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

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: «МиМАПР»

Студент Шлюков Алексей

Группа РК6-64Б

Тип задания Лабораторная работа

Название «Метод конечных разностей»

Студент _____ **Шлюков А.П.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Боровкова Н.М.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2025 г.

Оглавление

Метод конечных разностей

Задание

В данной задаче необходимо найти распределение температуры в двумерной области (пластине) Ω[3], представленной на рис. 1. Пластина изготовлена из однородного материала. Единицы измерения: время – секунды (сек.), пространство – миллиметры (мм), температура – градусы Цельсия (C°).

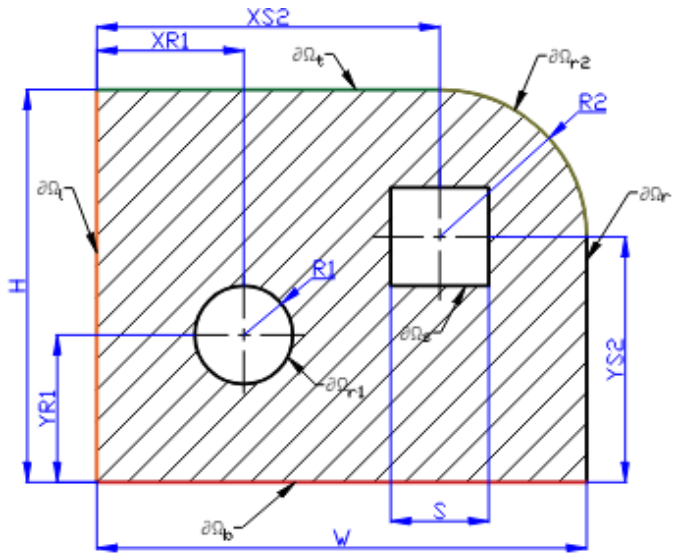


Рис. 1: Чертёж расчётной области пластины Ω

Основные размеры пластины: $H = 400$ мм, $W = 500$ мм, $R2 = 150$ мм, сторона квадратного отверстия $S = 100$ мм, радиус круглого отверстия $R1 = 50$ мм. В каждом варианте задания пластина имеет лишь одно отверстие из двух изображенных на рис. 1. Тип отверстия определяется в соответсвии с параметром $\gamma_1 \in \Gamma_1 = \{\partial\Omega_{r1}, \partial\Omega_s\}$ (пояснение на стр. 4). Координаты центра отверстия $(XR1, YR1)$ или $(XS2, YS2)$ определяются параметром $\gamma_2 \in \Gamma_2 = \{(155, 155), (155, 255), (355, 255), (355, 155), (255, 205)\}$.

Пусть граница пластины $\partial\Omega$ представлена несколькими участками (рис. 1):

$$\partial\Omega = \partial\Omega_l \cup \partial\Omega_r \cup \partial\Omega_t \cup \partial\Omega_b \cup \partial\Omega_{r1} \cup \partial\Omega_s \cup \partial\Omega_{r2};$$
$$\partial\Omega_{ex} = \partial\Omega_l \cup \partial\Omega_r \cup \partial\Omega_t \cup \partial\Omega_b \cup \partial\Omega_{r2}, \text{ – внешняя граница области } \Omega;$$
$$\partial\Omega_{in} = \partial\Omega_s \cup \partial\Omega_{r1}, \text{ – внутренняя граница области } \Omega.$$

Узловые точки, расположенные на границах $(x, y) \in \partial\Omega$ или их окрестности $(x, y) \in \Omega_{near}$ требуют особого внимания, поскольку в этих областях шаг сетки неравномерный, формулы для расчёта производных можно вывести используя материалы лекций по Вычислительной математике[4] или материалы БИГОР. Напомним, что внутренняя

область пластины обозначена просто Ω и рассматривается двумерная постановка задачи. Известно, что температура T в точке с координатами $(x, y) \in \Omega$ в момент времени t есть отображение $T\colon \mathbb{R} \times \mathbb{R}_+ \to \mathbb{R}$, которое вычисляется в результате решения дифференциального уравнения теплопроводности [6]:

$$\frac{\partial T}{\partial t} = \Delta T, \qquad (x, y) \in \Omega, \quad t \geq 0, \quad T = T(x, y, t), \tag{1}$$

где Δ – оператор Лапласа, т.е. $\Delta T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}$.

Начальное значение температуры (НУ) для всех вариантов имеет вид:

$$T\big|_{t=0} = 0, (x, y) \in \Omega. \tag{2}$$

При этом, граничные условия (ГУ) для сторон пластины $\partial\Omega_x$ будут иметь различный вид:

$$T\big|_{(x,y)\in\partial\Omega_x} = 100 \text{ – ГУ 1-о рода, задает источник нагрева;} \tag{3}$$

$$\nabla_{\bar{n}} T\big|_{(x,y)\in\partial\Omega_x} = 0 \text{ – ГУ 3-о рода, теплоизоляция,} \tag{4}$$

где $\nabla_{\bar{n}} T = \frac{\partial T}{\partial \bar{n}} = \nabla_{x,y} T \cdot \bar{n}$ – градиент температуры вдоль внешней нормали $\bar{n}$ к $\partial\Omega_x$;

$$\nabla_{\bar{n}} T\big|_{(x,y)\in\partial\Omega_x} = T \text{ – ГУ 3-о рода, для конвективного теплообмена на границе } \partial\Omega_x. \tag{5}$$

Физический смысл ГУ 1-о рода может означать, например, соприкосновение стороны поверхности $\partial\Omega_x$,если рассматирировать пластину как сечение тела, с некоторой средой высокой температуры (100 C°). Соотвествие $\partial\Omega_x$ конкретному ГУ задает параметр
$\gamma_3 \in \Gamma_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$, исходя из таблицы 2.

Требуется (базовая часть):

1 Определить форму пластины и ГУ в соответствии с $\mathrm{id}_{\Gamma,}[\gamma_i], \forall i \in [1:3]$ (пояснение на стр. 4).

