



Министерство науки и высшего образования Российской Федерации  
федеральное государственное автономное образовательное учреждение высшего  
образования «Московский государственный технический университет имени  
Н.Э. Баумана (национальный исследовательский университет)»

ФАКУЛЬТЕТ «Робототехники и комплексной автоматизации»  
КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

## ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

### по дисциплине «Модели и методы анализа проектных решений»

Студент:	Олейников Антон Александрович
Группа:	РК6-63Б
Тип задания:	лабораторная работа
Тема:	Метод конечных разностей при ре- шении задачи теплопроводности

Студент

подпись, дата

Олейников А.А.  
Фамилия, И.О.

Преподаватель

подпись, дата

Трудоношин В.А.  
Фамилия, И.О.

Москва, 2025

## Содержание

<b>Метод конечных разностей при решении задачи теплопроводности</b>	<b>3</b>
Цель выполнения лабораторной работы . . . . .	3

# Метод конечных разностей при решении задачи теплопроводности

## Задание

С помощью неявной разностной схемы решить нестационарное уравнение теплопроводности для пластины.

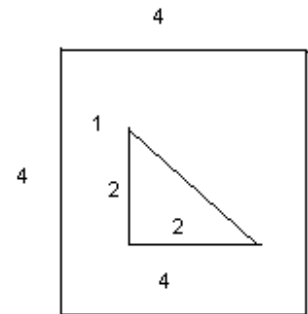
Начальное значение температуры пластины - 30 градусов.

Граничные условия следующие:

1. Внизу теплоизоляция;
2. Внутри  $dT/dn = T$ ;
3. На остальной границе 70 гр.

При выводе результатов показать динамику изменения температуры (например с помощью цветовой гаммы).

Отчет должен содержать: текст программы, рисунок объекта с распределением температуры в момент времени 25 сек сравнение результатов расчета с результатами, полученными с помощью пакета ANSYS.



## Цель выполнения лабораторной работы

Цель выполнения лабораторной работы: с помощью неявной разностной схемы решить нестационарное уравнение теплопроводности для пластины.

## Решение

Нестационарное уравнение теплопроводности в однородной изотропной пластине может быть представлено, как:

$$\rho c \frac{\partial T}{\partial t} = \lambda \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right),$$

где  $T(x, y, t)$  – температура,  $\rho$  – плотность,  $c$  – удельная теплоёмкость,  $\lambda$  – коэффициент теплопроводности.

Обозначив  $a = \lambda/(\rho c)$ , получим:

$$\frac{\partial T}{\partial t} = a \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right).$$

Дискретизируем область равномерной сеткой с шагами  $\Delta x$ ,  $\Delta y$  по пространству и  $\Delta t$  по времени. Запишем неявную разностную схему:

$$\frac{T_{i,j}^{k+1} - T_{i,j}^k}{\Delta t} = a \left( \frac{T_{i+1,j}^{k+1} - 2T_{i,j}^{k+1} + T_{i-1,j}^{k+1}}{(\Delta x)^2} + \frac{T_{i,j+1}^{k+1} - 2T_{i,j}^{k+1} + T_{i,j-1}^{k+1}}{(\Delta y)^2} \right),$$

где  $T_{i,j}^k$  – приближение температуры в узле  $(x_i, y_j)$  на  $k$ -м временном слое.

Перегруппируем уравнение относительно приближенных значений температуры в узлах:

$$\begin{aligned}
& T_{i,j}^{k+1} * ((dx)^2(dy)^2 + 2a(dy)^2dt + 2a(dx)^2dt) \\
& + T_{i-1,j}^{k+1} * (-a(dy)^2dt) \\
& + T_{i+1,j}^{k+1} * (-a(dy)^2dt) \\
& + T_{i,j-1}^{k+1} * (-a(dx)^2dt) \\
& + T_{i,j+1}^{k+1} * (-a(dx)^2dt) \\
& = T_{i,j}^k * ((dx)^2(dy)^2)
\end{aligned}$$

Рассмотрим учёт граничных условий:

