ФГАОУ ВО «Санкт-Петербургский Политехнический университет Петра Великого»

Физико-Механический институт

Высшая школа теоретической механики

Отчет по индивидуальному заданию

Решение уравнения теплопроводности методом конечных разностей. Вариант 27

Студент:

Чеботин А.А.
группа: 5030103/90301

Преподаватель:

Витохин Е.Ю

Санкт-Петербург, 2021 г.

Содержание

Глава 1

1.1. Формулировка задачи.

Требуется получить конечно-разностное решение первой начально-краевой задачи для уравнения теплопроводности

$$\rho c_v \frac{\partial T}{\partial t} = \lambda \frac{\partial^2 T}{\partial x^2}$$

$$x \in [0; l],\ t \in [0; t^*],\ l = 0.6,\ t^* = 0.01$$

Обозначения:

- λ - коэффициент теплопроводности
- c_v - теплоемкость среды
- ρ - плотность вещества
- T - температура

Дано начальное условие $T(x, 0) = T^*(x) = 0.9 + 2x(1 - x)$ и граничные условия

$$T(0, t) = T_0(t) = 3(0.3 - 2t)$$

$$T(l, t) = T(0.6, t) = T_l(t) = 1.38$$

Будем анализировать теплопередачу с параметрами $\lambda = 1$, $c_v = 1$, $\rho = 1$.

1.2. Описание численного метода

1. **Дискретизация**. Область непрерывного изменения значения аргументов x и t разбивается на конечное число интервалов, то есть на пространственно-временную область $\Omega \times T = (0 \le x \le l;\ 0 \le t \le t^*)$ наносится конечно-разностная сетка

$$x_i = ih,\ x_0 = 0,\ x_n = l = 0.6,\ i = 0, 1, ..., n\ (n = \frac{l}{h} = 6)$$

$$t^k = k\Delta t,\ t^0 = 0,\ t^k = t^* = 0.001,\ k = 0, 1, ..., K\ (K = \frac{t^*}{\Delta t})$$

где k - номер слоя по времени, Δt - шаг интегрирования по времени, i - номер узла по пространству, $h = 0.1$ - пространственный шаг.

Также вводятся два *временных слоя. Нижний* $t^k = k\Delta t$, на котором распределение искомой функции $T(x_i, t^k)$ известно, и *верхний* $t^{k+1} = (k + 1)\Delta t$, на котором распределение функции $T(x_i, t^{k+1})$ нужно вычислить. Введем определение:

Определение 1.1. *Сеточной функцией* T_i^k назовем функцию, определенную только в дискретном множестве точек (в узлах) разностной сетки.

Таким образом, мы на сетке введем две сеточные функции, первая T_i^k из которых известна, а вторую T_i^{k+1} нужно найти методом конечных разностей.

2. **Аппроксимация**. Идея метода конечных разностей: вместо производных мы используем их конечно-разностные аналоги.

   (a) **Явная схема**. Для определения T_i^{k+1} аппроксимируем дифференциалы уравнения теплопроводности:

   $$\frac{\partial T}{\partial t} = \frac{T_i^{k+1} - T_i^k}{\Delta t}$$

   $$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i-1}^k - 2T_i^k + T_{i+1}^k}{h^2}$$

   Подставляя эти аппроксимации в уравнение теплопроводности, получим:

   $$\frac{\rho c_v}{\Delta t}(T_i^{k+1} - T_i^k) = \frac{\lambda}{h^2}(T_{i+1}^k - 2T_i^k + T_{i-1}^k)$$

   В полученном разностном уравнении неизвестны только T_i^{k+1}, которые мы можем определить *явно* из разностного уравнения. Таким образом, явное выражение будет выглядеть следующим образом (и которое будем использовать в нашей программе):

   $$T_i^{k+1} = \frac{h^2 c_v T_i^k + \Delta t\lambda(T_{i+1}^k - 2T_i^k + T_{i-1}^k)}{\rho c_v h^2}$$

   (b) **Неявная схема**. Используя неявную схему, производные будут заменяться следующими выражениями

$$\frac{\partial T}{\partial t} = \frac{T_i^{k+1} - T_i^k}{\Delta t}$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i-1}^{k+1} - 2T_i^{k+1} - T_{i+1}^{k+1}}{h^2}$$

