Оглавление

[1.Постановка задачи 3](#_Toc200451833)

[1.1 Уравнение теплопроводности 3](#_Toc200451834)

[1.2 Начальные и граничные условия 3](#_Toc200451835)

[1.3 Источники тепла 4](#_Toc200451836)

[2.Методы решения. 5](#_Toc200451837)

[2.1 Метод расщепления. 5](#_Toc200451838)

[2.2 Неявная разностная схема. 5](#_Toc200451839)

[2.3 Метод прогонки (алгоритм Томаса). 5](#_Toc200451840)

[3.Реализация. 6](#_Toc200451841)

[3.1 Структура программы. 6](#_Toc200451842)

[3.2 Учёт граничных условий 7](#_Toc200451843)

[3.3 Источники тепла 7](#_Toc200451844)

[4. Листинг программы 8](#_Toc200451845)

[Листинг 1 – class Courswork 8](#_Toc200451846)

[Листинг 2 – class Solve 11](#_Toc200451847)

[Листинг 3 – class Create 16](#_Toc200451848)

[Листинг 4 – class Apply 20](#_Toc200451849)

[Листинг 5 – class Result 27](#_Toc200451850)

[5. Пример работы программы 33](#_Toc200451851)

[6. Заключение 36](#_Toc200451852)

[7. Список литературы 37](#_Toc200451853)

**Тема:** Численное решение начально-краевой задачи для нестационарного уравнения теплопроводности в трехмерном случае, конечно-разностным методом расщепления с учетом граничных условий Дирихле, Неймана и Робина на всех границах, возможностью задания источников тепла (включая точечные), а также визуализацией результатов в виде графиков распределения температуры вдоль фиксированных линий и двумерных линий уровня теплового поля на сечениях трехмерной области для анализа пространственно-временной динамики температурного поля.

# 1.Постановка задачи

## 1.1 Уравнение теплопроводности

Рассматривается нестационарное уравнение теплопроводности в трёхмерной области:

Где:

* u(x, y, z, t) – температура в точке (x, y, z) в момент времени t
* α – коэффициент температуропроводности
* Q(x, y, z, t) – функция источников тепла

## 1.2 Начальные и граничные условия

* Начальное условие:

где – заданное начальное распределение температуры.

* Граничные условия:

На границах области могут быть заданы условия трёх типов:

* + 1. Дирихле: u = g (фиксированная температура),
    2. Неймана: (тепловой поток)
    3. Робина: (смешанное условие).

## 1.3 Источники тепла

Функция Q(x, y, z, t) может описывать:

* Распределённые источники (например, равномерный нагрев),
* Точечные источники(моделируются дельта-функцией или её аппроксимацией).

# 2.Методы решения.

## 2.1 Метод расщепления.

Идея метода – разбить сложную трёхмерную задачу на последовательность более простых одномерных:

1. Решение для каждого слоя при фиксированных y и z:
2. Решение для каждого слоя при фиксированных x и z:
3. Решение для каждого слоя при фиксированных x и y:

## 2.2 Неявная разностная схема.

Для каждого этапа расщепления используется неявная схема, которая обеспечивает устойчивость при любых шагах по времени. Например, разностное уравнение для этапа по x:

где .

## 2.3 Метод прогонки (алгоритм Томаса).

Для решения трёхдиагональных систем на каждом этапе применяется метод прогонки, который имеет линейную сложность O(N2).

# 3.Реализация.

## 3.1 Структура программы.

Программа реализована на языке C# с использованием библиотеки MathNet.Numerics для работы с линейной алгеброй (матрицами и векторами).

Основные компоненты программы:

* Класс Coursework:

Содержит точку входа Main, где задаются параметры задачи (Размеры области, шаги сетки, граничные условия) и вызывается метод решения.

* Класс Solve:

Реализует метод расщепления для трёхмерного уравнения теплопроводности.

Основные методы:

* + SolveGeneral – управляет процессом решения, инициализирует сетку и вызывает пошаговое решение по осям.
  + SolveX, SolveY, Solvez – решают одномерные задачи для каждого направления с применением неявной схемы.
* Классы Create и Apply:
  + Create – создаёт трёхдиагональные матрицы и правые части для СЛАУ.
  + Apply – применяет граничные условия (Дирихле, Неймана, Робина) к матрицам и векторам
* Класс Results:

Обеспечивает вывод результатов:

* + Сохранение в файл Result.txt
  + Интерактивную визуализацию срезов по осям в консоли.
* Класс StartWrite:

Управляет вводом параметров и начальных условий через консоль.

