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 О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент	Никифорова Ирина Андреевна
Группа	РК6-61б
Тип задания	лабораторная работа
Тема лабораторной работы	Метод конечных разностей. Неявная разностная схема.

Студент _____ **Никифорова И. А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Трудоношин В. А.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2019 г.

Оглавление

Задание на лабораторную работу

С помощью неявной разностной схемы решить нестационарное уравнение теплопроводности для прямоугольной пластины размером 5*5см. Начальное значение температуры пластины - 20 градусов.

Граничные условия следующие: левая граница теплоизолирована, на нижней границе поддерживается 20, на остальной части границы температура 800 градусов.

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 10 сек, сравнение результатов расчета с результатами, полученными с помощью пакета ANSYS.

Описание и текст программы

В соответствии с заданием была разработана программа, рассчитывающая значения температуры в каждой точке выбранной сетки на пластине, размером 5 см на 5 см.

Для удобства написания и поддержки, код программы был разделен на несколько файлов, а сборка производилась с помощью программы make. Конфигурационный файл для сборки программы представлен в листинге 1.

Листинг 1. Makefile

```
CC=c++
CFLAGS=-c

all: main.out

main.out: matrix.o main.cpp
	$(CC) -o main.out matrix.o main.cpp

matrix.o: matrix.hpp matrix.cpp
	$(CC) matrix.hpp matrix.cpp $(CFLAGS)

clean:
	rm matrix.hpp.gch matrix.o
```

Файлы, которые нужны для компиляции программы и обработки ее вывода: main.cpp, lab1.hpp, matrix.hpp, matrix.cpp, plot.gnu, Makefile.

Необходимые для вычислений константы и макросы для определения граничных условий были помещены в файл lab1.hpp. Текст этого файла представлен в листинге 2. Здесь константы I_MAX, J_MAX и K_MAX указывают максимальные номера (начиная с нуля) пространственно-временной сетки. D_X, D_Y, D_K - шаги сетки в сантиметрах или секундах. Также здесь указана константа начальной температуры T_0 и макросы для определения граничных условий T_LEFT, T_BOTTOM, T_UP, T_RIGHT. Макросы используются как функции только на начальной итерации расчета. Далее макросы граничных условий первого рода выступают скорее в роли констант для СЛАУ, а макрос T_LEFT (для граничного условия второго рода) не используется.

Листинг 2. lab1.hpp

```
#ifndef LAB1_HPP
#define LAB1_HPP

// 10x10
// максимальные номера узлов сетки, минимум - 0 узел
#define I_MAX 10
#define J_MAX 10
#define K_MAX 10

// шаги сетки в см || с
#define D_X 0.5
#define D_Y 0.5
#define D_T 1

// начальные условия
#define T_0 20

// граничные условия
#define T_LEFT(neighbour) (neighbour)
#define T_BOTTOM() (20.0)
#define T_RIGHT() (800.0)
#define T_UP() (800.0)

#endif  // LAB1_HPP
```

Большинство вычислений в программе происходит для подсчета вектора неизвестных температур в узлах сетки с помощью метода Гаусса. Для решения этой задачи был написан специальный шаблонный класс матрицы. Его определение находится в файле matrix.hpp, текст которого представлен в листинге 3.

Листинг 3. matrix.hpp

```
#ifndef MATRIX_HPP
#define MATRIX_HPP

#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

template <typename T>
class Matrix {
private:
    T** _arr;
    int _rows;
    int _cols;
public:
    // методы, относящиеся к методу Гаусса
```

```
    void make_diag_one_Gauss(int i_from, Matrix<double>* fr);
    void substract_rows_Gauss(int i_from, int i_to, T coeff,
                              Matrix<double>* fr);
    void forward_Gauss(Matrix<double>* fr);
    void backward_Gauss(Matrix<double>* fr);
    void Gauss(Matrix<double>* fr);

    // основные используемые методы для матриц
    Matrix(const int rows = 1, const int cols = 1);
    const int get_rows() const;
    const int get_cols() const;
    const T get(int i, int j) const;
    void set(int i, int j, T val);
    const bool check_i_j(int i, int j) const;
    void swap_rows(int i1, int i2);

    void print(Matrix<double>* fr, string message = "") const;
    ~Matrix();
};

#endif // MATRIX_HPP
```

Были определены стандартные методы работы с матрицами, такие как: создание (конструктор), получение количества строк (get_rows), столбцов (get_cols), элемента (get), установка (set), проверка индексов на выход за границы (check_i_j), смена строк местами (swap_rows) и печать (print).

