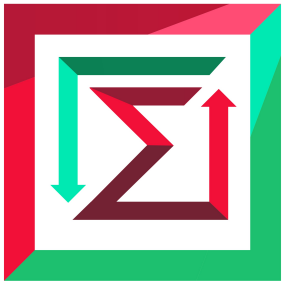
МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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 <streambuf>
#include <iostream>
#include <optional>
#include <fstream>
#include <memory>

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.filename().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::DIAGONAL, true);   /// solve LOS + DIAGONAL
        _timer.stop();                      /// stop timer

        _FEM.printAll();                    /// print input FEM data
        _FEM.printSparse();                 /// print sparse format
        _LOS.printX();                      /// print solution vector

        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;
}
```

FEM.hpp

```
#ifndef _FEM_HPP_
#define _FEM_HPP_
#include "utils/lightweight.hpp"
```

```cpp
#include "utils/overload.hpp"
#include "Function.hpp"
#include "Logger.hpp"
#include "Union.hpp"

#include <algorithm>
#include <cmath>
#include <set>

#define FIRST_BOUNDARY_COND  1
#define SECOND_BOUNDARY_COND 2
#define THIRD_BOUNDARY_COND  3

class FEM
{
private:
    Union::Param _size;

    std::vector<Union::XY>       nodes;
    std::vector<Union::Element>  elems;
    std::vector<Union::Boundary> boundarys;
    std::vector<Union::Material> materials;

    std::vector<double> gb;
    std::vector<double> gg;
    std::vector<double> di;
    std::vector<size_t> ig;
    std::vector<size_t> jg;

public:
    FEM(std::filesystem::path _path) {
        assert(readFile(_path));
        portrait(true);
        global();
        boundary ondition();
    }
    ~FEM() { }

    void printAll()    const;
    void printSparse() const;

    void writeFile(
        const std::filesystem::path&,
        const double,
        const size_t
```

```
    ) const;

private:
    void global();

    template<size_t N, typename _Struct>
    void loc_A_to_global(
        const std::array<std::array<double, N>, N>&,
        const _Struct&
    );

    template<size_t N, typename _Struct>
    void loc_b_to_global(const std::array<double, N>&, const _Struct& );

    array::xxx localA(const std::array<Union::XY, 3>&, size_t) const;
    array::x   buildF(const std::array<Union::XY, 3>&, size_t) const;

    array::xxx G(const std::array<Union::XY, 3>&, size_t) const;
    array::xxx M(const std::array<Union::XY, 3>&, size_t) const;

    bool readFile(const std::filesystem::path&  );
    void portrait(const bool isWriteList = false);

    void boundary ondition();
    void first (const Union::Boundary& bound);
    void second(const Union::Boundary& bound);
    void third (const Union::Boundary& bound);

    void resize();
};

void FEM::global() {

    std::array<Union::XY, 3> coords;

    for (size_t i = 0; i < _size.elems; i++) {
        for (size_t j = 0; j < 3; j++) {
            size_t point = elems[i].nodeIdx[j];
            coords[j].x = nodes[point].x;
            coords[j].y = nodes[point].y;
        }
        array::x   local_b = buildF(coords, elems[i].area);
        array::xxx local_A = localA(coords, elems[i].area);

        pretty(local_A);
```

```
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::boundary ondition() {
101     using namespace ::Log;
102
103     for (size_t _count = 0; _count < _size.conds; _count++) {
104
105         switch (boundarys[_count].cond)
106         {
107             case FIRST_BOUNDARY_COND:
108                 first(boundarys[_count]);
109                 break;
110             case SECOND_BOUNDARY_COND:
111                 second(boundarys[_count]);
112                 break;
113             case THIRD_BOUNDARY_COND:
114                 third(boundarys[_count]);
115                 break;
116             default:
117                 Logger::append(getLog("There is no such condition"));
118         }
119     }
120 }
121
122 void FEM::first(const Union::Boundary& bound) {
123     di[bound.nodeIdx[0]] = { 1 };
124     di[bound.nodeIdx[1]] = { 1 };
125
126     for (size_t i = 0; i < 2; i++)
127         gb[bound.nodeIdx[i]] =
128             Function::firstBound({
129                 nodes[bound.nodeIdx[i]].x,
130                 nodes[bound.nodeIdx[i]].y
131             }, bound.type);
132
133     for (size_t k = 0; k < 2; k++) {
134         size_t node = bound.nodeIdx[k];
135         for (size_t i = ig[node]; i < ig[node + 1]; i++) {
136             gb[jg[i]] -= gg[i] * gb[node];
137             gg[i] = 0;
138         }
```

