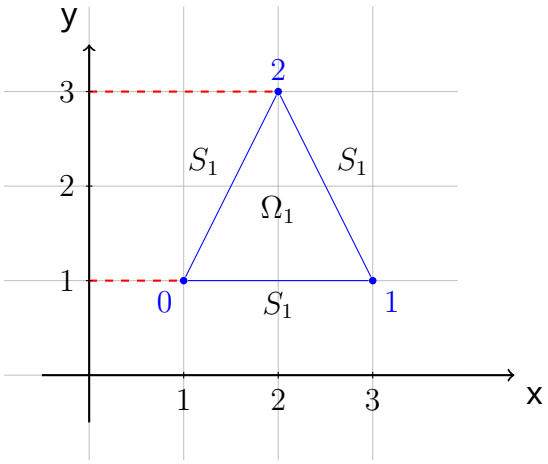
3 Тестирование

Тест №1

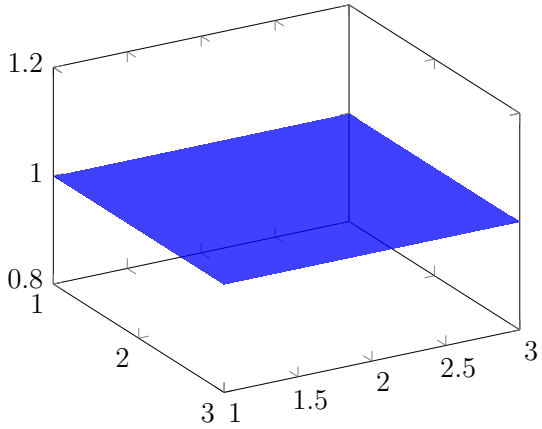
$$u(x,y) = 1$$

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

$\beta = 0$



$I_0 = 1$



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

Тест №3

Тест №4

Тест №5

4 Выводы

Выводы