В дополнение к стандартным методам были написаны функции для работы по методу Гаусса: установка диагонального элемента в значение один (make_diag_one_Gauss), вычитание строк с домножением на переданный коэффициент (substract_rows_Gauss), прямой (forward_Gauss) и обратный (backward_Gauss) методы Гаусса, а также общая обертка над ними (Gauss).

Большинство из перечисленных функций принимает в качестве аргумента вектор свободных членов fr, чтобы в процессе работы с ним также производить необходимые вычисления.

Определение функций из файлы matrix.hpp находится в файле matrix.cpp - листинг 4. Все необходимые для понимания комментарии приводятся в коде программы.

Листинг 4. matrix.cpp

```
#include "matrix.hpp"
```

```
// конструктор матрицы по строкам и столбцам
template <typename T>
Matrix<T>::Matrix(const int rows, const int cols) {
    _rows = rows;
    _cols = cols;
    _arr = new T*[rows];
    for (int i = 0; i < rows; i++) {
        _arr[i] = new T[cols];
    }
}

// возвращает количество строк в матрице
template <typename T>
const int Matrix<T>::get_rows() const {
    return _rows;
}

// возвращает количество столбцов в матрице
template <typename T>
const int Matrix<T>::get_cols() const {
    return _cols;
}

// возвращает элемент с индексами i и j
template <typename T>
const T Matrix<T>::get(int i, int j) const {
    if (!check_i_j(i, j)) {
        return 0;
    }

    return _arr[i][j];
}

// устанавливает элемент с индексами i и j в значение val
template <typename T>
void Matrix<T>::set(int i, int j, T val)  {
    if (!check_i_j(i, j)) {
        return;
    }

    _arr[i][j] = val;
}

// проверяет на корректность номера элементов
template <typename T>
const bool Matrix<T>::check_i_j(int i, int j) const {
    if (i >= _rows) {
        cout << "too high row:" << i << endl;
        return false;
    } else if (j >= _cols) {
        cout << "too high col:" << j << endl;
        return false;
    } else if (i < 0) {
        cout << "too low row:" << i << endl;
        return false;
    } else if (j < 0) {
        cout << "too low col:" << j << endl;
```

```
            return false;
        }

        return true;
    }

    // делит главную строку на диагональный элемент,
    // чтобы там была единица
    template <typename T>
    void Matrix<T>::make_diag_one_Gauss(int i_from, Matrix<double>* fr) {
        T coeff = _arr[i_from][i_from];
        for (int j = 0; j < _cols; j++) {
            _arr[i_from][j] = _arr[i_from][j] / coeff;
        }
        fr->set(i_from, 0, (fr->get(i_from, 0) / coeff));

    }

    // вычитает из обрабатываемой строки главную,
    // умноженную на рассчитанный коэффициент
    template <typename T>
    void Matrix<T>::substract_rows_Gauss(int i_from, int i_to, T coeff,
      Matrix<double>* fr) {
        for (int j = 0; j < _cols; j++) {
            _arr[i_to][j] -= _arr[i_from][j] * coeff;
        }
        fr->set(i_to, 0, fr->get(i_to, 0) - (fr->get(i_from, 0) * coeff));
    }

    // выполняет прямой ход метода Гаусса
    template <typename T>
    void Matrix<T>::forward_Gauss(Matrix<double>* fr) {
        //cout << "Forward: " << endl;
        int i;
        for (i = 0; i < get_rows() - 1; i++) {
            for (int k = i + 1; k < get_rows(); k++) {
                make_diag_one_Gauss(i, fr);
                T coeff = _arr[k][i];
                substract_rows_Gauss(i, k, coeff, fr);
            //    print(fr);
            }
        }
        make_diag_one_Gauss(i, fr);
        //print(fr);
    }

    // выполняет обратный ход метода Гаусса
    template <typename T>
    void Matrix<T>::backward_Gauss(Matrix<double>* fr) {
        //cout << "Backward: " << endl;
        int i;
        for (i = get_rows() - 1; i > 0 ; i--) {
            for (int k = i - 1; k >= 0; k--) {
                T coeff = _arr[k][i];
                substract_rows_Gauss(i, k, coeff, fr);
                //print(fr);
            }
```

```
    }
    //print(fr);
}

// выполняет приведение матрицы к диагональному виду методом Гаусса
template <typename T>
void Matrix<T>::Gauss(Matrix<double>* fr) {
    forward_Gauss(fr);
    backward_Gauss(fr);
}

// печатает переданное сообщение, саму матрицу
// и матрицу свободных членов
template <typename T>
void Matrix<T>::print(Matrix<double>* fr, string message) const {
    cout.precision(3);
    cout << message << endl;
    for (int i = 0; i < _rows; i++) {
        cout << "| ";
        for (int j = 0; j < _cols; j++) {
            cout << setw(5) << _arr[i][j] << " ";
        }
        cout << "|";
        if (fr != NULL) {
            cout << "  |" << setw(5) << fr->get(i, 0) << "|" << endl;
        } else {
            cout << endl;
        }
    }
    cout << endl;
}

// меняет местами строки
template <typename T>
void Matrix<T>::swap_rows(int i1, int i2) {
    check_i_j(i1, 0);
    check_i_j(i2, 0);
    T sw;
    for (int j = 0; j < _cols; j++) {
        sw = _arr[i1][j];
        _arr[i1][j] = _arr[i2][j];
        _arr[i2][j] = sw;
    }
    return;
}

// удаляет матрицу и очищает память
template <typename T>
Matrix<T>::~Matrix() {
    for (int i = 0; i < _rows; i++) {
        delete[] _arr[i];
    }
    delete[] _arr;
}

template class Matrix<double>;
```