```
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(int p = lbeg; p < lend; p++) {
144                 if(jg[p] == node) {
145                     gb[i] -= gg[p] * gb[node];
146                     gg[p] = 0;
147                 }
148             }
149         }
150     }
151 }
152
153 void FEM::second(const Union::Boundary& bound) {
154
155     std::array<Union::XY, 2>
156         coord_borders = {
157             nodes[bound.nodeIdx[0]],
158             nodes[bound.nodeIdx[1]]
159         };
160
161     double _koef = edgeLength(coord_borders) / 6;
162
163     std::array<double, 2> corr_b;
164     for (size_t i = 0; i < 2; i++)
165         corr_b[i] = _koef * (
166             2 * Function::secondBound({
167                     nodes[bound.nodeIdx[i]].x,
168                     nodes[bound.nodeIdx[i]].y
169                 }, bound.type) +
170                 Function::secondBound({
171                     nodes[bound.nodeIdx[1 - i]].x,
172                     nodes[bound.nodeIdx[1 - i]].y
173                 }, bound.type)
174             );
175
176     loc_b_to_global<2>(corr_b, bound);
177 }
178
179 void FEM::third(const Union::Boundary& bound) {
180
181     std::array<Union::XY, 2> coord_borders = {
182         nodes[bound.nodeIdx[0]],
183         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 (int 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 = abs(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
    std::array<std::array<double, 3>, 3> G = FEM::G(elem, area);
    std::array<std::array<double, 3>, 3> M = FEM::M(elem, area);
```

```
272     std::array<std::array<double, 3>, 3> A = G + M;
273     return A;
274 
275 }
276 
277 array::xxx FEM::G(const std::array<Union::XY, 3>& elem, size_t area) const {
278     double det   = abs(determinant(elem));
279     double _koef = Function::lambda(area) / (2 * det);
280 
281     std::array<std::array<double, 3>, 3> G;
282     std::array<std::array<double, 2>, 3> a {
283 
284             elem[1].y - elem[2].y,
285             elem[2].x - elem[1].x,
286 
287             elem[2].y - elem[0].y,
288             elem[0].x - elem[2].x,
289 
290             elem[0].y - elem[1].y,
291             elem[1].x - elem[0].x
292     };
293 
294     for (int i = 0; i < 3; i++)
295     for (int j = 0; j < 3; j++)
296         G[i][j] = _koef * (a[i][0] * a[j][0] + a[i][1] * a[j][1]);
297 
298     return G;
299 }
300 
301 array::xxx FEM::M(const std::array<Union::XY, 3>& elem, size_t area) const {
302     double det = abs(determinant(elem));
303     double gammaKoef = materials[area].gamma * det / 24;
304     std::array<std::array<double, 3>, 3> M;
305     for (size_t i = 0; i < 3; i++)
306     for (size_t j = 0; j < 3; j++) {
307         M[i][j] =
308             (i == j) ? (2 * gammaKoef) :
309                        (    gammaKoef) ;
310     }
311     return  M;
312 }
313 
314 void FEM::portrait(const bool isWriteList) {
315 
316     const size_t N {    _size.nodes     };
```

```
317     std::vector<std::set<size_t>> list(N);
318 
319     for (size_t el = 0; el < _size.elems; el++)
320     for (size_t point = 0; point < 3; point++) {
321         for (size_t i = point + 1; i < 3; i++) {
322             size_t idx1 = { elems[el].nodeIdx[point] };
323             size_t idx2 = { elems[el].nodeIdx[  i  ] };
324             idx1 > idx2 ?
325                 list[idx1].insert(idx2) :
326                 list[idx2].insert(idx1) ;
327         }
328     }
329 
330     for (size_t i = 2; i < ig.size(); i++)
331         ig[i] = ig[i - 1] + list[i - 1].size();
332 
333     jg.resize(ig[N] - ig[0]);
334     gg.resize(ig[N] - ig[0]);
335 
336    for (size_t index = 0, i = 1; i < list.size(); i++)
337     for (size_t value : list[i])
338         jg[index++] = value;
339 
340     if (isWriteList) {
341         std::cout << "list: " << '\n';
342         for (size_t i = 0; i < list.size(); i++) {
343             std::cout << i << ':' << ' ';
344             for (size_t j : list[i])
345                 std::cout << j << ' ';
346             std::cout << std::endl;
347         }
348     }
349 }
350 
351 void FEM::printAll() const {
352     #define PRINTLINE \
353         for (size_t i = 0; i < 20; std::cout << '-', i++);
354     #define ENDLINE std::cout << '\n';
355     SetConsoleOutputCP(65001);
356     PRINTLINE ENDLINE
357     std::cout << "PARAMS:         "                << '\n';
358     std::cout << "Size nodes:     " << _size.nodes  << '\n';
359     std::cout << "Size element:   " << _size.elems  << '\n';
360     std::cout << "Size areas:     " << _size.areas  << '\n';
361     std::cout << "Size condition: " << _size.conds  << '\n';
```