## 3.2 Учёт граничных условий

Граничные условия задаются в словаре boundaryConditions с ключами "x0", "x1", "y0", "y1", "z0", "z1". Каждое условие имеет тип (0 — Дирихле, 1 — Неймана, 2 — Робин) и значение.

## 3.3 Источники тепла

Точечные источники задаются членом уравнения теплопроводности, а также списком координат и мощности (точечные источники)

# 4. Листинг программы

## Листинг 1 – class Courswork

using ClassLibrary;

public class Coursework

{

public static void Main(string[] args)

{

int nx, ny, nz;

double lx, ly, lz, tMax, dt, alpha;

Func<double, double, double, double> initialCondition;

Console.WriteLine("1 - вручную\n2 - автоматически");

int swCheck = Convert.ToInt16(Console.ReadLine());

switch (swCheck)

{

case 1:

StartWrite.Write();

Console.WriteLine("Введите nx:");

nx = Convert.ToInt32(Console.ReadLine());

StartWrite.Write(nx);

Console.WriteLine("Введите ny:");

ny = Convert.ToInt32(Console.ReadLine());

StartWrite.Write(nx, ny);

Console.WriteLine("Введите nz:");

nz = Convert.ToInt32(Console.ReadLine());

StartWrite.Write(nx, ny, nz);

Console.WriteLine("Введите lx:");

lx = Convert.ToDouble(Console.ReadLine());

StartWrite.Write(nx, ny, nz, lx);

Console.WriteLine("Введите ly:");

ly = Convert.ToDouble(Console.ReadLine());

StartWrite.Write(nx, ny, nz, lx, ly);

Console.WriteLine("Введите lz:");

lz = Convert.ToDouble(Console.ReadLine());

StartWrite.Write(nx, ny, nz, lx, ly, lz);

Console.WriteLine("Введите tMax:");

tMax = Convert.ToDouble(Console.ReadLine());

StartWrite.Write(nx, ny, nz, lx, ly, lz, tMax);

Console.WriteLine("Введите dt:");

dt = Convert.ToDouble(Console.ReadLine());

StartWrite.Write(nx, ny, nz, lx, ly, lz, tMax, dt);

Console.WriteLine("Введите alpha:");

alpha = Convert.ToDouble(Console.ReadLine());

StartWrite.Write(nx, ny, nz, lx, ly, lz, tMax, dt, alpha);

var (function, description) = StartWrite.InitialCondition(lx, ly, lz);

initialCondition = function;

StartWrite.Write(nx, ny, nz, lx, ly, lz, tMax, dt, alpha, description);

break;

default:

nx = 10;

ny = 10;

nz = 10;

lx = 1.0;

ly = 1.0;

lz = 1.0;

tMax = 0.1;

dt = 0.001;

alpha = 0.1;

initialCondition = (x, y, z) => 0.0;

StartWrite.WriteDefault(nx, ny, nz, lx, ly, lz, tMax, dt, alpha);

break;

}

// Граничные условия (Дирихле, Неймана и Робин)

var boundaryConditions = new Dictionary<string, (int type, double value)>

{

{ "x0", (0, 15.0) }, // Дирихле: u(0, y, z, t) = 20

{ "x1", (1, 0.0) }, // Неймана: du/dx(lx, y, z, t) = 0

{ "y0", (0, 30.0) }, // Дирихле: u(x, 0, z, t) = 30

{ "y1", (1, 0.0) }, // Неймана: du/dy(x, ly, z, t) = 0

{ "z0", (0, 60.0) }, // Дирихле: u(x, y, 0, t) = 40

{ "z1", (1, 0.0) } // Неймана: du/dz(x, y, lz, t) = 0

};

// Коэффициенты для граничного условия Робина

var robinCoefficients = new Dictionary<string, (double alpha, double beta)>

{

{ "z0", (1.0, 1.0) }

};

List<(double x, double y, double z, double Q)> pointSources = new List<(double x, double y, double z, double Q)>

{

(lx / 4, ly / 4, lz / 4, 50),

(3 \* lx / 4, 3 \* ly / 4, 3 \* lz / 4, 50)

};

HeatSourceFunction heatSource = (x, y, z, t) =>

{

return 100 \* (x \* (lx - x) \* y \* (ly - y) \* z \* (lz - z)) \* Math.Exp(-t);

};

double[,,] solution = Solve.SolveGeneral(nx, ny, nz, lx, ly, lz, tMax, dt, alpha, initialCondition, boundaryConditions, robinCoefficients, heatSource, pointSources);