Главные вычисления программы происходят в файле main.cpp. Текст этой части программы приведен в листинге 5.

Листинг 5. main.cpp

```
#include <stdio.h>
#include "lab1.hpp"
#include "matrix.hpp"

// матрица состояний пластины
double p[I_MAX + 1][J_MAX + 1][K_MAX + 1];

// напечатать состояние
void print_state(int k) {
    cout << "" << endl;
    for (int i = 0; i <= I_MAX; i++) {
        for (int j = 0; j <= J_MAX; j++) {
            cout << setw(7) << p[i][j][k] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

// задать начальные условия
void begin_state() {
    // внутренние узлы из НУ
    for (int i = 1; i < I_MAX; i++) {
        for (int j = 1; j < J_MAX; j++) {
            p[i][j][0] = T_0;
        }
    }

    // верхняя и нижняя границы из ГУ
    for (int j = 0; j <= J_MAX; j++) {
        p[I_MAX][j][0] = T_BOTTOM();
        p[0][j][0] = T_UP();
    }

    // левая и правая границы из ГУ
    for (int i = 1; i <= I_MAX; i++) {
        p[i][0][0] = T_LEFT(p[i][1][0]);
        p[i][J_MAX][0] = T_RIGHT();
    }

}

// рассчитать состояние
void count_state(int k) {
    // расчет начальных условий
    if (k == 0) {
        begin_state();
        return;
    }

    // создание исходных матриц СЛАУ
```

```
        int size = (I_MAX + 1) * (J_MAX + 1);
        int cmax = size - 1;    // максимальный номер строки в векторе c
        Matrix <double> c(size, 1);
        Matrix <double> A(size, size);

        // Задание граничных условий в матрице

        // верх и низ
        // граничное условие первого рода
        for (int m = 0; m <= J_MAX; m++) {
            A.set(m, m, 1);
            c.set(m, 0, T_UP());

            A.set(cmax - m, cmax - m, 1);
            c.set(cmax - m, 0, T_BOTTOM());
        }

        // левая граница - каждый J_MAX + 1 -ый элемент
        // граничное условие второго рода
        for (int m = 0; m <= cmax; m += J_MAX + 1) {
            A.set(m, m, 1);
            A.set(m, m + 1, -1);
            c.set(m, 0, 0);
        }

        // правая граница - каждый J_MAX -ый элемент
        // граничное условие первого рода
        for (int m = J_MAX; m <= cmax; m += J_MAX + 1) {
            A.set(m, m, 1);
            c.set(m, 0, T_RIGHT());
        }

        // неявная разностная схема
        // пропускаем все граничные условия по верху
        // и первое по левой границе
        // в условии - останавливаемся до последней правой границы

        // коэффициенты
        double Tij_coef = D_X*D_X*D_Y*D_Y + 2*D_Y*D_Y*D_T + 2*D_X*D_X*D_T;
        double Ti_jv_coef = -D_Y*D_Y*D_T;
        double Tiv_j_coef = -D_X*D_X*D_T;
        double T_prev_coeff = D_Y*D_Y*D_X*D_X;

        for (int q = J_MAX + 2; q < cmax - J_MAX; q += J_MAX + 1 ) {
            for (int m = 0; m < J_MAX - 1; m++) {