Таблица 2: Варианты граничных условий в зависимости от γ_3

	«нагрев» (3)	«теплоизоляция» (4)	«конвекция» (5)
$\gamma_3 = 1$	$\partial\Omega_{ex}$	$\emptyset$	$\partial\Omega_{in}$
$\gamma_3 = 2$	$\partial\Omega_{in}$	$\emptyset$	$\partial\Omega_{ex}$
$\gamma_3 = 3$	$\partial\Omega_l$	$\partial\Omega_{in} \cup \partial\Omega_t \cup \partial\Omega_b$	$\partial\Omega_r$
$\gamma_3 = 4$	$\partial\Omega_r$	$\partial\Omega_{in} \cup \partial\Omega_t \cup \partial\Omega_b$	$\partial\Omega_l$
$\gamma_3 = 5$	$\partial\Omega_t$	$\partial\Omega_{in} \cup \partial\Omega_l \cup \partial\Omega_r$	$\partial\Omega_b$
$\gamma_3 = 6$	$\partial\Omega_b$	$\partial\Omega_{in} \cup \partial\Omega_l \cup \partial\Omega_r$	$\partial\Omega_t$
$\gamma_3 = 7$	$\partial\Omega_l \cup \partial\Omega_r$	$\partial\Omega_{in}$	$\partial\Omega_t \cup \partial\Omega_b$
$\gamma_3 = 8$	$\partial\Omega_l \cup \partial\Omega_r \cup \partial\Omega_{in}$	$\emptyset$	$\partial\Omega_t \cup \partial\Omega_b$
$\gamma_3 = 9$	$\partial\Omega_t \cup \partial\Omega_b$	$\partial\Omega_{in}$	$\partial\Omega_l \cup \partial\Omega_r$
$\gamma_3 = 10$	$\partial\Omega_t \cup \partial\Omega_b \cup \partial\Omega_{in}$	$\emptyset$	$\partial\Omega_l \cup \partial\Omega_r$
$\gamma_3 = 11$	$\partial\Omega_l \cup \partial\Omega_r$	$\emptyset$	$\partial\Omega_t \cup \partial\Omega_b \cup \partial\Omega_{in}$
$\gamma_3 = 12$	$\partial\Omega_t \cup \partial\Omega_b$	$\emptyset$	$\partial\Omega_l \cup \partial\Omega_r \cup \partial\Omega_{in}$
$\gamma_3 = 13$	$\partial\Omega_l \cup \partial\Omega_b$	$\partial\Omega_{in}$	$\partial\Omega_t \cup \partial\Omega_r$

2 Задать равномерный шаг дискретизации $h \in \{5, 10\}$[5] по координатам x и y. Построить расчётную сетку на множестве $\bar{\Omega}$ и рассчитать позиции узлов на границах.

3 Для каждого варианта шага h явным и неявным методом решить нестационарное уравнение теплопроводности (1) при заданных ГУ, определив значения температуры в узлах сетки в диапазоне времени $t \in (0; 100]$ сек, с шагом $h_t = 1$ сек.

4 Результаты необходимо сохранить в 4-х текстовых файлах[6], имя каждому следует задавать в формате согласно материалу инструкции по выполнению лабораторных работ [3][7]. Содержание каждого файла с результатами расчётов должно соответствовать следующему формату[8]:

```
t x y T
ti xj yk Tijk
...
```

5 Для неявного метода в отчёте должна быть приведена информация о СЛАУ: количество неизвестных (уравнений), число ненулевых элементов матрицы.

6 Сравнить полученные результаты вычислений с результатом моделирования аналогичной задачи в ANSYS.

Требуется (продвинутая часть):

7. Реализовать функцию кубической интеполярции значений температуры для произвольных точек пластины $(x, y) \in \mathbb{R}^2$ по известным значениям в узлах, в дискретные моменты времени;

8. Визуализировать результаты вычислений: функцию поля температуры $f(t_i, x, y)$ по всей пластине в виде цветовой диаграммы в требуемый момент времени (во время защиты)[9], для проверки корректности решения;

Цель выполнения лабораторной работы

Цель лабораторной работы: изучить МКР. Научиться применять МКР в задачах теплопроводности. Написать программу для решения поставленной задачи

Выполнение задачи

Форма пластины и граничные условия

Тип отверстия - круг с центром в точке (355, 155). Распределение

граничных условий по варианту номер 6.

Нестационарная задача

Процессы теплопередачи происходят в пространстве и времени. Поэтому исследование теплопроводности сводится к изучению пространственно-временного изменения температуры, т.е. к нахождению зависимости $T = T(x, y, z, t)$, где (x, y, z) – пространственные координаты в декартовой системе, t – время. Совокупность значений температуры во всех точках изучаемого пространства называется температурным полем. Различают стационарные и нестационарные температурные поля.

Нестационарным температурным полем называется такое поле, температура которого изменяется во времени. При решении нестационарной задачи теплопроводности в двумерном случае уравнение теплопроводности имеет вид:

$$\frac{\partial T}{\partial t} = a_1 \cdot \frac{\partial^2 T}{\partial x^2} + a_2 \cdot \frac{\partial^2 T}{\partial y^2}, \quad (6)$$

где t - время, a_1 и a_2 - коэффициент теплопроводности, T - температура, x и y - оси координатной системы.

Существует две различные схемы решения нестационарных тепловых задач: явная разностная схема и неявная. Явная разностная схема рассматривает один узел, находящийся на новом временном слое, и 5 узлов с предыдущего временного слоя. Таким образом, в явной схеме в каждом уравнении неизвестна только одна переменная, остальные же известны с предыдущего слоя или из граничных условий.

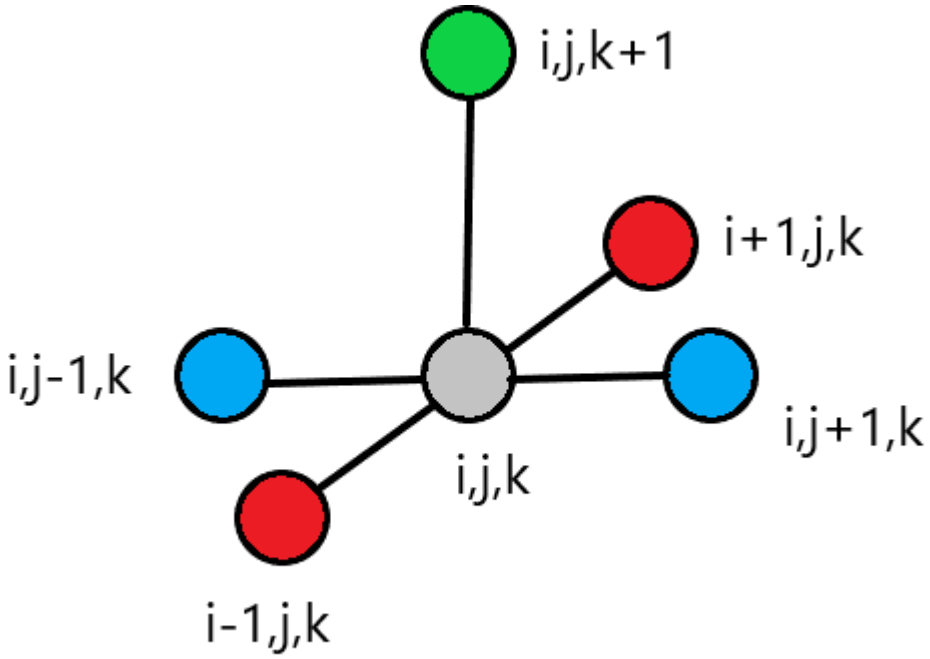


Рис. 3. Представление явной разностной схемы

Уравнение явной разностной схемы:

$$\frac{T^{k+1}_{i,j} - T^{k}_{i,j}}{\triangle t} = a^1 \cdot \frac{T^{k}_{i+1,j} - 2T^{k}_{i,j} + T^{k}_{i-1,j}}{\triangle x^2} + a_2 \cdot \frac{T^{k}_{i,j+1} - 2T^{k}_{i,j} + T^{k}_{i,j-1}}{\triangle y^2}, \quad (7)$$

Неявная разностная схема рассматривает один узел, находящийся на предыдущем временном слое, и 5 узлов с нового временного слоя. Таким образом, в неявной схеме много неизвестных в каждом уравнении и для их нахождения придется записать систему разностных уравнений для всех внутренних узлов сетки, и решить ее.