Console.WriteLine("\nВыберите способ вывода результатов:\n1) Вывод в .txt файл\n2) Визуализировать в консоли");

string writeCheck = Console.ReadLine();

switch (writeCheck)

{

default:

Results.WriteToFile(solution);

break;

case "2":

while (true)

{

Console.Clear();

Console.WriteLine("\n1) Выбрать срез\n2) Выйти");

int wCheck = Convert.ToInt16(Console.ReadLine());

if(wCheck == 1)

{

Results.PrintSlise(solution);

}

else

{

break;

}

}

break;}}}

## Листинг 2 – class Solve

using MathNet.Numerics.LinearAlgebra;

namespace ClassLibrary

{

public delegate double HeatSourceFunction(double x, double y, double z, double t);

public class Solve

{

public static double[,,] SolveGeneral(

int nx, int ny, int nz,

double lx, double ly, double lz,

double tMax,

double dt,

double alpha,

Func<double, double, double, double> initialCondition,

Dictionary<string, (int type, double value)> boundaryConditions,

Dictionary<string, (double alpha, double beta)> robinCoefficients,

HeatSourceFunction heatSource,

List<(double x, double y, double z, double Q)> pointSources)

{

double dx = lx / (nx - 1);

double dy = ly / (ny - 1);

double dz = lz / (nz - 1);

double rx = alpha \* dt / (2.0 \* dx \* dx);

double ry = alpha \* dt / (2.0 \* dy \* dy);

double rz = alpha \* dt / (2.0 \* dz \* dz);

double[,,] solution = new double[nx, ny, nz];

for (int i = 0; i < nx; i++)

{

for (int j = 0; j < ny; j++)

{

for (int k = 0; k < nz; k++)

{

double x = i \* dx;

double y = j \* dy;

double z = k \* dz;

solution[i, j, k] = initialCondition(x, y, z);

}

}

}

for (double t = 0; t < tMax; t += dt)

{

SolveX(solution, alpha, dt, rx, nx, ny, nz, dx, dy, dz, boundaryConditions, robinCoefficients, pointSources, heatSource, t);

SolveY(solution, alpha, dt, ry, nx, ny, nz, dx, dy, dz, boundaryConditions, robinCoefficients, pointSources, heatSource, t);

SolveZ(solution, alpha, dt, rz, nx, ny, nz, dx, dy, dz, boundaryConditions, robinCoefficients, pointSources, heatSource, t);

}

return solution;

}

private static void SolveX(

double[,,] solution,

double alpha,

double dt,

double rx,

int nx,

int ny,

int nz,

double dx,

double dy,

double dz,

Dictionary<string, (int type, double value)> boundaryConditions,

Dictionary<string, (double alpha, double beta)> robinCoefficients,

HeatSourceFunction heatSource,

List<(double x, double y, double z, double Q)> pointSources, double t)

{

for (int j = 0; j < ny; j++)

{

for (int k = 0; k < nz; k++)

{

Matrix<double> matrix = Create.TridiagonalMatrixX(nx, rx);

Vector<double> vector = Create.RightHandSideX(solution, alpha, dt, rx, nx, j, k, dx, dy, dz, pointSources, t);

Apply.BoundaryConditionsX(matrix, vector, rx, nx, j, dx, boundaryConditions, robinCoefficients);

Vector<double> x = matrix.Solve(vector);

for (int i = 0; i < nx; i++)

{

solution[i, j, k] = x[i];

}

}

}

}

private static void SolveY(double[,,] solution, double alpha, double dt, double ry, int nx, int ny, int nz, double dx, double dy, double dz, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients, HeatSourceFunction heatSource, List<(double x, double y, double z, double Q)> pointSources, double t)

{

for (int i = 0; i < nx; i++)

{

for (int k = 0; k < nz; k++)

{

Matrix<double> matrix = Create.TridiagonalMatrixY(ny, ry);

Vector<double> vector = Create.RightHandSideY(solution, alpha, dt, ry, ny, i, k, dx, dy, dz, pointSources, t);

Apply.BoundaryConditionsY(matrix, vector, ry, ny, dy, boundaryConditions, robinCoefficients);

Vector<double> y = matrix.Solve(vector);

for (int j = 0; j < ny; j++)

{

solution[i, j, k] = y[j];

}

}

}

}

private static void SolveZ(double[,,] solution, double alpha, double dt, double rz, int nx, int ny, int nz, double dx, double dy, double dz, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients, HeatSourceFunction heatSource, List<(double x, double y, double z, double Q)> pointSources, double t)

{

for (int i = 0; i < nx; i++)

{

for (int j = 0; j < ny; j++)