                A.set(q+m, q+m, Tij_coef);              // T i,j
                A.set(q+m, q+m+J_MAX+1, Tiv_j_coef);    // T i+1,j
                A.set(q+m, q+m-J_MAX-1, Tiv_j_coef);    // T i-1,j
                A.set(q+m, q+m+1, Ti_jv_coef);          // T i,j+1
                A.set(q+m, q+m-1, Ti_jv_coef);          // T i,j-1
                                                        c.set(q+m,     0,
p[(q+m)/(J_MAX+1)][(q+m)%(J_MAX+1)][k-1]*T_prev_coeff);
            }
        }

        // int middle = (int)(cmax/2);
        // c.set(middle, 0, c.get(middle, 0) + 500);
```

```
    //A.print(&c, "Исходное уравнение:");
    A.Gauss(&c);

    // перенос значений из вектора c в матрицу p
    int gi = 0;
    for (int i = 0; i < I_MAX + 1; i++) {
        for (int j = 0; j < J_MAX + 1; j++) {
            p[i][j][k] = c.get(gi, 0);
            gi++;
        }
    }

}

int main () {
    for (int k = 0; k <= K_MAX; k++) {
        count_state(k);
        print_state(k);
    }

    return 0;
}
```

В функции main для указанного количества временных шагов производится расчет состояние с помощью функции count_state и его вывод с помощью print_state.

Функция print_state только выводит матрицу состояния пластины во времени, названную в программе именем p. Эта матрица представляет собой трехмерный массив, измерениями которого является пространственно-временная сетка.

Функция count_state разделяется на два варианта:

1. Для момента времени k=0 она вызывает функцию begin_state, которая заполняет нулевой временной слой массива состояний пластины p следующим образом: внутренние узлы принимают начальное значение температуры, а граничные - в зависимости от граничных условий.

2. Для других моментов времени программа производит вычисления с помощью неявной разностной схемы с учетом граничных условий.

Вычисления состояния на ненулевом моменте времени начинаются с создания матрицы A - для коэффициентов и вектора c - для свободных членов.

Вектор неизвестных в программе не создается, так как в результате применения метода Гаусса ответы получаются в преобразованном векторе c. Подразумевается, что неизвестные температуры располагаются в векторе c в порядке, указанном формулой 1. Исходя из этого порядка и строится матрица A.

$$c^T = (T_{00}, T_{01}, T_{02}, \ldots, T_{0Jmax}, T_{10}, T_{11}, \ldots, T_{Imax\,Jmax}) \quad (1)$$

Сначала матрица A заполняется граничными условиями. Для первого рода в матрице на диагонали, на строке, соответствующей номеру неизвестной в векторе c ставится единица, а в векторе c на той же строке - значение в данном узле.

Граничное условие второго рода (теплоизоляция) задается аналогично: наружному узлу в соответствие ставится коэффициент один, а соседнему с ним узлу по оси X - минус один. В правой части остается ноль, так как пластина теплоизолирована.