- Первого рода (Дирихле): задаётся значение температуры  $T = T_w$ .  
Разностный аналог:  $T_{i,j} = T_w$ .
- Второго рода (Неймана): задаётся тепловой поток  $\lambda \partial T / \partial n = q_n$ . В частности, для теплоизоляции  $q_n = 0$ , что даёт  $\partial T / \partial n = 0$ .  
Разностный аналог:  $T_{n_i, n_j}^{k+1} - T_{i,j}^{k+1} = \sqrt{(dx)^2(n_i - i)^2 + (dy)^2(n_j - j)^2} \cdot q_n / \lambda$ .
- Третьего рода: характеризует закон конвективного теплообмена между поверхностью тела и окружающей средой (закон Ньютона):  $-\lambda \frac{\partial T}{\partial n} = \sigma(T - T_{\text{окр}})$ . ( $\sigma$  - коэффициент теплоотдачи).  
Разностный аналог:  $T_{n_i, n_j}^{k+1} - T_{i,j}^{k+1} \left( 1 - \sqrt{(dx)^2(n_i - i)^2 + (dy)^2(n_j - j)^2} \cdot \frac{\sigma}{\lambda} \right) = \frac{\sigma}{\lambda} \cdot \sqrt{(dx)^2(n_i - i)^2 + (dy)^2(n_j - j)^2} \cdot T_{\text{окр}}$

Разностная аппроксимация приводит к системе линейных алгебраических уравнений вида:

$$A \mathbf{T}^{k+1} = \mathbf{b},$$

где  $\mathbf{T}^{k+1}$  – вектор неизвестных температур в узлах сетки на новом временном слое. Полученную СЛАУ будем решать методом Гаусса.

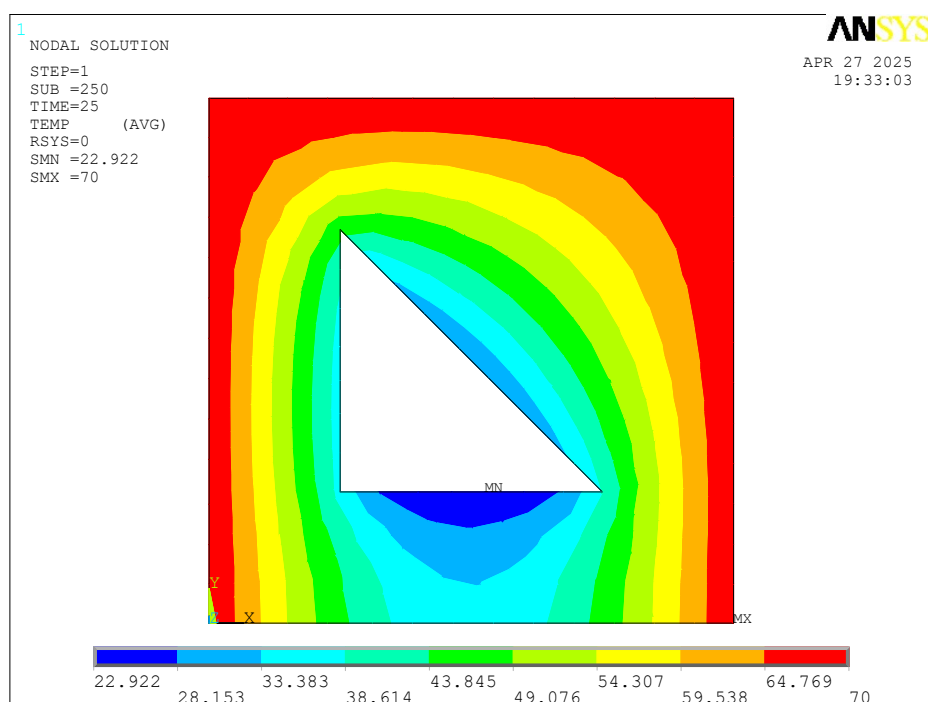


Рис. 1. Поле температур, полученное в ANSYS в момент времени  $t=25$ с.