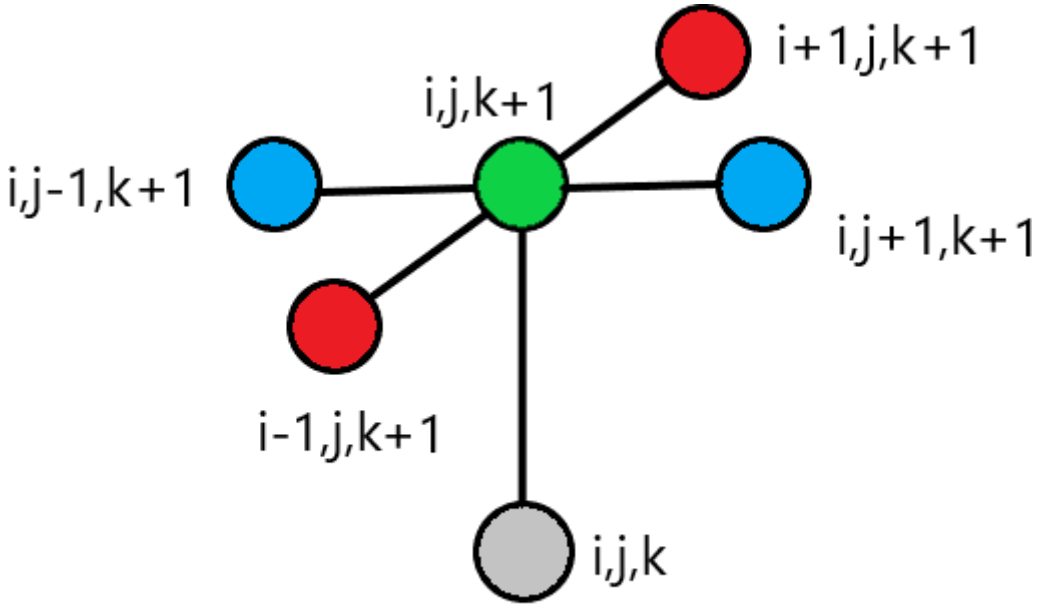


Рис. 4. Представление неявной разностной схемы

Уравнение неявной разностной схемы:

$$\frac{T^{k+1}_{i,j} - T^{k}_{i,j}}{\triangle t} = a^1 \cdot \frac{T^{k+1}_{i+1,j} - 2T^{k+1}_{i,j} + T^{k+1}_{i-1,j}}{\triangle x^2} + a_2 \cdot \frac{T^{k+1}_{i,j+1} - 2T^{k+1}_{i,j} + T^{k+1}_{i,j-1}}{\triangle y^2} \qquad (8)$$

В случае криволинейных границ для двумерных объектов практически невозможно нанести сетку на объект таким образом, чтобы узлы сетки попали строго на границу. В таком случае необходимо использовать метод дробления шага.

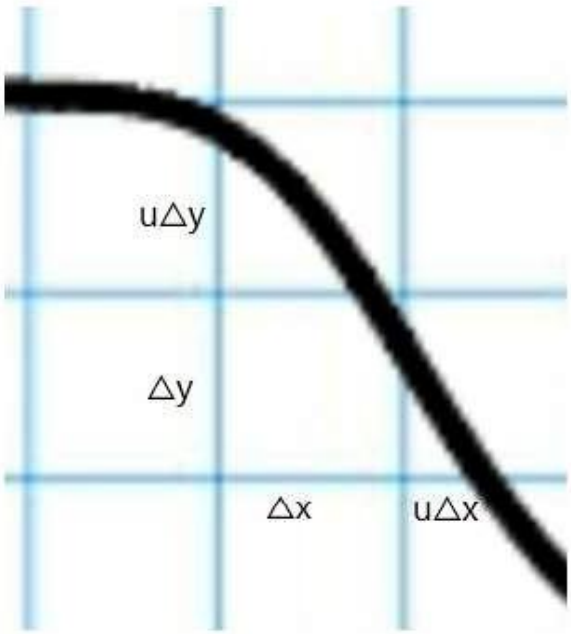


Рис. 5. Граница неправильной формы

Формула второй производной с дроблением шага:

$$\frac{\partial^2 T}{\partial x^2} = 2\frac{\mu V_C + V_A - (\mu+1)V_B}{(\mu^2+\mu)\triangle x^2} \quad (9)$$

Формула первой производной с дроблением шага:

$$\frac{\partial T}{\partial x} = \frac{\mu^2 V_C - V_A - (\mu^2-1)V_B}{(\mu^2-\mu)\triangle x} \quad (10)$$

Для узлов, не являющимися граничными, $\mu = 1$.

Реализация

Первым делом, проводим дискретизацию по времени по времени и пространству. Накладываем сетку с шагом на пластину. Начало координат совпадает с левым нижним углом пластины. В узлах сетки вычисляем значение температур. Исключение – граничные узлы, не попадающие на сетку. Они находятся на пересечении границ пластины с одной из "линий" сетки. Количество узлов в сетке по горизонтали : $n = \frac{W}{h} + 1$, по вертикали: $m = \frac{H}{h} + 1$.

Время дискретизируется путем разделения задачи на шаги, в каждом из которых производится вычисление температуры во всех узлах при фиксированном времени с выбранным шагом. Начальный момент времени $t = 0$.

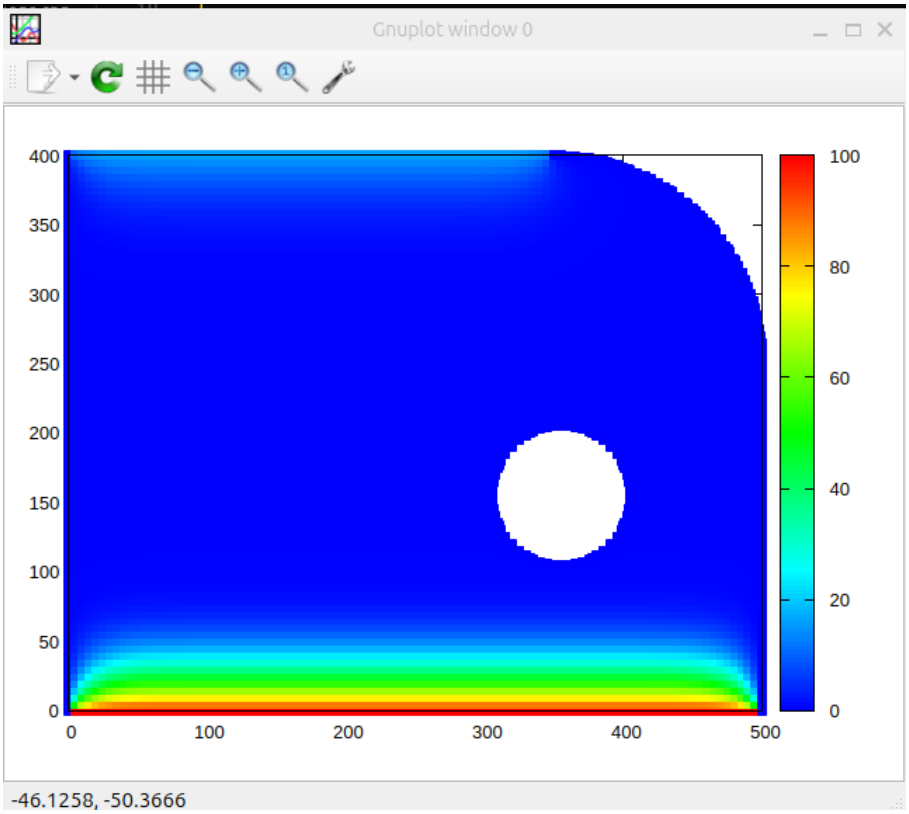
Результаты работы программы записываются в файлы согласно заданию. Визуализация данных выполнена с помощью модуля gnuplot.