{

Matrix<double> matrix = Create.TridiagonalMatrixZ(nz, rz);

Vector<double> vector = Create.RightHandSideZ(solution, alpha, dt, rz, nz, i, j, dx, dy, dz, pointSources, t);

Apply.BoundaryConditionsZ(matrix, vector, rz, nz, dz, boundaryConditions, robinCoefficients);

Vector<double> z = matrix.Solve(vector);

for (int k = 0; k < nz; k++)

{

solution[i, j, k] = z[k];

}

}

}

}

}

}

## Листинг 3 – class Create

using MathNet.Numerics.LinearAlgebra;

namespace ClassLibrary

{

public class Create

{

public static Matrix<double> TridiagonalMatrixX(int nx, double rx)

{

var matrix = Matrix<double>.Build.Dense(nx, nx, 0.0);

for (int i = 1; i < nx - 1; i++)

{

matrix[i, i - 1] = -rx;

matrix[i, i] = 1 + 2 \* rx;

matrix[i, i + 1] = -rx;

}

return matrix;

}

public static Matrix<double> TridiagonalMatrixY(int ny, double ry)

{

var matrix = Matrix<double>.Build.Dense(ny, ny, 0.0);

for (int i = 1; i < ny - 1; i++)

{

matrix[i, i - 1] = -ry;

matrix[i, i] = 1 + 2 \* ry;

matrix[i, i + 1] = -ry;

}

return matrix;

}

public static Matrix<double> TridiagonalMatrixZ(int nz, double rz)

{

var matrix = Matrix<double>.Build.Dense(nz, nz, 0.0);

for (int i = 1; i < nz - 1; i++)

{

matrix[i, i - 1] = -rz;

matrix[i, i] = 1 + 2 \* rz;

matrix[i, i + 1] = -rz;

}

return matrix;

}

public static Vector<double> RightHandSideX(double[,,] solution, double alpha, double dt, double rx, int nx, int j, int k, double dx, double dy, double dz, List<(double x, double y, double z, double Q)> pointSources, double t)

{

Vector<double> vector = Vector<double>.Build.Dense(nx, 0.0);

double y = j \* dy;

double z = k \* dz;

for (int i = 1; i < nx - 1; i++)

{

double x = i \* dx;

vector[i] = solution[i, j, k] + rx \* (solution[i - 1, j, k] - 2 \* solution[i, j, k] + solution[i + 1, j, k]);

if (pointSources != null)

{

foreach (var source in pointSources)

{

double distance = Math.Sqrt(Math.Pow(x - source.x, 2) + Math.Pow(y - source.y, 2) + Math.Pow(z - source.z, 2));

if (distance <= dx / 2)

{

vector[i] += alpha \* dt \* source.Q / (dx \* dy \* dz);

}

}

}

}

return vector;

}

public static Vector<double> RightHandSideY(double[,,] solution, double alpha, double dt, double ry, int ny, int i, int k, double dx, double dy, double dz, List<(double x, double y, double z, double Q)> pointSources, double t)

{

Vector<double> vector = Vector<double>.Build.Dense(ny, 0.0);

double x = i \* dx;

double z = k \* dz;

for (int j = 1; j < ny - 1; j++)

{

double y = j \* dy;

vector[j] = solution[i, j, k] + ry \* (solution[i, j - 1, k] - 2 \* solution[i, j, k] + solution[i, j + 1, k]);

if (pointSources != null)

{

foreach (var source in pointSources)

{

double distance = Math.Sqrt(Math.Pow(x - source.x, 2) + Math.Pow(y - source.y, 2) + Math.Pow(z - source.z, 2));

if (distance <= dy / 2)

{

vector[j] += alpha \* dt \* source.Q / (dx \* dy \* dz);

}

}

}

}

return vector;

}

public static Vector<double> RightHandSideZ(double[,,] solution, double alpha, double dt, double rz, int nz, int i, int j, double dx, double dy, double dz, List<(double x, double y, double z, double Q)> pointSources, double t)

{

Vector<double> vector = Vector<double>.Build.Dense(nz, 0.0);

double x = i \* dx;

double y = j \* dy;

for (int k = 1; k < nz - 1; k++)

{

double z = k \* dz;

vector[k] = solution[i, j, k] + rz \* (solution[i, j, k - 1] - 2 \* solution[i, j, k] + solution[i, j, k + 1]);

if (pointSources != null)

{

foreach (var source in pointSources)

{

double distance = Math.Sqrt(Math.Pow(x - source.x, 2) + Math.Pow(y - source.y, 2) + Math.Pow(z - source.z, 2));

if (distance <= dz / 2)