Далее в программе с матрицы заносятся уравнения неявной разностной схемы. Исходное уравнение неявной разностной схемы представлено формулой 2.

$$\frac{T_{ij}^{k+1} - T_{ij}^{k}}{\Delta t} = \frac{T_{ij+1}^{k} - 2T_{ij}^{k} + T_{ij-1}^{k}}{\Delta y^2} + \frac{T_{i+1j}^{k} - 2T_{ij}^{k} + T_{i-1j}^{k}}{\Delta x^2} \quad (2)$$

После заполнения матриц происходит вызов метода Гаусса из файла matrix.cpp и занесение итогового вектора c в матрицу состояний p.

Визуализация результатов работы программы выполнялась с помощью утилиты gnuplot. Скрипт для создания изображения представлен в листинге 6. Данный скрипт обрабатывает одну матрицу, помещенную в файл matrix.dat и создает изображение под названием result.png.

Листинг 6. plot.gnu

```
set terminal png
set output "result.png"
```

```
set size ratio 0.5
set title "Температура пластины"

set xlabel "X"
set ylabel "Y"

set tic scale 0

set palette rgbformulae 10,13,22
set palette negative

set palette maxcolors 400
set cbrange [0:1100]
#unset cbtics

set xrange [0:10]
set yrange [0:10]

set view map

splot 'matrix.dat' matrix with image
```

Результаты работы программы

Программы выдает результаты в виде набора матриц в каждый момент времени с температурами пластины по узлам пространственной сетки. Результат работы для сетки 10 на 10 клеток (11 х 11 узлов) показан в листинге 7.

Листинг 7. Результат работы программы для сетки 10 х 10

```
  800   800   800   800   800   800   800   800   800   800   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800
   20    20    20    20    20    20    20    20    20    20   800

  800   800   800   800   800   800   800   800   800   800   800
497.539 497.539 498.248  499.91 503.102 508.955 519.623 539.266 576.279 648.799   800
313.755 313.755 315.106 318.268 324.317 335.334 355.177 390.979  456.12 576.116   800
202.057 202.057 203.931 208.305 216.642 231.722 258.565 306.097 390.136 538.573   800
    134     134 136.237 141.455 151.385 169.279 200.903 256.235 352.286 517.683   800
92.2076 92.2076 94.6196 100.258 111.011 130.426 164.759 224.712 328.162 504.296   800
66.0548 66.0548 68.4308 74.0106 84.7287 104.262 139.183 200.872 308.395 492.411   800
49.0396 49.0396 51.1457 56.1276 65.8134 83.7736 116.636 176.415 284.233 475.056   800
37.1783 37.1783 38.7714 42.5726 50.0771 64.3266 91.3325 143.022 243.124 437.345   800
28.0185 28.0185 28.8817 30.9575 35.1152 43.2047 59.1785 91.9712 163.678 335.535   800
   20    20    20    20    20    20    20    20    20    20   800
```

800 800 800 800 800 800 800 800 800 800 800

616.386 616.386 617.817 620.977 626.505 635.456 649.397 670.483 701.376 744.597 800

461.053 461.053 463.795 469.852 480.439 497.547 524.092 563.965 621.696 700.963 800

339.802 339.802 343.632 352.094 366.882 390.755 427.687 482.835 561.875 668.771 800

249.157 249.157 253.764 263.955 281.801 310.661 355.346 421.998 517.131 644.797 800

182.694 182.694 187.693 198.786 218.311 250.088 299.65 374.12 481.067 625.063 800

133.854 133.854 138.795 149.819 169.392 201.647 252.77 331.116 446.181 604.581 800

97.0229 97.0229 101.405 111.254 128.958 158.683 207.063 283.953 402.407 575.123 800

67.8049 67.8049 71.1153 78.6135 92.2894 115.791 155.454 222.11 333.913 518.522 800

42.9332 42.9332 44.7235 48.8057 56.348 69.6052 92.8816 134.893 215.311 385.345 800

20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800

664.357 664.357 666.161 669.996 676.314 685.765 699.133 717.187 740.41 768.545 800

538.904 538.904 542.376 549.764 561.948 580.192 606.013 640.882 685.667 739.757 800

429.442 429.442 434.319 444.715 461.899 487.702 524.324 573.891 637.612 714.513 800

337.513 337.513 343.416 356.034 376.984 408.628 453.852 515.507 595.312 692.12 800