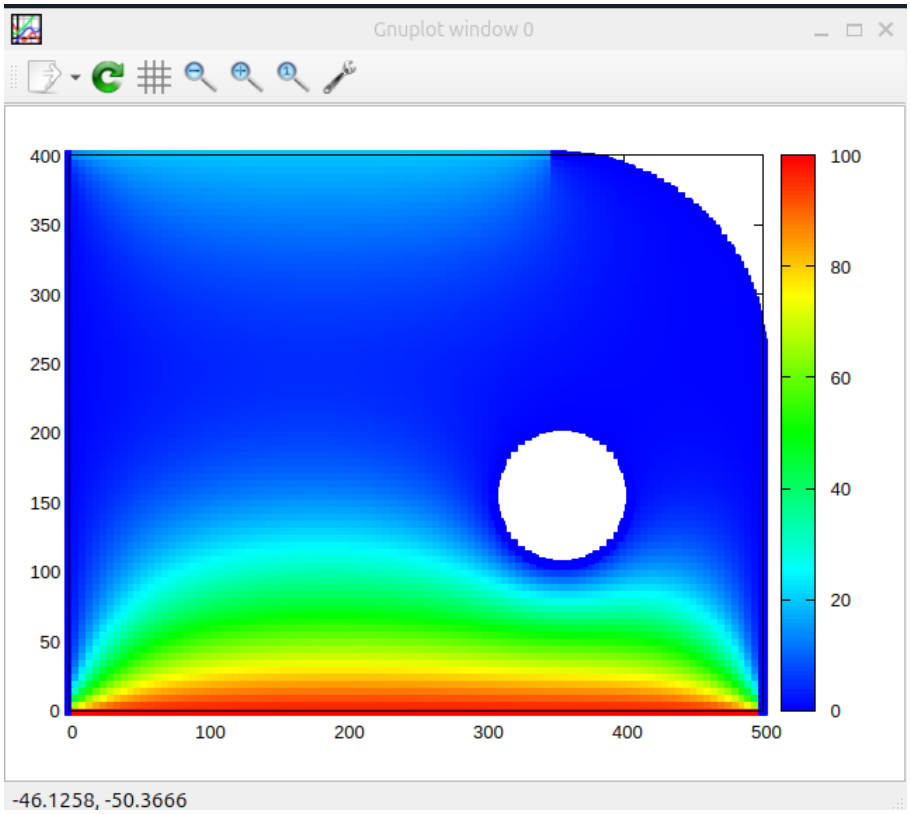
1.явная h=5

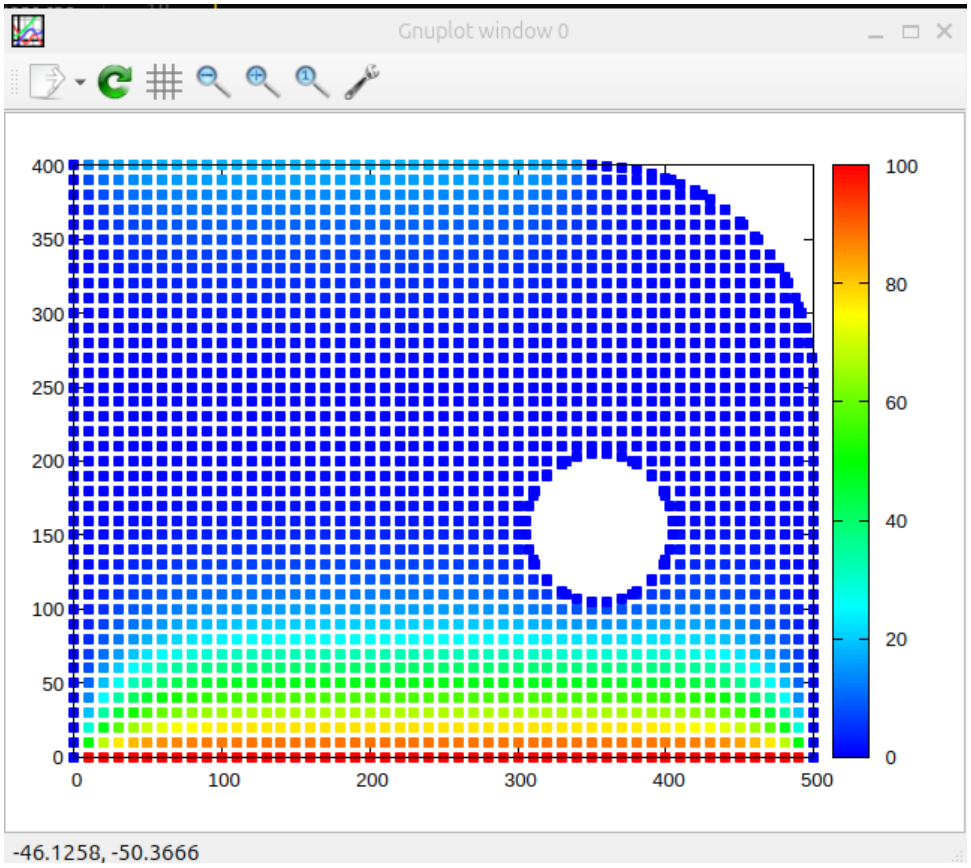
2. неявная h=5

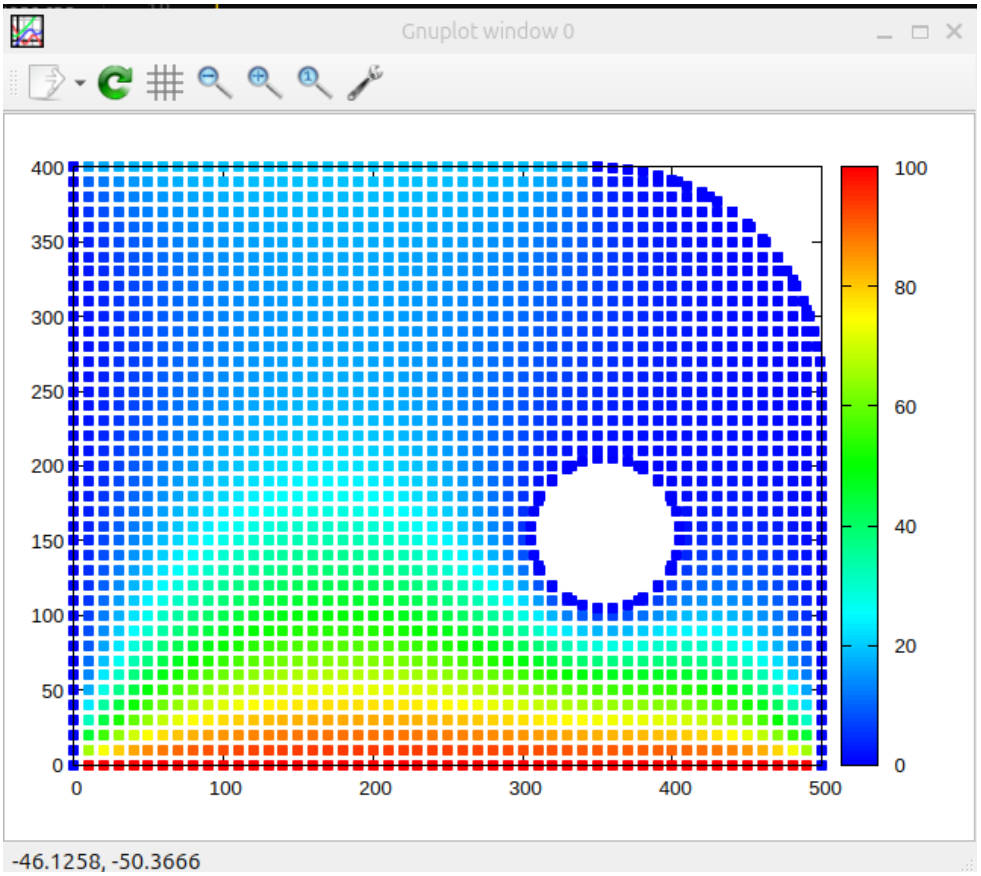
3.явный h=10

4.неявный h=10


Gnuplot window 0
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
100
80
60
40
20
0
-46.1258, -50.3666


Gnuplot window 0
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
100
80
60
40
20
0
-46.1258, -50.3666


Gnuplot window 0
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
100
80
60
40
20
0
-46.1258, -50.3666


Gnuplot window 0
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
100
80
60
40
20
0
-46.1258, -50.3666

Моделирование в ANSYS

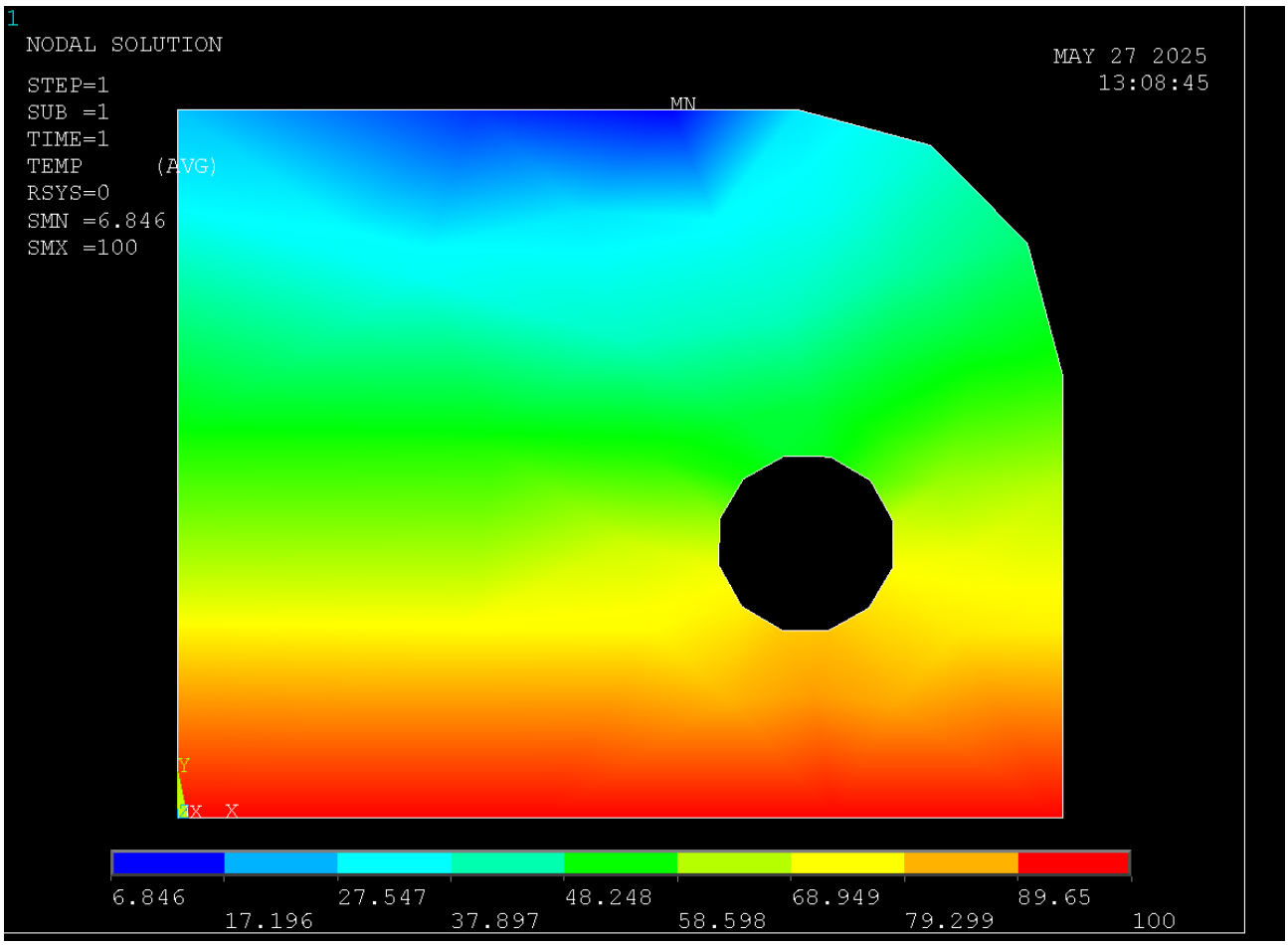


Рисунок 10. Результат решения уравнения теплопроводности (1) для пластины, изображённой на Рисунке 1 в ANSYS

Листинг программы

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include "mimapr_lab.h"
using namespace std;

size_t Form::counter_ = 0;
double Form::Function(double, double) {
    return 0;
}
pair<double, double> Form::Deriative(double, double) {
    return {0, 0};
}
pair<double, double> Form::size()
{
    return {0, 0};
}
bool Form::Inhere(double, double) {
    return false;
}
pair<double, double> Form::missX(double)
{
    return {0, 0};
}
pair<double, double> Form::missY(double)
{
    return {0, 0};
}
Form::Form() {
    id_ = counter_++;
    excluded_ = false;
}
size_t Form::Get_ID() const {
    return id_;
}
bool Form::Excluded() const {
    return excluded_;
}
int Form::GetB() {return _boundtype;}
bool Form::operator==(size_t id) const {
    return id_ == id;
}