То есть мы дифференциал по x аппроксимировали соотношением на верхнем временном слое, в отличие от аппроксимации по явной схеме. После подставления данных аппроксимаций, приводим подобные слагаемые и получаем следующее разностное уравнение:

$$-AT_{i-1}^{k+1} + BT_i^{k+1} - CT_{i+1}^{k+1} = F_i,$$

$$A = \frac{\lambda}{h^2},\ B = \frac{\rho c_v h^2 + 2\lambda\Delta t}{\Delta t h^2},\ C = \frac{\lambda}{h^2},\ F_i = \frac{\rho c_v}{\Delta t} T_i^k$$

Разностное уравнение есть СЛАУ с трехдиагональной матрицей, поэтому в случае неявной схемы нужно перейти к ее решению (этап 3).

3. **Решение СЛАУ**. В результате аппроксимации мы свели исходную задачу к системе алгебраических уравнений. Образованная этими уравнениями СЛАУ имеет трехдиагональную матрицу, поэтому будем применять метод прогонки, алгоритм которого имеет сложность $O(n)$.

   СЛАУ МКР будет иметь следующую структуру:

$$\underbrace{\begin{pmatrix} B & -C & 0 & 0 & ... & 0 \\ -A & B & -C & 0 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & ... & 0 & -A & B & -C \\ 0 & ... & 0 & 0 & -A & B \end{pmatrix}}_{Matrix} \cdot \begin{pmatrix} T_1^{k+1} \\ T_2^{k+1} \\ \vdots \\ T_{n-1}^{k+1} \\ T_n^{k+1} \end{pmatrix} = \begin{pmatrix} F_1^{k+1} \\ F_2^{k+1} \\ \vdots \\ F_{n-1}^{k+1} \\ F_n^{k+1} \end{pmatrix}$$

   Алгоритм метода прогонки. Мы должны найти прогоночные коэффициенты.

   (a) Прямой ход.

$$i = 1:\ P_1 = \frac{C}{B},\ Q_1 = \frac{F_1}{B}$$

$$i = 2, ..., n:\ P_i = \frac{C}{B - AP_{i-1}},\ Q_i = \frac{F_i + AQ_{i-1}}{B - AP_{i-1}}$$

   (b) Обратный ход. Находим искомое решение, используя найденные прогоночные коэффициенты

$$T_i^{k+1} = P_i T_{i+1}^{k+1} + Q_i$$

1.3. Подготовка контрольных тестов

Для исследования метода будем проводить следующие тесты:

1. Построим поверхности явной и неявной схем;
2. Построим разрезы, зафиксировав определенный момент времени;
3. Сравним схемы численно.

1.4. Модульная структура программы

В коде имеются различные пользовательские типы:

- enum ListOfSubstances - перечисление, состоящее из набора целочисленных констант. В нашем случае перечислителями будут вещества: (StainlessSteel=
- сталь, Iron=1 - железо, Gold=2 - золото, Brass=3 - латунь, Copper=4 - медь, Mercury=5 - ртуть, Silver=6 - серебро, Diamond=7 - алмаз, Glass=8 - стекло, Vacuum=9 - вакуум, Ones=10 - по умолчанию);
- struct MaterialsData - простейшая структура, имеющая три элемента: (const double cvHeatCapacity - теплоемкость, const double lambdaThermalConductivity - коэффициент теплопроводности, const double rhoDensity - плотность);
- namespace MaterialsDataBase - пространство имен. Здесь используется контейнер vector из стандартной библиотеки C++. Мы используемся вектор структур MaterialsData и при инициализации задаем значения для всех элементов структуры MaterialsData;
- struct IntegrateParameters - простейшая структура, содержащая два элемента - шаг интегрирования по времени и по координате x;
- class HeatEquationSolver - основной класс программы, решающий уравнение теплопроводности. Класс имеет свои функции. Для передачи структур в функции класса HeatEquationSolver используется константная ссылка, чтобы повысить эффективность алгоритма и избавиться от дополнительных затрат. Обычные переменные будут передаваться по значению, не затрачивая при этом расходы на дополнительный доступ в функции класса.