{

vector[k] += alpha \* dt \* source.Q / (dx \* dy \* dz);

}

}

}

}

return vector;

}

}

}

## Листинг 4 – class Apply

using MathNet.Numerics.LinearAlgebra;

namespace ClassLibrary

{

public class Apply

{

public static void BoundaryConditionsX(Matrix<double> matrix, Vector<double> vector, double rx, int nx, int j, double dx, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

// Граница X0

if (boundaryConditions.ContainsKey("x0"))

{

var bc = boundaryConditions["x0"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(0, Vector<double>.Build.Dense(nx));

matrix[0, 0] = 1;

vector[0] = bc.value;

break;

case 1: // Нейман

matrix[0, 0] = 1 + 2 \* rx;

matrix[0, 1] = -2 \* rx;

vector[0] = vector[0] + 2 \* rx \* dx \* bc.value;

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("x0"))

{

throw new ArgumentException("Robin coefficients are missing for x0 boundary.");

}

var robin = robinCoefficients["x0"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[0, 0] = alpha \* (1 + 2 \* rx) + beta \* (2 \* rx / dx);

matrix[0, 1] = -alpha \* (2 \* rx);

vector[0] = alpha \* vector[0] + 2 \* beta \* rx \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for x0 is missing. Setting to zero gradient");

matrix[0, 0] = 1 + 2 \* rx;

matrix[0, 1] = -2 \* rx;

vector[0] = vector[0];

}

// Граница X1

if (boundaryConditions.ContainsKey("x1"))

{

var bc = boundaryConditions["x1"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(nx - 1, Vector<double>.Build.Dense(nx));

matrix[nx - 1, nx - 1] = 1;

vector[nx - 1] = bc.value;

break;

case 1: // Нейман

matrix[nx - 1, nx - 1] = 1 + 2 \* rx;

matrix[nx - 1, nx - 2] = -2 \* rx;

vector[nx - 1] = vector[nx - 1] - 2 \* rx \* dx \* bc.value;

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("x1"))

{

throw new ArgumentException("Robin coefficients are missing for x1 boundary.");

}

var robin = robinCoefficients["x1"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[nx - 1, nx - 1] = alpha \* (1 + 2 \* rx) + beta \* (2 \* rx / dx);

matrix[nx - 1, nx - 2] = -alpha \* (2 \* rx);

vector[nx - 1] = alpha \* vector[nx - 1] + 2 \* beta \* rx \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for x1 is missing. Setting to zero gradient");

matrix[nx - 1, nx - 1] = 1 + 2 \* rx;

matrix[nx - 1, nx - 2] = -2 \* rx;

vector[nx - 1] = vector[nx - 1];

}

}

public static void BoundaryConditionsY(Matrix<double> matrix, Vector<double> vector, double ry, int ny, double dy, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

// Граница Y0

if (boundaryConditions.ContainsKey("y0"))

{

var bc = boundaryConditions["y0"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(0, Vector<double>.Build.Dense(ny));

matrix[0, 0] = 1;

vector[0] = bc.value;

break;

case 1: // Нейман

matrix[0, 0] = 1 + 2 \* ry;

matrix[0, 1] = -2 \* ry;

vector[0] = vector[0] + 2 \* ry \* dy \* bc.value;

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("y0"))

{

throw new ArgumentException("Robin coefficients are missing for y0 boundary.");

}

var robin = robinCoefficients["y0"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[0, 0] = alpha \* (1 + 2 \* ry) + beta \* (2 \* ry / dy);

matrix[0, 1] = -alpha \* (2 \* ry);

vector[0] = alpha \* vector[0] + 2 \* beta \* ry \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for y0 is missing. Setting to zero gradient");

matrix[0, 0] = 1 + 2 \* ry;

matrix[0, 1] = -2 \* ry;

vector[0] = vector[0];

}

// Граница Y1

if (boundaryConditions.ContainsKey("y1"))

{

var bc = boundaryConditions["y1"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(ny - 1, Vector<double>.Build.Dense(ny));

matrix[ny - 1, ny - 1] = 1;

vector[ny - 1] = bc.value;

break;

case 1: // Нейман

matrix[ny - 1, ny - 1] = 1 + 2 \* ry;

matrix[ny - 1, ny - 2] = -2 \* ry;

vector[ny - 1] = vector[ny - 1] - 2 \* ry \* dy \* bc.value;

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("y1"))

{

throw new ArgumentException("Robin coefficients are missing for y1 boundary.");

