РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Моделирование физических процессов с применением разностных схем

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2021 г.

Министерство науки и высшегообразования 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)

УТВЕРЖДАЮ
Заведующий кафедрой _____
_____ _____
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____Вычислительная физика_____

Студент группы ___ФН4-81Б__

_____Ярков Андрей Владимирович_____
(Фамилия, имя, отчество)

Тема курсовой работы ___Моделирование физических процессов с применением_____
разностных схем_____
_____
_____

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
_____Учебная_____
Источник тематики (кафедра, предприятие, НИР) _____кафедра__

График выполнения работы: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание_____С использованием разностных схем получить численное решение_____
уравнения теплопроводности в поставленной задаче_____
_____
_____
_____

Оформление курсовой работы:

Расчетно-пояснительная записка на _____ листах формата А4.

_____
_____
_____
_____

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсовой работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Оглавление

Введение

Математическое моделирование физических процессов получает всё большее распространение в современном мире. Этому способствует как усложнение проводимых физических экспериментов, так и совершенствование используемых вычислительных средств и методов. Сложность физических экспериментов порой вовсе ставит крест на некоторые стороны изучаемых явлений из-за дороговизны эксперимента или отсутствия способов и средств необходимого исследования. В свою очередь эту ситуацию сглаживает активное сопоставление результатов эксперимента с результатами моделирования, что позволяет смотреть на недоступные стороны физического процесса посредством математического моделирования.

В частности, с помощью математического моделирования при использовании разностных схем может быть получено решение уравнения теплопроводности для систем, конфигурация которых весьма сложна. Для корректности получаемых результатов, во-первых, необходимо подобрать соответствующую схему, которая не только даст корректное решение, но и будет аппроксимировать рассматриваемую дифференциальную задачу, то есть будет сходящейся. Во-вторых, необходимо постоянно проверять “физичность” получаемых результатов, чтобы избежать неправильной интерпретации результатов.

Теоретическая часть

1. Уравнение теплопроводности

Уравнение теплопроводности представляет собой дифференциальное уравнение в частных производных, описывающее распределение температуры в заданной области и его изменение со временем. Для его получения необходимо записать уравнение баланса тепла в рассматриваемой системе.

Наибольшей простотой обладает одномерный случай в декартовой системе координат. Так, изменение количества тепла в области $[x_1, x_2]$ реализуется за счёт потока через эту область за промежуток времени $[t_1, t_2]$ и выделения тепла за счёт источников, распределённых в этой области:

$$Q(t_2) - Q(t_1) = j(x_2) - j(x_1) + F, \quad (1)$$

или, если записать представленные величины через интегралы:

$$\int_{x_1}^{x_2} C[u(x,t_2) - u(x,t_1)]dx = \int_{t_1}^{t_2} [w(x_1,t) - w(x_2,t)]dt + \int_{x_1}^{x_2}\int_{t_1}^{t_2} f(x,t)dxdt\,, \quad (2)$$

где $C = c\rho$ – теплоёмкость среды (c – теплоёмкость единицы массы, ρ – плотность среды, в общем случае зависят от x, t, u), $u(x,t)$ – температура среды в точке x в момент времени t, $w = -\lambda\frac{\partial u}{\partial x}$ – плотность потока тепла, $\lambda = \lambda(x,t)$ – коэффициент теплопроводности среды, $f(x,t)$ – плотность распределения источников тепла.

Если предположить, что рассматриваемая область имеет физически бесконечно малый объём, то есть $x_2 = x_1 + dx$, то уравнение (2) может быть записано в дифференциальном виде:

$$c\rho\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\lambda\frac{\partial u}{\partial x}\right) + f(x,t). \quad (3)$$

Уравнение (3) имеет единственное решение, если заданы непротиворечащие друг другу начальные и граничные условия. Так, начальное условие имеет вид:

$$u(x,0) = u_0(x), \quad (4)$$

а в качестве граничных условий для большинства задач подходят граничные условия третьего рода (которые могут быть сведены к граничным условиям первого и второго рода):

$$\left(k_1 \frac{\partial u}{\partial x} - \theta_1 u\right)_{(0,t)} = \mu_1(t), \tag{5}$$

$$\left(k_2 \frac{\partial u}{\partial x} - \theta_2 u\right)_{(l,t)} = \mu_2(t), \tag{6}$$

где l – размер рассматриваемой области, $\theta = \theta(x,t)$ некоторый коэффициент.