Скриншоты программы:

HeatEquation.cpp

Mechanics_testlab1 | (Глобальная область) | main()

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#include <vector>
#include <array>
#include <stdint.h>

#define N 7
#define K 11

/*just enumeration of various materials*/
enum ListOfSubstances {
    StainlessSteel = 0,
    Iron = 1,
    Gold = 2,
    Brass = 3,
    Copper = 4,
    Mercury = 5,
    Silver = 6,
    Diamond = 7,
    Glass = 8,
    Vacuum = 9,
    Ones = 10,
    Wood = 11
};

/*struct consists of material parameters*/
struct MaterialsData {
    const double cv_HeatCapacity;
    const double lambda_ThermalConductivity;
    const double rho_Density;
};

/*prepared material parameters in container for materials from ListOfSubstances*/
namespace MaterialDataBase {
    std::vector<MaterialsData> materials = {
        {490, 15, 7800},
        {452, 92, 7870},
        {128.7, 320, 19300},
        {378, 100, 8600},
        {383, 401, 8900 },
        {139, 8.3, 13540},
        {235, 430, 10500},
        {502, 1700, 3500},
        {500, 1, 25},
        {0, 0, 0},
        {1, 1, 1},
        {2300, 110, 470 }
    };
}

/*integrate steps*/
struct IntegrateParameters {
    double delta_tStep;
    double h_xStep;
};

class HeatEquationSolver {
    //HeatEquationSolver(); //class constructor
    //~HeatEquationSolver() {}; //destructor

private:

    inline double InitialValueFunction(double x);
    inline double LeftBoundaryConditionFunction(double t);
    inline double RightBoundaryConditionFunction(double t);

public:

    void SetCoordinateXGrid(std::array<double, N>& x, const IntegrateParameters& params);
    void SetTimeGrid(std::array<double, K>& t, const IntegrateParameters& params);

    void getSolutionExplicit(const MaterialsData& material, double(*Temperature)[N], std::array<double, N>& x, std::array<dou
        const IntegrateParameters& params);
    void getSolutionImplicit(const MaterialsData& material, double(*Temperature)[N], std::array<double, N>& x, std::array<dou
        const IntegrateParameters& params);
};
```

```
80
81  /* NU T(x,0) */
82  inline double HeatEquationSolver::InitialValueFunction(double x)
83  {
84      return 0.9 + 2 * x * (1 - x);
85  }
86
87  /* T(0, t) */
88  inline double HeatEquationSolver::LeftBoundaryConditionFunction(double t)
89  {
90      return 3 * (0.3 - 2 * t);
91  }
92
93  /* GU2 T(l,t) */
94  inline double HeatEquationSolver::RightBoundaryConditionFunction(double t)
95  {
96      return 1.38;
97  }
98
99  /*creating discrete grids*/
100 void HeatEquationSolver::SetCoordinateXGrid(std::array<double, N>& x, const IntegrateParameters& params)
101 {
102     x[0] = 0;
103     for (uint8_t i = 1; i < N; ++i) {
104         x[i] = i * params.h_xStep;
105     }
106 }
107
108 void HeatEquationSolver::SetTimeGrid(std::array<double, K>& t, const IntegrateParameters& params)
109 {
110     t[0] = 0;
111     for (uint8_t k = 1; k < K; ++k) {
112         t[k] = k * params.delta_tStep;
113     }
114 }
115
116 /*solution by explicit finite difference method*/
117 void HeatEquationSolver::getSolutionExplicit(const MaterialsData& material, double (*Temperature)[N], std::array<double, N>&
118     std::array<double, K>& t, const IntegrateParameters& params)
119 {
120     /*setting initial condition*/
121     for (uint8_t i = 0; i < N; ++i)
122     {
123         Temperature[0][i] = InitialValueFunction(x[i]);
124     }
125
126     /*setting boundary conditions*/
127     for (uint8_t k = 0; k < K; ++k)
128     {
129         Temperature[k][0] = LeftBoundaryConditionFunction(t[k]);
130         Temperature[k][N-1] = RightBoundaryConditionFunction(t[k]);
131     }
132
133     /*to be short*/
134     double c_v = material.cv_HeatCapacity;
135     double lambda = material.lambda_ThermalConductivity;
136     double rho = material.rho_Density;
137
138     double h = params.h_xStep;
139     double delta_t = params.delta_tStep;
140
141     /*explicit formule*/
142     for (uint8_t k = 0; k < K-1; ++k)
143     {
144         for (uint8_t i = 1; i < N - 1; ++i)
145         {
146             Temperature[k + 1][i] = (h * h * c_v * Temperature[k][i] + delta_t * lambda *
147                 (Temperature[k][i + 1] - 2 * Temperature[k][i] + Temperature[k][i - 1])) / (rho * c_v * h * h);
148         }
149     }
150
151 }
152
153 void HeatEquationSolver::getSolutionImplicit(const MaterialsData& material, double(*Temperature)[N], std::array<double, N>&
154     std::array<double, K>& t, const IntegrateParameters& params)
155 {
156     /*setting initial condition*/
157     for (uint8_t i = 0; i < N; ++i)
158     {
159         Temperature[0][i] = InitialValueFunction(x[i]);
160     }
```