}

var robin = robinCoefficients["y1"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[ny - 1, ny - 1] = alpha \* (1 + 2 \* ry) + beta \* (2 \* ry / dy);

matrix[ny - 1, ny - 2] = -alpha \* (2 \* ry);

vector[ny - 1] = alpha \* vector[ny - 1] + 2 \* beta \* ry \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for y1 is missing. Setting to zero gradient");

matrix[ny - 1, ny - 1] = 1 + 2 \* ry;

matrix[ny - 1, ny - 2] = -2 \* ry;

vector[ny - 1] = vector[ny - 1];

}

}

public static void BoundaryConditionsZ(Matrix<double> matrix, Vector<double> vector, double rz, int nz, double dz, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

// Граница Z0

if (boundaryConditions.ContainsKey("z0"))

{

var bc = boundaryConditions["z0"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(0, Vector<double>.Build.Dense(nz));

matrix[0, 0] = 1;

vector[0] = bc.value;

break;

case 1: // Нейман

matrix[0, 0] = 1 + 2 \* rz;

matrix[0, 1] = -2 \* rz;

vector[0] = vector[0] + 2 \* rz \* dz \* bc.value;

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("z0"))

{

throw new ArgumentException("Robin coefficients are missing for z0 boundary.");

}

var robin = robinCoefficients["z0"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[0, 0] = alpha \* (1 + 2 \* rz) + beta \* (2 \* rz / dz);

matrix[0, 1] = -alpha \* (2 \* rz);

vector[0] = alpha \* vector[0] + 2 \* beta \* rz \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for z0 is missing. Setting to zero gradient");

matrix[0, 0] = 1 + 2 \* rz;

matrix[0, 1] = -2 \* rz;

vector[0] = vector[0];

}

// Граница Z1

if (boundaryConditions.ContainsKey("z1"))

{

var bc = boundaryConditions["z1"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(nz - 1, Vector<double>.Build.Dense(nz));

matrix[nz - 1, nz - 1] = 1;

vector[nz - 1] = bc.value;

break;

case 1: // Нейман

matrix[nz - 1, nz - 1] = 1 + 2 \* rz;

matrix[nz - 1, nz - 2] = -2 \* rz;

vector[nz - 1] = vector[nz - 1] - 2 \* rz \* dz \* bc.value;

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("z1"))

{

throw new ArgumentException("Robin coefficients are missing for z1 boundary.");

}

var robin = robinCoefficients["z1"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[nz - 1, nz - 1] = alpha \* (1 + 2 \* rz) + beta \* (2 \* rz / dz);

matrix[nz - 1, nz - 2] = -alpha \* (2 \* rz);

vector[nz - 1] = alpha \* vector[nz - 1] + 2 \* beta \* rz \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for z1 is missing. Setting to zero gradient");

matrix[nz - 1, nz - 1] = 1 + 2 \* rz;

matrix[nz - 1, nz - 2] = -2 \* rz;

vector[nz - 1] = vector[nz - 1];

}

}

}

}

## Листинг 5 – class Result

namespace ClassLibrary

{

public class Results

{

public static void WriteToFile(double[,,] solution)

{

string filename = "Result.txt";

try

{

using (StreamWriter writer = new StreamWriter(filename))

{

int nx = solution.GetLength(0);

int ny = solution.GetLength(1);

int nz = solution.GetLength(2);

for (int i = 0; i < nx; i++)

{

for (int j = 0; j < ny; j++)

{

for (int k = 0; k < nz; k++)

{

writer.WriteLine($"{i} {j} {k} {solution[i, j, k]}"); // Сохраняем координаты и температуру

}

}

}

}

Console.Clear();

Console.WriteLine($"Решение записано в {filename}");

}

catch (Exception e)

{

Console.Clear();

Console.WriteLine($"Error writing to file: {e.Message}");

}

}

public static void PrintSlise(double[,,] solution)

{

while (true)

{

Console.Clear();

Console.WriteLine("\n1) Выберите ось:\nX\nY\nZ\n\nQ - Назад");

string axis = Console.ReadLine();

if (axis == "q" || axis == "Q") break;

int size = axis switch

{

"x" or "X" => solution.GetLength(0),

"y" or "Y" => solution.GetLength(1),

"z" or "Z" => solution.GetLength(2),

\_ => throw new ArgumentException("Недопустимая ось. Используйте 'x', 'y' или 'z'")

};

Console.Clear();

Console.WriteLine($"\nПослойные срезы по оси {axis.ToUpper()}:");

Thread.Sleep(1000);

Console.WriteLine("(Для навигации используйте стрелки, для выхода - Esc)\n");