Rectangle::Rectangle(double a, double b, double h_x, double h_y, bool excluded, int
btype) : a_(a), b_(b), h_x_(h_x), h_y_(h_y) {
    excluded_ = excluded;
    _boundtype = btype;
}
pair<double, double> Rectangle::missX(double y)
{
    return {0.5 / h_x_ + a_, -0.5 / h_x_ + a_};
}
```

```
pair<double, double> Rectangle::missY(double x)
{
    return {0.5 / h_y_ + b_, -0.5 / h_y_ + b_};
}

pair<double, double> Rectangle::size()
{
    return {1 / h_x_, 1 / h_y_};
}
double Rectangle::Function(double x, double y) {
    return max(h_x_ * abs(x - a_), h_y_ * abs(y - b_));
}
pair<double, double> Rectangle::Deriative(double x, double y) {
    return {(h_x_ / 2) * ((x - a_) / abs(x - a_)), (h_y_ / 2) * ((y - b_) / abs(y -
b_))};
}
bool Rectangle::Inhere(double x, double y) {
    return Function(x, y) <= 0.5;
}



Arc::Arc(double a, double b, double h_x, double h_y, bool excluded, int btype):
a_(a), b_(b), h_x_(h_x), h_y_(h_y) {
    excluded_ = excluded;
    _boundtype = btype;
}
pair<double, double> Arc::missY(double x)
{
    return {sqrt(1 - pow((h_x_ * (x - a_)), 2))/h_y_ + b_, sqrt(1 - pow((h_x_ * (x
- a_)), 2))/h_y_ + b_};
}
pair<double, double> Arc::missX(double y)
{
    return {sqrt(1 - pow((h_y_ * (y - b_)), 2))/h_x_ + a_, sqrt(1 - pow((h_y_ * (y
- b_)), 2))/h_x_ + a_};
}
double Arc::Function(double x, double y) {
    if (x >= a_ && y >= b_) {
        return pow(h_x_ * (x - a_), 2) + pow(h_y_ * (y - b_), 2);
    }
    return -1.0;
}
pair<double, double> Arc::Deriative(double x, double y) {
    if (x >= a_ && y >= b_) {
        return {2 * h_x_ * (x - a_), 2 * h_y_ * (y - b_)};
    }
    if (x < a_) {
        cout << "x < a\n";
    }
    if (y < b_) {
        cout << "y < b\n";
    }
    return {-1.0, -1.0};
```

```
}
pair<double, double> Arc::size()
{
    return {1 / h_x_, 1 / h_y_};
}
bool Arc::Inhere(double x, double y) {
    return Function(x, y) >= 1;
}


double Node::X() const {return _x;}
double Node::Y() const {return _y;}
double Node::T() const
{
    if (!_btype)
        return _t;
    if (_btype == 1)
        return 100.;
    if (_btype == 2)
    {
        if (!_left)
            if (_right)
                return _right->T();
        if (!_right)
            if (_left)
                return _left->T();
        if (!_above)
            if (_below)
                return _below->T();
        if (!_below)
            if (_above)
                return _above->T();
        if (_right && _left)
        {
            if (_right->IsBound())
                return _left->T();
            return _right->T();
        }
        if (_above && _below)
        {
            if (_above->IsBound())
                return _below->T();
            return _above->T();
        }
    }
    if (_btype == 3)
    {
        if (!_left && _right)        return _right->T();
        if (!_right && _left)       return _left->T();
        if (!_below && _above)      return _above->T();
        if (!_above && _below)      return _below->T();

        if (_right && _left) {
            // выбираем из двух соседа тот, что не граничный
```

```
            Node* n = _right->IsBound() ? _left : _right;
            return n->T();
        }
        if (_above && _below) {
            Node* n = _above->IsBound() ? _below : _above;
            return n->T();
        }
    }

    // на всякий случай
    return 0.0;
}
double Node::Dist(const Node* to) const
{
    return sqrt(pow(X() - to->X(), 2) + pow(Y() - to->Y(), 2));
}
Node*& Node::l() {return _left;}
Node*& Node::r() {return _right;}
Node*& Node::u() {return _above;}
Node*& Node::d() {return _below;}
void Node::LinkX(Node* l, Node* r)
{
    _left = l;
    _right = r;
}
void Node::LinkY(Node* d, Node* u)
{
    _below = d;
    _above = u;
}
void Node::SetT(double t)
{
    _t = t;
}
bool Node::IsBound() {return _btype;}
void Node::SetB(int type) {_btype = type;}


Object::Object(): _w(0), _h(0) {}
double Object::Inhere(double x, double y)
{
    for (auto form: forms_)
    {
        if (form->Excluded())
        {
            if (form->Inhere(x, y))
            {
                return false;
            }
        }
        else
        {
            if (form->Inhere(x, y))
            {
```

```
                return true;
            }
        }
    }
    return false;
}
void Object::Updsize()
{
    for (auto form : forms_)
    {
        _w = max(_w, form->size().first);
        _h = max(_h, form->size().second);
    }
}
bool Object::Add_Form(const string &name, map<string, double> &args, bool excluded,
int btype) {
    if (name == "Rectangle") {
        forms_.push_back(new Rectangle(args["a"], args["b"], args["h_x"],
args["h_y"], excluded, btype));
        Updsize();
        return true;
    } else if (name == "Arc") {
        forms_.push_back(new Arc(args["a"], args["b"], args["h_x"], args["h_y"],
excluded, btype));
        Updsize();
        return true;
    } else if (name == "Circle") {
        forms_.push_back(new Circle(
            args["a"], args["b"], args["R"],
            excluded, btype
        ));
        Updsize();
        return true;

    }
    return false;
}
pair<double, double> Object::Fillx(double x, double y)
{
    for (auto form: forms_)
    {
        if (form->Inhere(x, y))
        {
            return form->missX(y);
        }
    }
    return {0, 0};
}
pair<double, double> Object::Filly(double x, double y)
{
    for (auto form: forms_)
    {
        if (form->Inhere(x, y))
        {
```

```
            return form->missY(x);
        }
    }
    return {0, 0};
}
double Object::Width() const
{
    return _w;
}
double Object::Height() const
{
    return _h;
}
bool Object::Delete_Form(size_t id) {
    return false;
}
vector<size_t> Object::Get_IDs() {
    vector<size_t> ids;
    ids.reserve(forms_.size());
    for (auto form: forms_) {
        ids.push_back(form->Get_ID());
    }
    return ids;
}
Form* Object::Who(double x, double y)
{
    for (auto form: forms_)
    {
        if (form->Inhere(x, y))
        {
            return form;
        }

    }
    return forms_.back();
}
Object::~Object()
{
    for (auto form : forms_)
        delete form;
}


Mesh::Mesh(Object& obj, double step): _obj(obj), _step(step)
{
    for (double y = 0; y <= _obj.Height(); y += _step)
    {
        _mesh.push_back(vector<Node*>());
        for (double x = 0; x <= _obj.Width(); x += _step)
        {
            _mesh.back().push_back(new Node(x, y));
        }
    }
    LinkX();
```