```
161
162     /*setting boundary conditions*/
163     for (uint8_t k = 0; k < K; ++k)
164     {
165         Temperature[k][0] = LeftBoundaryConditionFunction(t[k]);
166         Temperature[k][N-1] = RightBoundaryConditionFunction(t[k]);
167     }
168
169     double c_v = material.cv_HeatCapacity;
170     double lambda = material.lambda_ThermalConductivity;
171     double rho = material.rho_Density;
172
173     double h = params.h_xStep;
174     double delta_t = params.delta_tStep;
175
176     /*setting coefficients of finite difference equation*/
177     double A = (lambda) / (h * h);
178     double C = (lambda) / (h * h);
179     double B = (rho * c_v * h * h + 2 * lambda * delta_t) / (delta_t * h * h);
180     double F[N], P[N], Q[N];
181
182     for (uint8_t k = 0; k < K - 1; ++k)
183     {
184
185         for (uint8_t i = 0; i < N; i++)
186         {
187             F[i] = Temperature[k][i] * ((rho * c_v) / (delta_t));
188         }
189         /*Progonka algorithm coefficients*/
190         P[0] = C / B;
191         Q[0] = F[0] / B;
192
193         for (uint8_t j = 1; j < N; ++j)
194         {
195             P[j] = C / (B - A * P[j - 1]);
196             Q[j] = (F[j] + A * Q[j - 1]) / (B - A * P[j - 1]);
197         }
198
199         /*back substitution*/
200         for (uint8_t j = N - 2; j > 0; --j)
201         {
202             Temperature[k + 1][j] = P[j] * Temperature[k + 1][j + 1] + Q[j];
203         }
204     }
205
206
207 }
208
209
210 int main()
211 {
212     using namespace MaterialDataBase;
213
214     HeatEquationSolver mySolver;
215     IntegrateParameters params;
216
217     params.h_xStep = 0.1;
218     params.delta_tStep = 0.001;
219
220     double Temperature[K][N] = { 0 };
221     std::array<double, N> x;
222     std::array<double, K> t;
223
224     mySolver.SetCoordinateXGrid(x, params);
225     mySolver.SetTimeGrid(t, params);
226
227     /*Здесь можно задать материал (см. enum)*/
228     mySolver.getSolutionImplicit(materials[Ones], Temperature, x, t, params);
229
```

```
229
230     for (uint8_t i = 0; i < N; i++)
231         printf("%lf ", x[i]);
232     printf("\n\n");
233     for (uint8_t k = 0; k < K; k++)
234         printf("%lf ", t[k]);
235     printf("\n\n");
236
237     for (uint8_t i = 0; i < K; ++i)
238     {
239         for (uint8_t j = 0; j < N; ++j)
240             printf("%lf\t", Temperature[i][j]);
241         printf("\n\n");
242     }
243
244     return 0;
245 }
```