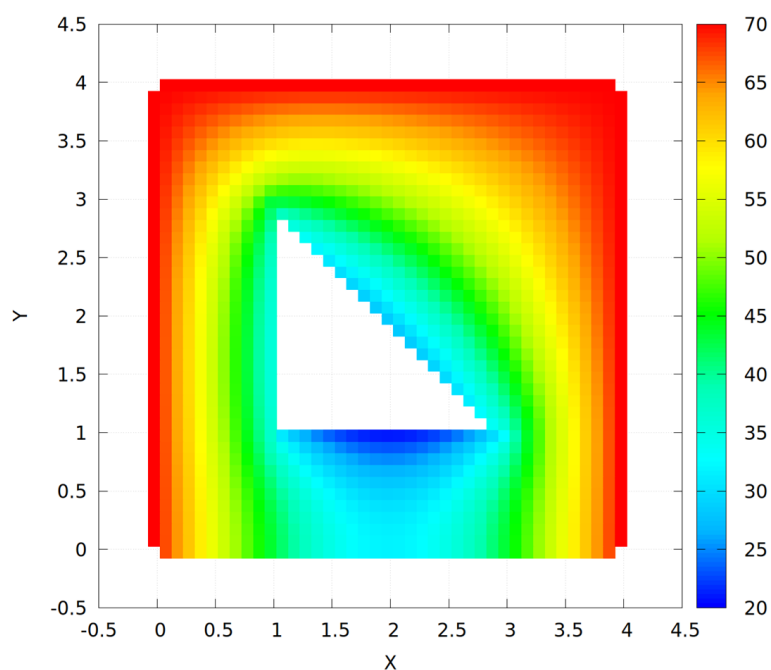


Рис. 2. Поле температур, полученное разработанной программой в момент времени  $t=25$ с.

Сравнение результатов расчётов программы (рис. 2) и моделирования в ANSYS (рис. 1) показывает их хорошее соответствие, что свидетельствует о правильности реализации алгоритма.

Листинг 1. Программная реализация рассмотренной в лабораторной работа задачи на языке программирования C++.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <cmath>
#include <cstdlib>
#include <unistd.h>
#include <stack>
#include <iomanip>
#include <unordered_map>

// Вспомогательный тип данных для хранения вектора В СЛАУ
template<typename T>
class OrderIndexMap {
public:
    OrderIndexMap(size_t expected_size) {
        map.reserve(expected_size); // заранее зарезервировать размер
    }

    // Возвращает индекс элемента: либо уже существующий, либо новый
    int get_or_add(const T& value) {
        auto it = map.find(value);
        if (it != map.end()) {
            return it->second;
        } else {
            int index = current_index++;
            map[value] = index;
            return index;
        }
    }

    void clear() {
        map.clear(); // Удаляем все элементы из хэш-таблицы
        current_index = 0; // Сбрасываем счётчик индексов
    }

private:
    std::unordered_map<T, int> map;
    int current_index = 0;
};

// Краевые условия:
// type 0 - начальные условия,  $T(t=0) = val$ 
// 1 - Dirichlet,  $T = val$ 
// 2 - Neumann,  $dT/dn = val/lambda$ 
// 3 - Robin,  $dT/dn = -sigma/lambda(T - val)$ 
class BoundaryCondition {
public:
    int type;
    double val;
    int n_i, n_j;
    BoundaryCondition(int _type, double _val, int _n_i, int _n_j) :
        type(_type), val(_val), n_i(_n_i), n_j(_n_j) {}
};

// Класс узла сетки
class Node {
public:
    BoundaryCondition bc;
    int i, j; // Индексы в КЭ сетке
    double x, y; // Координаты узла
    double t; // Температура узла
    Node(int _i, int _j, double _x, double _y, int _bc_type, double _bc_val,
        int _t = 0., int _n_i = 0, int _n_j = 0) :
        i(_i), j(_j), x(_x), y(_y), bc(_bc_type, _bc_val, _n_i, _n_j), t(_t) {}
};
```

```

// Класс сетки (также содержит данные о характеристиках)
class PlateMeshGrid {
public:
    std::vector<Node> nodes;
    int nRows, nCols;    // nRows - число строк (индекс i), nCols - число столбцов (индекс j)
    double dx, dy;
    double lambda, rho, c, sigma;

    // Чтение сетки из CSV файла
    int read_mesh_from_csv(const std::string& filename) {
        std::ifstream fin(filename);
        std::vector<std::string> lines;
        std::string line;