```
    LinkY();
    Adapt();
}
void Mesh::LinkX()
{
    for (int i = 0; i < _mesh.size(); i++)
    {
        _mesh[i][0]->LinkX(nullptr, _mesh[i][1]);
        for (int j = 1; j < _mesh[i].size() - 1; j++)
            _mesh[i][j]->LinkX(_mesh[i][j - 1], _mesh[i][j + 1]);
        _mesh[i].back()->LinkX(_mesh[i][_mesh[i].size() - 2], nullptr);
    }
    for (int i = 0; i < _mesh.size(); i++)
        _hlines.push_back(_mesh[i][0]);
}
void Mesh::LinkY()
{
    for (int j = 0; j < _mesh[0].size(); j++)
    {
        _mesh[0][j]->LinkY(nullptr, _mesh[1][j]);
        for (int i = 1; i < _mesh.size() - 1; i++)
            _mesh[i][j]->LinkY(_mesh[i - 1][j], _mesh[i + 1][j]);
        _mesh[_mesh.size() - 1][j]->LinkY(_mesh[_mesh.size() - 2][j], nullptr);
    }
    for (int i = 0; i < _mesh[0].size(); i++)
        _vlines.push_back(_mesh[0][i]);
}
void Mesh::Adapt()
{
    for (int i = 0; i < _mesh.size(); i++)
    {
        int s = _mesh[i].size();
        for (int j = 0; j < s; j++)
        {
            if (!_obj.Inhere(_mesh[i][j]->X(), _mesh[i][j]->Y()))
            {
                Delnode(i, j);
                j--;
                s--;
            }
        }
    }
}
void Mesh::ShowLinks()
{
    for (auto line : _mesh)
    {
        for (auto node : line)
        {
            if (node->d())
                cout << "|  ";
        }
        cout << '\n';
        for (auto node : line)
```

```
        {
            if (node->l())
            {
                cout << '-';
            }
            cout << 'N';
            if (node->r())
            {
                cout << '-';
            }
            else
            {
                cout << '\n';
            }
        }
        for (auto node : line)
        {
            if (node->u())
                cout << "|";
            cout << "  ";
        }
        cout << '\n';
    }
}
void Mesh::Delnode(int i, int j)
{
    Node* node = _mesh[i][j];
    double bndX1 = _obj.Fillx(node->X(), node->Y()).first;
    double bndX2 = _obj.Fillx(node->X(), node->Y()).second;
    double bndY1 = _obj.Filly(node->X(), node->Y()).first;
    double bndY2 = _obj.Filly(node->X(), node->Y()).second;
    int btype = _obj.Who(node->X(), node->Y())->GetB();
    if (node->l())
    {
        if (node->l()->X() != bndX2 && node->l()->X() != bndX1)
        {
            if (bndX1 != bndX2)
            {
                Node* left = new Node(bndX2, node->Y(), btype);
                Node* right = new Node(bndX1, node->Y(), btype);
                node->l()->r() = left;
                if (node->r())
                    node->r()->l() = right;
                left->LinkX(node->l(), right);
                right->LinkX(left, node->r());
                node->l() = right;
                _mesh[i].push_back(left);
                _mesh[i].push_back(right);
            }
            else
            {
                Node* left = new Node(bndX2, node->Y(), btype);
                node->l()->r() = left;
                if (node->r())
```

```
                    node->r()->l() = left;
                left->LinkX(node->l(), node->r());
                node->l() = left;
                _mesh[i].push_back(left);
            }
        }
        else
            node->l()->r() = node->r();
    }
    if (node->r())
    {
        node->r()->l() = node->l();
    }
    if (node->d())
    {
        if (node->d()->Y() != bndY2 && node->d()->Y() != bndY1)
        {
            if (bndY2 != bndY1)
            {
                Node* down = new Node(node->X(), bndY2, btype);
                Node* up = new Node(node->X(), bndY1, btype);
                node->d()->u() = down;
                if (node->u())
                    node->u()->d() = up;
                down->LinkY(node->d(), up);
                up->LinkY(down, node->u());
                node->d() = up;
                _mesh[i].push_back(down);
                _mesh[i].push_back(up);
            }
            else
            {
                Node* down = new Node(node->X(), bndY2, btype);
                node->d()->u() = down;
                if (node->u())
                    node->u()->d() = down;
                down->LinkY(node->d(), node->u());
                node->d() = down;
                _mesh[i].push_back(down);
            }
        }
        else
            node->d()->u() = node->u();
    }
    if (node->u())
    {
        node->u()->d() = node->d();
    }
    _mesh[i].erase(_mesh[i].begin() + j);
    delete node;
}
vector<vector<Node*>>& Mesh::Nodes() {return _mesh;}
vector<Node*>& Mesh::LineX()  {return _hlines;}
vector<Node*>& Mesh::LineY()  {return _vlines;}
```

```
Mesh::~Mesh()
{
    for (auto line : _mesh)
        for (auto node : line)
            delete node;
}


void System::DefineBounds(int l, int t, int r, int b)
{
    Node* cur = _mesh.LineX().front();
    while (cur)
    {
        cur->SetB(b);
        cur = cur->r();
    }
    cur = _mesh.LineX().back();
    while (cur)
    {
        cur->SetB(t);
        cur = cur->r();
    }
    cur = _mesh.LineY().front();
    while (cur)
    {
        cur->SetB(l);
        cur = cur->u();
    }
    cur = _mesh.LineY().back();
    while (cur->u())
    {
        cur->SetB(r);
        cur = cur->u();
    }
}
vector<vector<Node*>>& System::Nodes() {return _mesh.Nodes();}
vector<Node*>& System::LineX() {return _mesh.LineX();}
vector<Node*>& System::LineY()  {return _mesh.LineY();}
double System::step() const {return _step;}
double System::a1() const {return _a1;}
double System::a2() const {return _a2;}


void Solver::SolveImplicit(System& sys, double tstop) const
{
    ofstream ImplicitOut(_name_2);
    for (double t = 0.0; t < tstop; t += _dt)
    {
        for (int i = 1; i < sys.LineX().size() - 1; i++)
        {
            vector<Node*> temperature;
            Node* cur = sys.LineX()[i];
            while (cur)
            {
```

```
                if (cur->r() &&  cur->r()->X() - cur->X() > sys.step())
                {
                    temperature.push_back(cur);
                    SolveLine(sys, temperature);
                    temperature.clear();
                    cur = cur->r();
                }
                else
                {
                    temperature.push_back(cur);
                    cur = cur->r();
                }
            }
            SolveLine(sys, temperature);
        }
        for (int i = 1; i < sys.LineY().size() - 1; i++)
        {
            vector<Node*> temperature;
            Node* cur = sys.LineY()[i];
            while (cur)
            {
                if (cur->u() &&  cur->u()->Y() - cur->Y() > sys.step())
                {
                    temperature.push_back(cur);
                    SolveLine(sys, temperature);
                    temperature.clear();
                    cur = cur->u();
                }
                else
                {
                    temperature.push_back(cur);
                    cur = cur->u();
                }
            }
            SolveLine(sys, temperature);
        }
        for (auto line : sys.Nodes())
        {
            for (auto node : line)
                ImplicitOut << t << " " << node->X() << " " << node->Y() << " " <<
node->T() << '\n';
        }
        ImplicitOut << "\n\n";
    }
}
void Solver::SolveExplicit(System& sys, double tstop) const
{
    ofstream ExplicitOut(_name_1);
    for (double t = 0.; t < tstop; t += _dt)
    {
        for (auto line : sys.Nodes())
            for (auto node : line)
            {
                if (!node->IsBound())
```