```
Вывод
Показать выходные данные из: Отладка
 "Mechanics_testlab1.exe" (Win32). Загружено "C:\Windows\SysWOW64\msvcrt.dll".
 "Mechanics_testlab1.exe" (Win32). Загружено "C:\Windows\SysWOW64\rpcrt4.dll".
 Поток 0x5810 завершился с кодом 0 (0x0).
 Поток 0x42cc завершился с кодом 0 (0x0).
 Программа "[18420] Mechanics_testlab1.exe" завершилась с кодом 0 (0x0).
Developer PowerShell | Список ошибок | Вывод
```

1.5. Численный анализ решения задачи

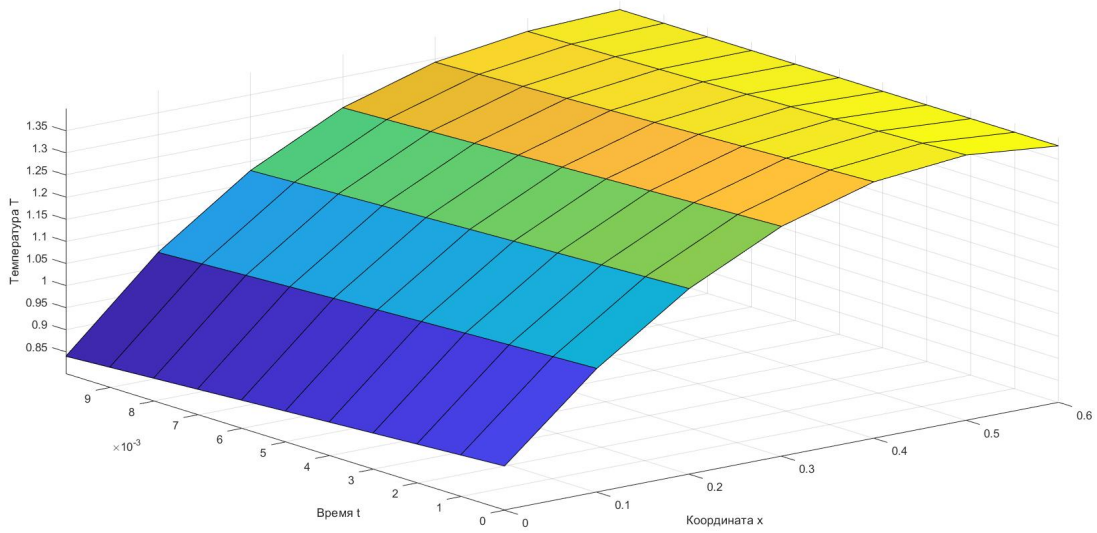

Рис 1.1 Поверхность (явная схема)

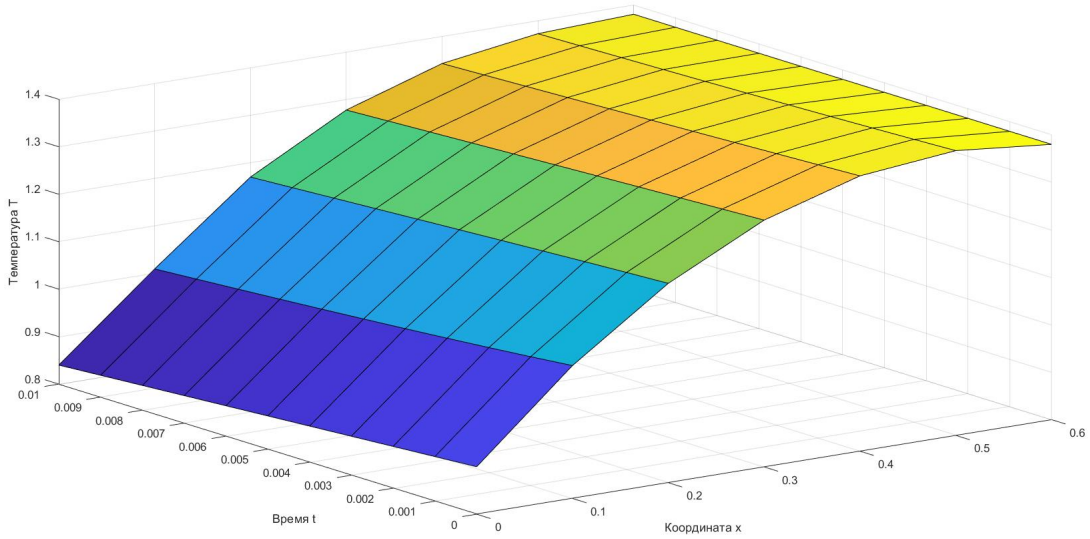

Рис 1.2. Поверхность (неявная схема)

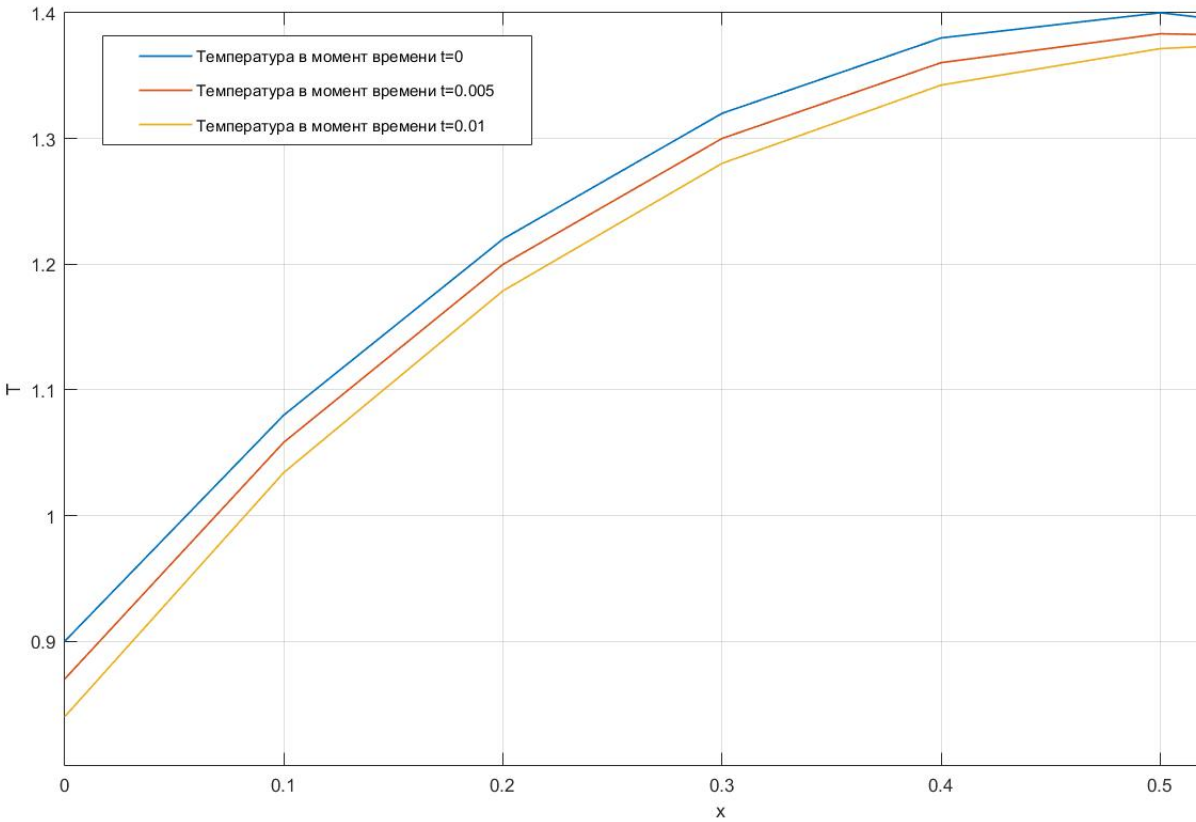

Рис 2.1. Распределение температуры в разные моменты времени (явная схема)

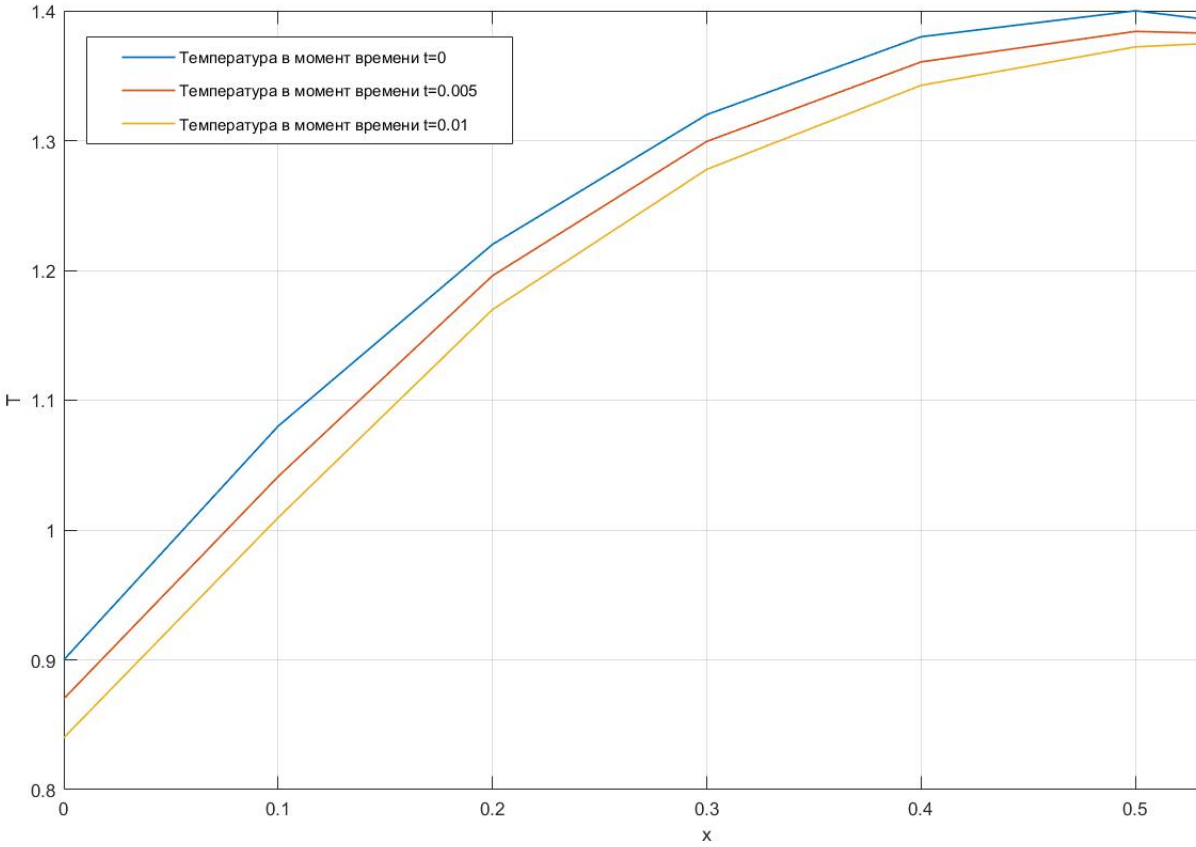