        // прочтение метаданных о сетке
        while (std::getline(fin, line)) {
            const auto first = line.find_first_not_of(" \t");
            if (first == std::string::npos || line[first] == '#')
                continue;
            std::istringstream ss(line);
            char delim;
            ss >> this->nCols >> delim
                >> this->nRows >> delim
                >> this->dx >> delim
                >> this->dy >> delim
                >> this->lambda >> delim
                >> this->rho >> delim
                >> this->c >> delim
                >> this->sigma;
            break; // метаданные о сетке прочитаны
        }

        while (std::getline(fin, line)) {
            const auto first = line.find_first_not_of(" \t");
            // если строка пустая или первый непробельный символ - '#', пропускаем её
            if (first == std::string::npos || line[first] == '#')
                continue;
            std::istringstream ss(line);
            int i, j, bc_type;
            double x, y, bc_val;
            int n_i, n_j;
            char delim;
            ss >> i >> delim
                >> j >> delim
                >> x >> delim
                >> y >> delim
                >> bc_type >> delim
                >> bc_val;
            if (bc_type == 2 || bc_type == 3) {
                ss >> delim >> n_i >> delim
                    >> n_j;
                nodes.emplace_back(i, j, x, y, bc_type, bc_val, 0., n_i, n_j);
            }
            else{
                nodes.emplace_back(i, j, x, y, bc_type, bc_val, bc_val);
            }
        }
        fin.close();
        return 0;
    }

    // Функция отрисовки (записи) распределения температуры
    int print_mesh(const std::string& filename) {
        std::ofstream fout(filename);
        fout << std::setprecision(6);

        for (const auto& node : nodes) {
            // Формат: X Y Temperature dX/2 dY/2
            fout << node.x << " " << node.y << " " << node.t
                << " " << dx/2 << " " << dy/2 << "\n";
        }
    }
};

```

```

    }
    fout.close();
    return 0;
}

};

// Класс решателя
class PlateTemperatureSolver {
public:
    int solve_mesh(PlateMeshGrid& mesh, const double time_final, const double dt) {
        const double lambda = mesh.lambda;
        const double rho = mesh.rho;
        const double c = mesh.c;
        const double sigma = mesh.sigma;
        const double alpha = lambda / rho / c;

        const int N = mesh.nodes.size();
        int timesteps = (int) time_final / dt;
        const int nodes_count = mesh.nodes.size();

        std::vector<std::vector<double>> A(N, std::vector<double>(N+1));
        OrderIndexMap<int> B(N);
        for (auto& node : mesh.nodes) {
            int tmp = B.get_or_add(node.j * mesh.nCols + node.i);
        }

        for (int k = 0; k < timesteps; k++) {
            for (auto& row : A) std::fill(row.begin(), row.end(), 0.0);

            for (int n = 0; n < nodes_count; n++) { // составляем СЛАУ
                Node node = mesh.nodes[n];

                int b_poz_i_j, b_poz_im_j, b_poz_ip_j, b_poz_i_jm, b_poz_i_jp, b_poz, b_poz_n;
                switch (node.bc.type) { // см описание класса
                    case 0:
                        b_poz_i_j = B.get_or_add(node.j * mesh.nCols + node.i);
                        b_poz_im_j = B.get_or_add(node.j * mesh.nCols + node.i - 1);
                        b_poz_ip_j = B.get_or_add(node.j * mesh.nCols + node.i + 1);
                        b_poz_i_jm = B.get_or_add((node.j - 1) * mesh.nCols + node.i);
                        b_poz_i_jp = B.get_or_add((node.j + 1) * mesh.nCols + node.i);

                        A[n][N] = mesh.dx*mesh.dx*mesh.dy*mesh.dy * mesh.nodes[b_poz_i_j].t;

                        A[n][b_poz_i_j] = (mesh.dx*mesh.dx*mesh.dy*mesh.dy) \
                            + 2*alpha*mesh.dy*mesh.dy*dt \
                            + 2*alpha*mesh.dx*mesh.dx*dt;

                        A[n][b_poz_im_j] = -alpha*mesh.dy*mesh.dy*dt;
                        A[n][b_poz_ip_j] = -alpha*mesh.dy*mesh.dy*dt;