```
362     PRINTLINE ENDLINE
363     std::cout << std::setw(4) << "X" << std::setw(4) << "Y" << '\n';
364     for (size_t i = 0; i < _size.nodes; i++)
365         std::cout << std::setw(4) << nodes[i].x
366                   << std::setw(4) << nodes[i].y << '\n';
367     PRINTLINE ENDLINE
368     std::cout << "Elements: " << '\n';
369     for (size_t i = 0; i < _size.elems; i++)
370         std::cout << elems[i].nodeIdx[0] << ' '
371                   << elems[i].nodeIdx[1] << ' '
372                   << elems[i].nodeIdx[2] << " -> area "
373                   << elems[i].area       << '\n';
374     PRINTLINE ENDLINE
375     std::cout << "Areas: " << '\n';
376     for (size_t i = 0; i < _size.areas; i++) {
377         std::cout << "\u03B3 = " << materials[i].gamma  << ',' << ' '
378                   << "\u03B2 = " << materials[i].betta
379                   << " -> area " << i << '\n';
380     }
381     PRINTLINE ENDLINE
382     std::cout << "Borders: " << '\n';
383     for (size_t i = 0; i < _size.conds; i++)
384         std::cout << boundarys[i].area       << ' '
385                   << boundarys[i].nodeIdx[0] << ' '
386                   << boundarys[i].nodeIdx[1] << ' '
387                   << boundarys[i].cond       << ' '
388                   << boundarys[i].type       << ' ' << '\n';
389     PRINTLINE ENDLINE
390     #undef PRINTLINE
391     #undef ENDLINE
392 }
393
394 void FEM::printSparse() const {
395     #define PRINTLINE \
396         for (size_t i = 0; i < 20; std::cout << '-', i++); \
397         std::cout << '\n';
398
399     PRINTLINE
400     std::cout << "ig: "; print(ig);
401     std::cout << "jg: "; print(jg);
402     std::cout << "di: "; print(di);
403     std::cout << "gg: "; print(gg);
404     PRINTLINE
405
406     #undef PRINTLINE
```

```
}

bool FEM::readFile(const std::filesystem::path& path) {
    using namespace ::Log;
    bool isError { true };

    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++)
^^I^^Ifin >> nodes[i].x >> nodes[i].y;
    fin.close();

    fin.open(path / "elems.txt");
    isError &= is_open(fin, getLog("Error - elems.txt"));
^^Ifor (size_t i = 0; i < _size.elems; i++) {
^^I^^Ifin >> 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++)
```

```
452         fin >> boundarys[i].area
453             >> boundarys[i].nodeIdx[0]
454             >> boundarys[i].nodeIdx[1]
455             >> boundarys[i].cond
456             >> boundarys[i].type;
457     fin.close();
458     return isError;
459 }
460
461 void FEM::writeFile(
462         const std::filesystem::path& _path,
463         const double _eps,
464         const size_t _max_iter) const {
465
466     std::filesystem::create_directories(_path);
467     bool is_dir = std::filesystem::is_directory(_path);
468
469     using namespace ::Log;
470     if (not is_dir) assert(
471         Logger::append(getLog("Error - create directory"))
472     );
473
474     std::ofstream fout(_path / "kuslau.txt");
475     fout << _size.nodes             << '\n';
476     fout << std::scientific << _eps << '\n';
477     fout << _max_iter;
478     fout.close();
479
480     Output::write(_path / "gg.txt", gg, { 14, ' ' });
481     Output::write(_path / "di.txt", di, { 14, ' ' });
482     Output::write(_path / "jg.txt", jg);
483     Output::write(_path / "ig.txt", ig);
484     Output::write(_path / "gb.txt", gb);
485 }
486
487 void FEM::resize() {
488     nodes.    resize(  _size.nodes  );
489     elems.    resize(  _size.elems  );
490     boundarys.resize(  _size.conds  );
491     materials.resize(  _size.areas  );
492
493     gb.resize(  _size.nodes  );
494     di.resize(  _size.nodes  );
495     ig.resize(_size.nodes + 1);
496 }
```

```
497
498 #undef FIRST_BOUNDARY_COND
499 #undef SECOND_BOUNDARY_COND
500 #undef THIRD_BOUNDARY_COND
501 #endif /// _FEM_HPP_
```

3 Тестирование

4 Выводы