Рис 2.2. Распределение температуры в разные моменты времени (неявная схема)

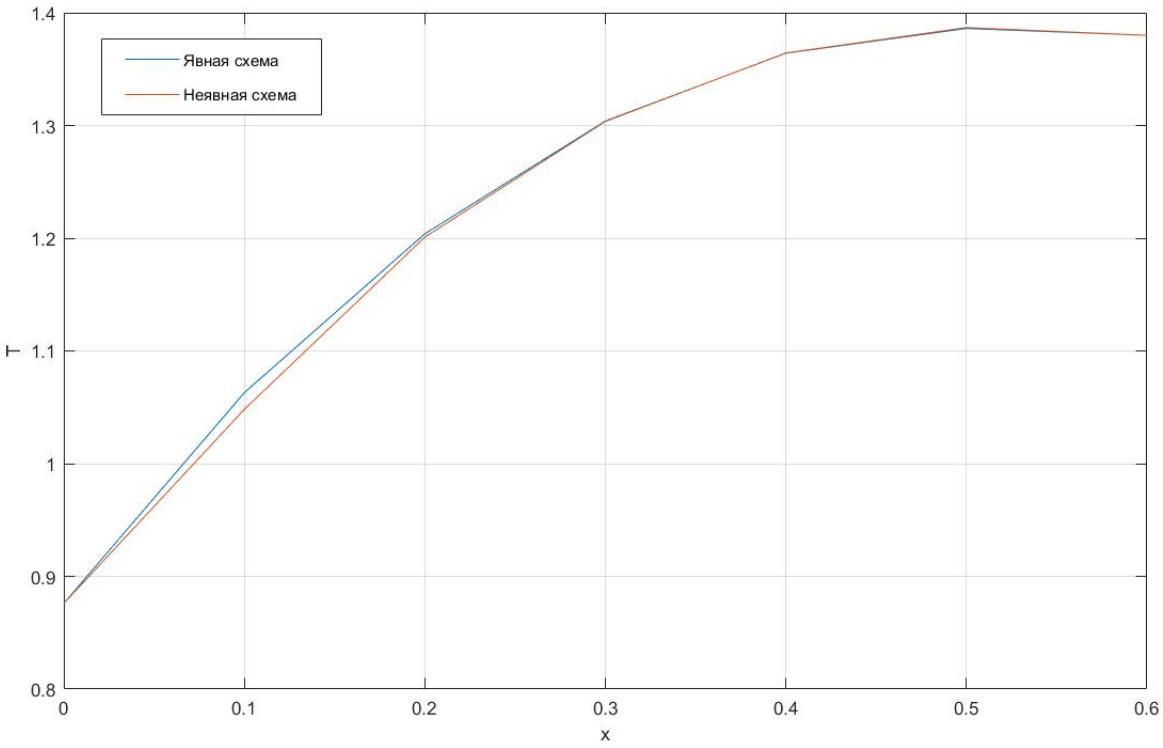

Рис 3. Сравнение схем в момент времени $t = 0.4$.

Сравним численно:

Таблица 1. Явная схема

	$x_0 = 0$	$x_1 = 0.1$	$x_2 = 0.2$	$x_3 = 0.3$	$x_4 = 0.4$	$x_5 = 0.5$	$x_6 = 0.6$
$t_0 = 0$	0.900000	1.080000	1.220000	1.320000	1.380000	1.400000	1.380000
$t_1 = 0.001$	0.894000	1.076000	1.216000	1.316000	1.376000	1.396000	1.380000
$t_2 = 0.002$	0.888000	1.071800	1.212000	1.312000	1.372000	1.392400	1.380000
$t_3 = 0.003$	0.882000	1.067440	1.207980	1.308000	1.368040	1.389120	1.380000
$t_4 = 0.004$	0.876000	1.062950	1.203928	1.304002	1.364144	1.386100	1.380000
$t_5 = 0.005$	0.870000	1.058353	1.199838	1.300009	1.360325	1.383294	1.380000