261.77 261.77 268.211 282.04 305.17 340.467 391.576 462.352 555.554 670.487 800

199.355 199.355 205.748 219.559 242.904 279.088 332.613 408.829 512.685 645.629 800

146.921 146.921 152.606 164.979 186.177 219.729 270.919 347.095 457.354 609.608 800

101.278 101.278 105.575 115.004 131.4 157.996 200.202 267.065 373.767 544.067 800

59.7045 59.7045 62.0286 67.161 76.2001 91.2025 116.014 158.434 236.544 399.282 800

20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800

690.31 690.31 692.19 696.097 702.309 711.182 723.075 738.215 756.534 777.496 800

585.229 585.229 588.859 596.416 608.455 625.7 648.888 678.508 714.458 755.687 800

488.098 488.098 493.223 503.917 521.02 545.652 578.988 621.876 674.3 734.777 800

400.507 400.507 406.744 419.808 440.835 471.389 513.203 567.706 635.261 714.188 800

322.427 322.427 329.27 343.68 367.088 401.555 449.566 513.533 594.837 692.229 800

252.669 252.669 259.493 273.964 297.76 333.451 384.468 454.82 548.145 665.327 800

189.416 189.416 195.505 208.523 230.251 263.613 313 384.63 486.461 625.857 800

130.698 130.698 135.312 145.26 162.127 188.719 229.81 293.624 394.488 555.704 800

74.7211 74.7211 77.2194 82.6421 91.959 107.007 131.3 172.206 247.344 405.381 800
20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800
706.281 706.281 708.087 711.796 717.579 725.645 736.169 749.2 764.562 781.767 800
614.748 614.748 618.246 625.442 636.694 652.449 673.103 698.816 729.29 763.571 800
527.097 527.097 532.056 542.286 558.367 581.042 611.031 648.747 693.918 745.2 800
444.237 444.237 450.302 462.871 482.783 511.167 549.242 597.941 657.339 725.917 800
366.244 366.244 372.933 386.88 409.212 441.547 485.84 543.996 617.098 704.061 800
292.518 292.518 299.22 313.305 336.169 369.968 417.641 482.72 568.561 676.189 800
222.051 222.051 228.056 240.788 261.791 293.646 340.33 407.658 503.34 634.85 800
153.737 153.737 158.302 168.066 184.446 210.002 249.209 309.998 406.511 562.119 800
86.6194 86.6194 89.0954 94.4296 103.504 118.027 141.357 180.706 253.594 408.72 800
20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800
716.619 716.619 718.306 721.751 727.074 734.419 743.892 755.493 769.039 784.1 800
634.135 634.135 637.411 644.11 654.501 668.903 687.585 710.615 737.683 767.946 800
553.222 553.222 557.882 567.446 582.367 603.219 630.552 664.65 705.228 751.094 800
474.181 474.181 479.906 491.716 510.302 536.601 571.632 616.183 670.311 732.678 800
396.902 396.902 403.245 416.421 437.403 467.61 508.789 562.698 630.4 710.995 800
320.943 320.943 327.33 340.707 362.327 394.161 438.953 500.095 580.924 682.634 800
245.705 245.705 251.451 263.599 283.575 313.803 358.096 422.149 513.656 640.23 800
170.636 170.636 175.017 184.368 200.019 224.418 261.921 320.372 413.899 565.974 800
95.4097 95.4097 97.7911 102.912 111.609 125.532 147.977 186.11 257.444 410.729 800
20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800
723.443 723.443 725.016 728.22 733.156 739.942 748.663 759.311 771.713 785.479 800
647.018 647.018 650.078 656.324 665.983 679.331 696.594 717.822 742.729 770.548 800
570.755 570.755 575.121 584.067 597.994 617.41 642.81 674.457 712.094 754.635 800
494.508 494.508 499.892 510.987 528.42 553.055 585.845 627.552 678.272 736.782 800