Thread.Sleep(2000);

int currentSlice = 0;

ConsoleKey key;

while(true)

{

Console.Clear();

PrintSingleSlice(solution, axis, currentSlice);

Console.WriteLine($"\nСрез {axis.ToUpper()} = {currentSlice} (Всего: {size - 1})");

Console.WriteLine("<- -> - листать слои | | Esc - выход");

key = Console.ReadKey(true).Key;

if (key == ConsoleKey.Escape)

break;

currentSlice = key switch

{

ConsoleKey.LeftArrow => currentSlice == 0 ? currentSlice : currentSlice - 1,

ConsoleKey.RightArrow => currentSlice == size - 1 ? currentSlice : currentSlice + 1

};

}

}

}

private static void PrintSingleSlice(double[,,] solution, string axis, int slice)

{

switch (axis.ToLower())

{

case "x":

PrintSliceYZ(solution, slice);

break;

case "y":

PrintSliceXZ(solution, slice);

break;

case "z":

PrintSliceXY(solution, slice);

break;

}

}

private static void PrintSliceXY(double[,,] solution, int z)

{

Console.WriteLine($"┌{new string('─', solution.GetLength(1) \* 10 + 1)}┐");

Console.Write("│ X→ ");

for (int x = 0; x < solution.GetLength(0); x++) Console.Write($"{x,8}");

Console.WriteLine(" │");

Console.WriteLine($"├{new string('─', solution.GetLength(1) \* 10 + 1)}┤");

for (int y = 0; y < solution.GetLength(1); y++)

{

Console.Write($"│ Y{y,2} │");

for (int x = 0; x < solution.GetLength(0); x++)

{

Console.Write($"{solution[x, y, z],8:F2}");

}

Console.WriteLine(" │");

}

Console.WriteLine($"└{new string('─', solution.GetLength(1) \* 10 + 1)}┘");

}

private static void PrintSliceXZ(double[,,] solution, int y)

{

Console.WriteLine($"┌{new string('─', solution.GetLength(0) \* 10 + 1)}┐");

Console.Write("│ X→ ");

for (int x = 0; x < solution.GetLength(0); x++) Console.Write($"{x,8}");

Console.WriteLine(" │");

Console.WriteLine($"├{new string('─', solution.GetLength(0) \* 10 + 1)}┤");

for (int z = 0; z < solution.GetLength(2); z++)

{

Console.Write($"│ Z{z,2} │");

for (int x = 0; x < solution.GetLength(0); x++)

{

Console.Write($"{solution[x, y, z],8:F2}");

}

Console.WriteLine(" │");

}

Console.WriteLine($"└{new string('─', solution.GetLength(0) \* 10 + 1)}┘");

}

private static void PrintSliceYZ(double[,,] solution, int x)

{

Console.WriteLine($"┌{new string('─', solution.GetLength(1) \* 10 + 1)}┐");

Console.Write("│ Y→ ");

for (int y = 0; y < solution.GetLength(1); y++) Console.Write($"{y,8}");

Console.WriteLine(" │");

Console.WriteLine($"├{new string('─', solution.GetLength(1) \* 10 + 1)}┤");

for (int z = 0; z < solution.GetLength(2); z++)

{

Console.Write($"│ Z{z,2} │");

for (int y = 0; y < solution.GetLength(1); y++)

{

Console.Write($"{solution[x, y, z],8:F2}");

}

Console.WriteLine(" │");

}

Console.WriteLine($"└{new string('─', solution.GetLength(1) \* 10 + 1)}┘");

}

}

}

# 5. Пример работы программы

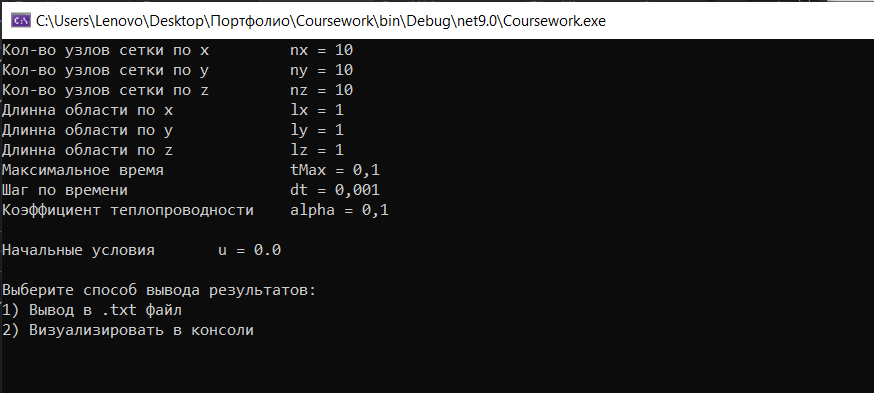


Рисунок 1 – Вывод начальных условий.