Таким образом, полная задача имеет следующий математический вид при объединении уравнений (3)-(6):

$$\begin{cases} c\rho \dfrac{\partial u}{\partial t} = \dfrac{\partial}{\partial x}\left(\lambda \dfrac{\partial u}{\partial x}\right) + f(x,t), \;\; 0 < x < l, \;\; 0 < t \le T; \\ u(x,0) = \psi(x), \;\; 0 \le x \le l; \\ \left(k_1 \dfrac{\partial u}{\partial x} - \theta_1 u\right)_{(0,t)} = \mu_1(t), \;\; 0 \le t \le T; \\ \left(k_2 \dfrac{\partial u}{\partial x} - \theta_2 u\right)_{(l,t)} = \mu_2(t), \;\; 0 \le t \le T, \end{cases}$$

где коэффициенты k_1 и k_2 могут быть равны 0 или 1.

Численное решение данной задачи требует записи устойчивой разностной схемы.

2. Построение разностной схемы для уравнения теплопроводности

Разностная схема – это конечная система алгебраических уравнений, соответствующих конкретной дифференциальной задаче и аппроксимирующая его с определённым порядком точности. Для её построения сначала необходимо разбить область и временной промежуток на конечное число узловых точек. Как правило, прибегают к равномерному разбиению: $t^n = t^0 + n\tau$, $x_m = x_0 + mh$, где $n = \overline{0,N}$, $m = \overline{0,M}$, $\tau = \frac{T}{N}$, $h = \frac{l}{M}$. Тогда точное решение дифференциальной задачи $u(x,t)$ заменяется сеточной функцией u_m^n, аппроксимирующей точное решение.

Если ввести дифференциальный оператор L, определяющий всю дифференциальную часть уравнения (3), соответствующий ему конечно-разностный оператор L_h и оператор проекции на пространство сеточных функций P_h, то говорят, что разностная схема аппроксимирует дифференциальную задачу, если

$$\|L(u) - L_h(P_h u)\| \underset{h\to 0}{\longrightarrow} 0. \quad (7)$$

Однако условия аппроксимации недостаточно для использования разностной схемы. Она должны быть сходящейся, то есть:

$$\|u_h - P_h u\| \underset{h\to 0}{\longrightarrow} 0. \quad (8)$$

Принципиально то, что условие (8) следует из условия (7), если используемая разностная схема корректна, то есть выполняются следующие два условия:

1. Существует единственное решение для любых $P_h f$.
2. Существует такая постоянная R, не зависящая от h, что $\|u_h\| \leq R\|P_h f\|$.

То есть существует обратный оператор, непрерывный по h, позволяющий получить искомое решение задачи при задании конкретной функции-источника.

Но и этого зачастую недостаточно для оправданного использования конкретных разностных схем ввиду того, что в большинстве задач требуется устойчивость разностной схемы, заключающаяся в том, что погрешности решения, появляющиеся по той или иной причине (округление, неточность вычисления), не должны возрастать. Устойчивость схем проверяется подстановкой в них в качестве искомой функции возмущения $\delta u_m^n = \tilde{\lambda}^n e^{i\omega t}$ и анализа на выполнение условия Неймана: $|\tilde{\lambda}| \leq 1$.

Этими качествами обладает следующая неявная разностная схема, аппроксимирующая дифференциальную задачу (3)-(6) порядком $O(h^2 + \tau)$, что является весьма хорошим приближением в случае достаточно малого шага по времени:

$$\begin{cases} \dfrac{u_m^{n+1}-u_m^n}{\tau} = a^2\dfrac{u_{m+1}^{n+1}-2u_m^{n+1}+u_{m-1}^{n+1}}{h^2}+\varphi_m^{n+1}, \qquad m=\overline{1,M-1},\ \ n=\overline{0,N-1} \\ u_m^0=\psi_m,\ \ m=\overline{0,M}, \\ k_1\dfrac{u_1^{n+1}-u_0^{n+1}}{h}-\theta_1^{n+1}u_0^{n+1}-\dfrac{h}{2a^2}\left[\dfrac{u_0^{n+1}-u_0^n}{\tau}-\varphi_0^{n+1}\right]=\mu_1^{n+1}, \qquad n=\overline{0,N-1} \\ k_2\dfrac{u_M^{n+1}-u_{M-1}^{n+1}}{h}-\theta_2^{n+1}u_M^{n+1}+\dfrac{h}{2a^2}\left[\dfrac{u_M^{n+1}-u_M^n}{\tau}-\varphi_M^{n+1}\right]=\mu_2^{n+1}, \qquad n=\overline{0,N-1} \end{cases}$$

где введены новые обозначения $a^2=\frac{\lambda}{c\rho}$ (рассматривается случай постоянного коэффициента теплопроводности), $\varphi=\frac{f}{c\rho}$.