417.959 417.959 423.95 436.386 456.174 484.657 523.516 574.479 638.649 715.248 800
340.683 340.683 346.74 359.424 379.926 410.146 452.763 511.143 588.66 686.623 800
262.285 262.285 267.754 279.322 298.361 327.233 369.701 431.434 520.158 643.583 800
182.564 182.564 186.746 195.68 210.658 234.083 270.273 327.055 418.58 568.387 800
101.642 101.642 103.92 108.823 117.168 130.583 152.342 189.604 259.891 411.991 800
20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800
727.98 727.98 729.46 732.475 737.118 743.499 751.698 761.711 773.377 786.332 800
655.614 655.614 658.498 664.385 673.488 686.067 702.343 722.368 745.882 772.163 800
582.512 582.512 586.638 595.094 608.259 626.625 650.673 680.674 716.406 756.844 800
508.224 508.224 513.328 523.85 540.394 563.803 595.017 634.805 683.301 739.359 800
432.261 432.261 437.96 449.798 468.66 495.864 533.079 582.041 643.893 717.934 800
354.173 354.173 359.955 372.075 391.704 420.718 461.784 518.276 593.606 689.158 800
273.678 273.678 278.914 290.006 308.307 336.161 377.319 437.458 524.335 645.723 800
190.794 190.794 194.809 203.399 217.844 240.533 275.777 331.407 421.598 569.934 800
105.953 105.953 108.143 112.866 120.932 133.961 155.225 191.884 261.472 412.801 800
20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800
731.005 731.005 732.416 735.291 739.723 745.82 753.663 763.253 774.44 786.874 800
661.355 661.355 664.108 669.73 678.432 690.472 706.072 725.295 747.9 773.193 800
590.386 590.386 594.333 602.425 615.04 632.666 655.788 684.689 719.173 758.256 800
517.441 517.441 522.334 532.431 548.332 570.874 601.003 639.504 686.541 741.012 800
441.906 441.906 447.385 458.778 476.966 503.264 539.343 586.958 647.282 719.664 800
363.304 363.304 368.876 380.576 399.566 427.722 467.714 522.931 596.815 690.795 800
281.413 281.413 286.472 297.207 314.967 342.094 382.342 441.401 527.053 647.11 800
196.395 196.395 200.282 208.613 222.667 244.829 279.414 334.262 423.566 570.938 800
108.892 108.892 111.014 115.602 123.463 136.215 157.134 193.381 262.504 413.328 800
20 20 20 20 20 20 20 20 20 20 800

800 800 800 800 800 800 800 800 800 800 800
733.023 733.023 734.385 737.161 741.446 747.347 754.95 764.258 775.131 787.226 800

665.189	665.189	667.848	673.283	681.705	693.373	708.515	727.205	749.211	773.861	800
595.653	595.653	599.471	607.306	619.535	636.651	659.145	687.312	720.975	759.173	800
523.617	523.617	528.36	538.154	553.603	575.547	604.94	642.58	688.653	742.087	800
448.383	448.383	453.703	464.78	482.494	508.164	543.471	590.183	649.497	720.792	800
369.448	369.448	374.87	386.268	404.809	432.37	471.629	525.99	598.915	691.864	800
286.626	286.626	291.558	302.038	319.417	346.038	385.664	443.996	528.836	648.017	800
200.176	200.176	203.97	212.117	225.893	247.689	281.824	336.145	424.859	571.596	800
110.877	110.877	112.951	117.441	125.157	137.717	158.399	194.37	263.183	413.673	800
20	20	20	20	20	20	20	20	20	20	800

Сравнение решения с решением в программе ANSYS

Визуализация с помощью gnuplot (рис. 1) и ANSYS (рис. 2) позволила проверить правильность выполнения программы. Различия лишь в отображении пластины: в gnuplot использовалась более крупная сетке, нумерация там идет по оси Y снизу вверх.