```
                {
                    double m1 = 1.;
                    double m2 = 1.;
                    double tx = 2 * (m1 * node->r()->T() - (m1 + 1) * node->T() +
node->l()->T()) / (m1 * (m1 + 1) * pow(sys.step(), 2));
                    double ty = 2 * (m2 * node->u()->T() - (m2 + 1) * node->T() +
node->d()->T()) / (m2 * (m2 + 1) *pow(sys.step(), 2));
                    double t = _dt * sys.a1() * (tx + ty) + node->T();
                    node->SetT(t);
                }
            }
        for (auto line : sys.Nodes())
        {
            for (auto node : line)
                ExplicitOut << t << " " << node->X() << " " << node->Y() << " " <<
node->T() << '\n';
        }
        ExplicitOut << "\n\n";
    }
}
void Solver::SolveLine(System& sys, vector<Node*>& n) const
{
    int size = n.size() - 2;
    double mu1 =  n.front()->Dist(n[1]) / sys.step();
    double mu2 = n.back()->Dist(n[n.size() - 2]) / sys.step();
    double val2 = -(2 * sys.a1()) / (pow(sys.step(), 2)) - 1 / _dt;
    double val1 = sys.a1() / (pow(sys.step(), 2));
    vector<vector<double>> next(size);
    vector<double> right(size);
    for (int i = 0; i < next.size(); i++)
        next[i].resize(3, 0.0);
    next[0][0] = -(2 * sys.a1()) / (mu1 * pow(sys.step(), 2)) - 1 / _dt;//val2;
    next[0][1] = (2 * sys.a1()) / ((mu1 + 1) * pow(sys.step(), 2));// val1;
    next.back()[1] = (2 * sys.a1()) / ((mu2 + 1) * pow(sys.step(), 2));// val1;
    next.back()[2] = -(2 * sys.a1()) / (mu2 * pow(sys.step(), 2)) - 1 / _dt;//val2;
    for (int i = 1; i < size - 1; i++)
    {
        next[i][0] = val1;
        next[i][1] = val2;
        next[i][2] = val1;
    }
    for (int i = 0; i < right.size(); i++)
        right[i] = -n[i+1]->T() / _dt;
    right.front() += -(2 * sys.a1() * n.front()->T()) / (mu1 * (mu1 + 1) *
pow(sys.step(), 2));
    right.back() += -(2 * sys.a1() * n.back()->T()) / (mu2 * (mu2 + 1) *
pow(sys.step(), 2));
    vector<double> tmps = ThomasMethod(next, right);
    for (int i = 0; i < tmps.size(); i++)
        n[i + 1]->SetT(tmps[i]);
}

// метод прогонки для 3-диагон. матриц
```

```
vector<double> Solver::ThomasMethod(vector<vector<double>>& A, vector<double>& b)
const
{
    int row = b.size() - 1;
    vector<double>alph(row);
    vector<double>bet(row);
    vector<double>x(b.size());
    alph[0] = - A[0][1] / A[0][0];
    bet[0] = b[0] / A[0][0];
    for (int i = 1; i < row; i++)
    {
        double a = A[i][0];
        double b1 = A[i][1];
        double c = A[i][2];
        alph[i] = -c / (a * alph[i - 1] + b1);
        bet[i] = (b[i] - a * bet[i - 1]) / (a * alph[i - 1] + b1);
    }
    x.back() = (b.back() - A.back()[1]*bet.back()) / (A.back()[2] +
A.back()[1]*alph.back());
    for (int i = row - 1; i > -1; i--)
        x[i] = alph[i] * x[i+1] + bet[i];
    return x;
}

// визуализация
void visualize(ofstream &file, string filename, int time_end, int t) {
    file << "unset key\n";
    file << "set size ratio 0.8\n";
    file << "set cbrange [0:100]\n";
    file << "set style line 1 lw 2\n";
    file << "plot '" << filename << "' using 2:3:4 index " << t << " w points
linestyle 5 palette pointsize 1\n";
    file << "pause mouse\n";
}

int main()
{
    // Параметры пластины
    double w = 500.;
    double h = 400.;
    double S = 100.;
    double R2 = 150.;
    double Xrec = 155.;
    double Yrec = 155.;
    // Граничные условия: r, l, t, b определяют сторону пластины. // (1) => T =
100, нагрев; (2) => grad(T)=T, конвекция; (3) => grad(T)=0, теплоизоляция
    int rl = 3;
    int rt = 2;
    int rr = 3;
    int rb = 1;
    int rec = 3;
    int r2 = 3;

    double h5 = 5.;
```

```
    double h10 = 10.;
    double time_step = 1.;
    double time_end = 101.;
    double C = 10.;

    map<string, double> base{{"a", w / 2}, {"b", h / 2}, {"h_x", 1/ w}, {"h_y", 1/
h}};
    map<string, double> prores{{"a", Xrec}, {"b", Yrec}, {"h_x", 1/ S}, {"h_y", 1/
S}};
    map<string, double> arc{{"a", w - R2}, {"b", h - R2}, {"h_x", 1 / R2}, {"h_y",
1 / R2}};
    map<string,double> circleArgs{
    {"a", 355.0},  // x-координата центра
    {"b", 155.0},  // y-координата центра
    {"R", 50.0}    // радиус
    };

    Object obj;
    obj.Add_Form("Arc", arc, true, r2);
    //obj.Add_Form("Rectangle", prores, true, rec);
    obj.Add_Form("Circle", circleArgs, true, rec);
    obj.Add_Form("Rectangle", base, false, 1);

    System explicit_5(obj, h5, C);
    explicit_5.DefineBounds(rl, rt, rr, rb);
    System implicit_5(obj, h5, C);
    implicit_5.DefineBounds(rl, rt, rr, rb);
    Solver slv_5("res1.txt", "res2.txt", time_step);
    slv_5.SolveExplicit(explicit_5, time_end);
    slv_5.SolveImplicit(explicit_5, time_end);
    ofstream script("explicit_5.plt");
    script.close();
    script.open("implicit_5.plt");
    visualize(script, "res2.txt", 100, 99);
    script.close();

    System explicit_10(obj, h10, C);
    explicit_10.DefineBounds(rl, rt, rr, rb);
    System implicit_10(obj, h10, C);
    implicit_10.DefineBounds(rl, rt, rr, rb);
    Solver slv_10("res3.txt", "res4.txt", time_step);
    slv_10.SolveExplicit(explicit_10, time_end);
    slv_10.SolveImplicit(implicit_10, time_end);
    script.open("explicit_10.plt");
    script.close();
    script.open("implicit_10.plt");
    script.close();

    return 0;
}
```

Заключение

В ходе выполнения лабораторной работы был изучен МКР.

Построена модель в ANSYS, заданы необходимые граничные условия для задачи теплопроводности, реализован программный аналог решения задачи на языке программирования C++.