$t_6 = 0.006$	0.864000	1.053666	1.195706	1.296023	1.356591	1.380668	1.380000
$t_7 = 0.007$	0.858000	1.048903	1.191534	1.292048	1.352942	1.378194	1.380000
$t_8 = 0.008$	0.852000	1.044076	1.187322	1.288086	1.349378	1.375849	1.380000
$t_9 = 0.009$	0.846000	1.039193	1.183074	1.284139	1.345896	1.373617	1.380000
$t_{10} = 0.01$	0.840000	1.034262	1.178792	1.280208	1.342492	1.371483	1.380000

Таблица 2. Неявная схема

	$x_0 = 0$	$x_1 = 0.1$	$x_2 = 0.2$	$x_3 = 0.3$	$x_4 = 0.4$	$x_5 = 0.5$	$x_6 = 0.6$
$t_0 = 0$	0.900000	1.080000	1.220000	1.320000	1.380000	1.400000	1.380000
$t_1 = 0.001$	0.894000	1.071239	1.215601	1.315969	1.376025	1.396335	1.380000
$t_2 = 0.002$	0.888000	1.063075	1.210914	1.311891	1.372092	1.392954	1.380000
$t_3 = 0.003$	0.882000	1.055393	1.206025	1.307762	1.368207	1.389812	1.380000
$t_4 = 0.004$	0.876000	1.048106	1.200995	1.303583	1.364378	1.386875	1.380000
$t_5 = 0.005$	0.870000	1.041141	1.195871	1.299359	1.360604	1.384113	1.380000
$t_6 = 0.006$	0.864000	1.034442	1.190687	1.295097	1.356886	1.381501	1.380000
$t_7 = 0.007$	0.858000	1.027963	1.185470	1.290805	1.353224	1.379020	1.380000
$t_8 = 0.008$	0.852000	1.021667	1.180238	1.286492	1.349615	1.376651	1.380000
$t_9 = 0.009$	0.846000	1.015525	1.175006	1.282165	1.346058	1.374381	1.380000
$t_{10} = 0.01$	0.840000	1.009513	1.169784	1.277832	1.342551	1.372196	1.380000

1.6. Выводы

Был исследован метод конечных разностей. Метод достаточно прост в реализации. Была получена поверхность с распределением температуры по явной и неявной схеме. Проведено графическое сравнение схем.