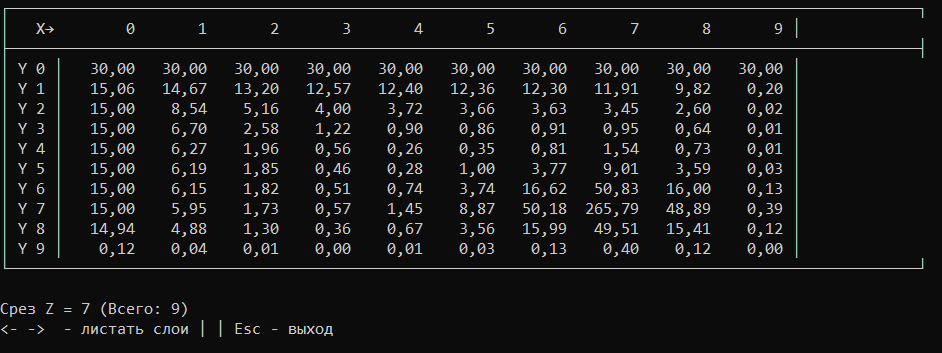


Рисунок 2 – Срез по Z = 7.

По данному графику видно точку нагрева в координатах (7, 7, 7) и нагрев прилежащих к ней узлов.

Графики нагрева линии узлов с фиксированными координатами

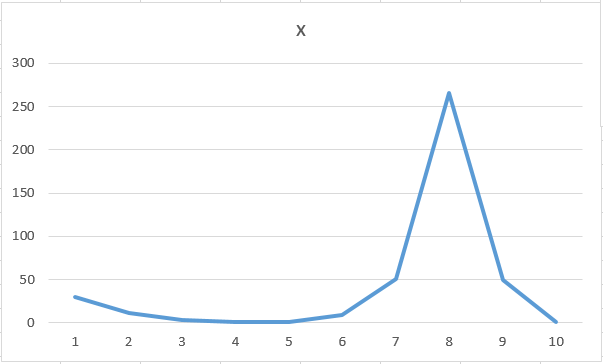


Рисунок 3 – График температуры по x с фиксированными (y = 7, z = 7).

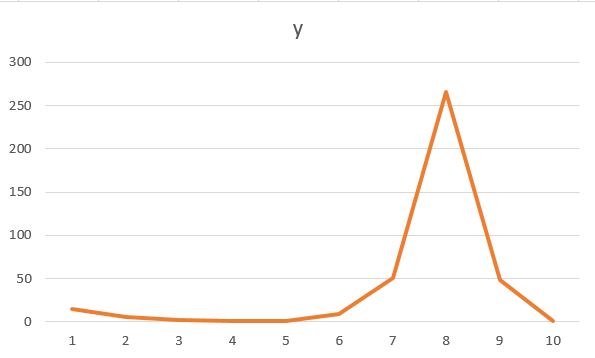


Рисунок 4 – График температуры по y с фиксированными (x = 7, z = 7).

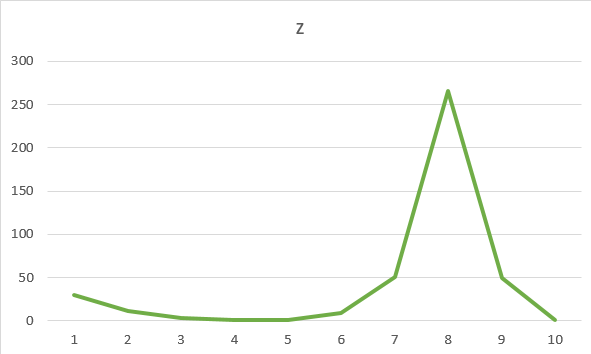


Рисунок 5 – График температуры по z с фиксированными (y = 7, x = 7).

# 6. Заключение

Программа эффективно решает трёхмерное уравнение теплопроводности методом расщепления с поддержкой:

* Различных граничных условий (Дирихле, Неймана, Робина).
* Точечных и распределённых источников тепла.
* Гибкой визуализации результатов.

Код структурирован и легко расширяется для более сложных задач (например, нелинейных уравнений).

# 7. Список литературы

1.Самарский А.А., Вабищевич П.Н. Численные методы решения задач теплопроводности и диффузии.

2.Тихонов А.Н., Самарский А.А. Уравнения математической физики.