                        A[n][b_poz_i_jm] = -alpha*mesh.dx*mesh.dx*dt;
                        A[n][b_poz_i_jp] = -alpha*mesh.dx*mesh.dx*dt;
                        break;
                    case 1:
                        b_poz = B.get_or_add(node.j * mesh.nCols + node.i);
                        A[n][b_poz] = 1.;
                        A[n][N] = node.bc.val;
                        break;
                    case 2:
                        b_poz = B.get_or_add(node.j * mesh.nCols + node.i);
                        b_poz_n = B.get_or_add(node.bc.n_j * mesh.nCols + node.bc.n_i);
                        A[n][b_poz_n] = 1.;
                        A[n][b_poz] = -1.;
                        A[n][N] = node.bc.val * sqrt((mesh.dx*mesh.dx*(node.bc.n_i-node.i) \
                            *(node.bc.n_i-node.i) \
                            + mesh.dy*mesh.dy*(node.bc.n_j-node.j) \
                            *(node.bc.n_j-node.j))) /

                        break;
                    case 3:
                        b_poz = B.get_or_add(node.j * mesh.nCols + node.i);
                        b_poz_n = B.get_or_add(node.bc.n_j * mesh.nCols + node.bc.n_i);

```



```

        A[n][b_poz] = (sqrt((mesh.dx*mesh.dx*(node.bc.n_i-node.i) \
        *(node.bc.n_i-node.i) \
        + mesh.dy*mesh.dy*(node.bc.n_j-node.j) \
        *(node.bc.n_j-node.j))) * sigma / lambda - 1.);

        A[n][b_poz_n] = 1.;
        A[n][N] = sqrt((mesh.dx*mesh.dx*(node.bc.n_i-node.i)\
        *(node.bc.n_i-node.i) \
        + mesh.dy*mesh.dy*(node.bc.n_j-node.j)\
        *(node.bc.n_j-node.j))) * node.bc.val \
        * sigma / lambda;
        break;
    }
}

// решение расширенной СЛАУ
gauss_elimination(A);

for (auto& node : mesh.nodes) {
    if (node.bc.type != 1) {
        node.t = A[B.get_or_add(node.j * mesh.nCols + node.i)][N];
    }
}

return 0;
}

private:
// 0 - СЛАУ несовместна
// -n - бесконечное количество решений (n - установленный ранг матрицы)
// n - СЛАУ имеет единственное решение (n - установленный ранг матрицы)
int gauss_elimination(std::vector<std::vector<double>>& mat) {
    gauss_forward_elimination(mat);
    int res = gauss_check_solutions(mat);
    if (res != 0)
        gauss_backward_elimination(mat);
    return res;
}

// Прямой ход для приведения к верхнетреугольному виду
void gauss_forward_elimination(std::vector<std::vector<double>>& mat) {
    for (int k = 0; k < mat.size(); k++) {
        /* поиск строки с максимальным абсолютным значением в текущем столбце
        (от строки k вниз), чтобы избежать деления на слишком маленькие числа
        и улучшить численную стабильность алгоритма. */
        int max_row = k;
        for (int i = k + 1; i < mat.size(); i++) {
            if (fabs(mat[i][k]) > fabs(mat[max_row][k])) {
                max_row = i;
            }
        }

        // Меняем строки, если необходимо
        if (max_row != k) {
            mat[k].swap(mat[max_row]);
        }

        // Нормализация основной строки
        double pivot = mat[k][k];
        for (int j = k; j < mat[k].size(); j++) {
            mat[k][j] /= pivot;
        }

        // Обнуление элементов под основной строкой
        for (int i = k + 1; i < mat.size(); i++) {
            double factor = mat[i][k];
            mat[i][k] = 0.;
            for (int j = k+1; j < mat[k].size(); j++) {
                mat[i][j] -= factor * mat[k][j];
            }
        }
    }
}

```

```

    }
}

// Обратный ход для приведения к диагональному виду
void gauss_backward_elimination(std::vector<std::vector<double>>& mat) {
    for (int k = mat.size() - 1; k >= 0; k--) {
        for (int i = k - 1; i >= 0; i--) {
            double factor = mat[i][k];
            mat[i][k] = 0.;
            for (int j = k + 1; j < mat[i].size(); j++) {
                mat[i][j] -= factor * mat[k][j];
            }
        }
    }
}