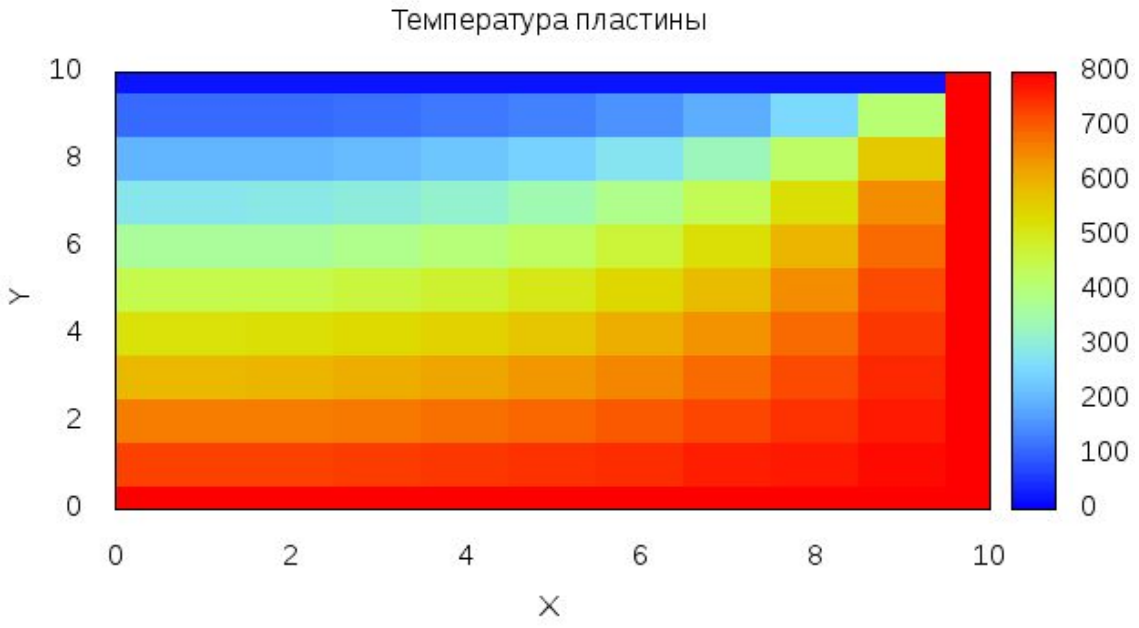

Рис. 1. Визуализация решения в момент времени 10 секунд с помощью gnuplot

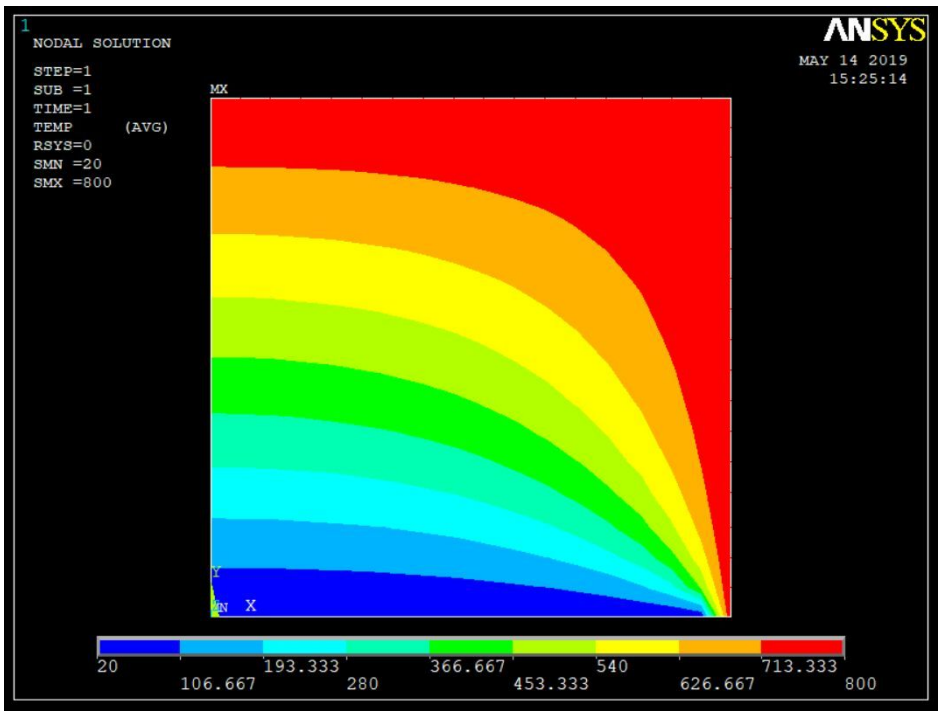

Рис. 2. Визуализация решения в ANSYS

Список использованных источников

1. Трудоношин В.А. Лекции по курсу “Модели и методы проектных решений”.