// Функция для проверки количества решений СЛАУ
// Принимает на вход матрицу верхнетреугольного вида
// 0 - СЛАУ несовместна (!!определяется только если переданная матрица была расширенной)
// -m - бесконечное количество решений (m - установленный ранг матрицы)
// n - СЛАУ имеет единственное решение (n - установленный ранг матрицы)
int gauss_check_solutions(std::vector<std::vector<double>>& mat) {
    int rank = 0;

    for (int i = 0; i < mat.size(); i++) {
        bool row_nonzero = false;
        for (int j = 0; j < mat[i].size() - 1; j++) {
            if (fabs(mat[i][j]) > 1e-12) { // Пороговое значение для сравнения с нулем
                row_nonzero = true;
                break;
            }
        }
        // 0 0 0 0 | 4
        if (!row_nonzero && fabs(mat[i].back()) > 1e-12)
            return 0;

        if (row_nonzero) rank++;
    }

    if (rank == mat.size())
        return rank;
    else
        return -rank;
}

};

// Определяет параметры моделирования, задаваемые пользователем
struct ModelParameters {
    double duration; // Продолжительность моделирования
    double dt; // Шаг по времени
    bool service_mode; // Флаг служебного режима без файлового вывода (для benchmark)
    std::string output_file; // Название файла для вывода результатов моделирования
    std::string mesh_file; // Название файла-источника сетки для моделирования
};

class ParameterParser {
public:
    ParameterParser(int c, char** v) : _argument_count(c), _argument_strings(v) {}

    // Разбирает аргументы, переданные при создании, для определения параметров моделирования.
    // выбрасывает std::invalid_argument при некорректных значениях параметров.
    ModelParameters getParameters() const {
        // Значения по умолчанию.
        double dt = 1.;
        double duration = 25.;
        std::string output_file = "data.txt";
        bool service_mode = false;
        std::string mesh_file;

        // Разбор аргументов при помощи getopt.
        int option = 0;

```

```

while ((option = getopt(_argument_count, _argument_strings, "i:o:t:d:s")) != -1) {
    switch (option) {
        case 't':
            duration = atof(optarg);
            if (duration <= 0.) {
                throw std::invalid_argument("ModelParameters: duration must be positive");
            }
            break;
        case 'd':
            dt = atof(optarg);
            if (dt <= 0.) {
                throw std::invalid_argument("ModelParameters: dt must be positive");
            }
            break;
        case 'o':
            output_file = optarg;
            break;
        case 'i':
            mesh_file = optarg;
        case 's':
            service_mode = true;
            break;
        default:
            break;
    }
}

if (mesh_file.empty()) {
    std::cout << "Parameter -i (mesh_file) is required" << std::endl;
    throw std::invalid_argument("Parameter -i (mesh_file) is required");
}

ModelParameters parameters;
parameters.output_file = output_file;
parameters.mesh_file = mesh_file;
parameters.duration = duration;
parameters.service_mode = service_mode;

return parameters;
}

private:
const int _argument_count; // Число аргументов запуска программы.
char** _argument_strings; // Массив строковых значений аргументов запуска программы.
};


int main(int argc, char** argv) {
    try {
        ParameterParser parser(argc, argv);
        ModelParameters params = parser.getParameters();
        PlateMeshGrid mesh;
        mesh.read_mesh_from_csv(params.mesh_file);


        PlateTemperatureSolver solver;
        solver.solve_mesh(mesh, params.duration, params.dt);
        mesh.print_mesh(params.output_file);
    } catch (...) {
        return 1;
    }
    std::cout << "Симуляция проведена успешно\n";
    return 0;
}

```

## Выходные данные

Олейников А.А. Отчет о выполнении лабораторной работы по дисциплине «Модели и методы анализа проектных решений». [Электронный ресурс] — Москва: 2025. — 12 с. URL: <https://gitlab.sa2systems.ru> (система контроля версий кафедры РК6)

Постановка:  доцент кафедры РК-6, к.т.н. Трудоношин В.А.

Решение:  студент группы РК6-63Б, Олейников А.А.

2025, весенний семестр