Представленная разностная схема решается методом прогонки, согласно которому основное разностное соотношение переписывается в виде:

$$Au_{m-1}^{n+1}-Cu_m^{n+1}+Bu_{m+1}^{n+1}=-F_m, \tag{9}$$

в котором коэффициенты имеют следующие значения: $A=B=1, \mathrm{C}=2+\frac{h^2}{a^2\tau}$, $F_m=\frac{h^2}{a^2\tau}[u_m^n+\tau\varphi_m^{n+1}]$. Решение ищется в виде рекуррентной формулы:

$$u_m^{n+1}=\alpha_{m+1}u_{m+1}^{n+1}+\beta_{m+1}, \tag{10}$$

в котором прогоночные коэффициенты определяются выражениями:

$$\alpha_{m+1}=\frac{B}{C-A\alpha_m},\ \ \beta_{m+1}=\frac{F_m+A\beta_m}{C-A\alpha_m}. \tag{11}$$

Начальные значения прогоночных коэффициентов определяются с помощью левого граничного условия:

$$\alpha_1=\frac{1}{1+\frac{h^2}{2a^2\tau}+\theta_1^{n+1}h},\ \ \beta_1=\frac{\frac{h^2}{2a^2\tau}[u_0^n+\tau\varphi_0^{n+1}]-h\mu_1^{n+1}}{1+\frac{h^2}{2a^2\tau}+\theta_1^{n+1}h}, k_1=1. \tag{12.1}$$

$$\alpha_1=0,\ \ \beta_1=-\frac{\mu_1^{n+1}}{\theta_1^{n+1}}, k_1=0. \tag{12.2}$$

Второе граничное условие, в свою очередь определяет значение на правой границе рассматриваемой области:

$$u_M^{n+1} = \frac{\beta_M + \frac{h^2}{2a^2\tau}[u_M^n + \tau\varphi_M^{n+1}] + h\mu_2^{n+1}}{1 + \frac{h^2}{2a^2\tau} - \alpha_M + \theta_2^{n+1}h}, k_2 = 1. \quad (13.1)$$

$$u_M^{n+1} = -\frac{\mu_2^{n+1}}{\theta_2^{n+1}}, k_2 = 0. \quad (13.2)$$

Подобная вариативность определения коэффициентов и значения на правой границе вытекает из присутствия или отсутствия производной в граничном условии, что сказывается на виде аппроксимирующего соотношения.

Практическая часть

1. Постановка задачи

Образец стекла размером $40 \times 40 \times 0.17$ мм, прикрипленный к охлаждаемому столику, равномерно облучается в вакууме электронами и протонами с энергиями 15 кэв и 30 кэв соответственно. Рассмотреть раздельное и совместное облучение электронами и протонами. Температура столика (20 ± 1)℃. Построить распределение температуры в стекле при облучении разными плотностями потоков $(10^{10}, 10^{11}, 10^{12}, 10^{13})$ электронов $/(\text{см}^2 \cdot c)$ и $(10^9, 10^{10}10^{11}, 10^{12})$ протонов$/(\text{см}^2 \cdot c)$ соотвественно. Время облучения от 0 до момента наступления установившегося температурного режима. Имеет место только радиационный сброс тепла с облучаемой поверхности стекла.

Распределение плотности поглощаемой энергии по толщине образца заданы в виде таблиц.

Таким образом, если пренебречь влиянием конечности площади поверхности стекла и неоднородностью коэффициента теплопроводности, то рассматриваемая задача является одномерной и представляется упрощённой версией уравнений (3)-(6), в которых функция-источник и функции-коэффициенты имею следующие значения:

$$f(x,t) = E'_e(x) \cdot j_e + E'_p(x) \cdot j_p;$$

$$\psi(x) = 293.15 \text{ К}$$

$$k_1 = 0, k_2 = 1;$$

$$\theta_1(x, t+\tau) = -1, \theta_2(x, t+\tau) = \frac{\sigma}{\lambda}(u(x,t))^3;$$

$$\mu_1(t) = 293.15 \text{ К};$$

$$\mu_2(t) = 0,$$

где $E'_e(x)$ и $E'_p(x)$ – заданные таблично плотности распределения поглощённой энергии электронов и протонов соответственно, j_e и j_p – соответствующие плотности потоков, σ – постоянная Стефана-Больцмана $(5.670374419 \times 10^{-8} \frac{\text{Вт}}{\text{м}^2\text{К}^4})$, λ – теплопроводность стекла (взято значение $0.8 \frac{\text{Вт}}{\text{м}\cdot\text{К}}$). В качестве

значения плотности стекла взято значение $\rho = 2.5 \frac{\text{г}}{\text{см}^3}$, а в качестве теплоёмкости единицы массы $c = 703 \frac{\text{Дж}}{\text{кг}\cdot\text{К}}$.

При численном решении задачи было взято $M + 1 = 100$ пространственных точек и $N + 1 = 1000$ временных.

Графический вид распределения плотности поглощённой энергии следующий:

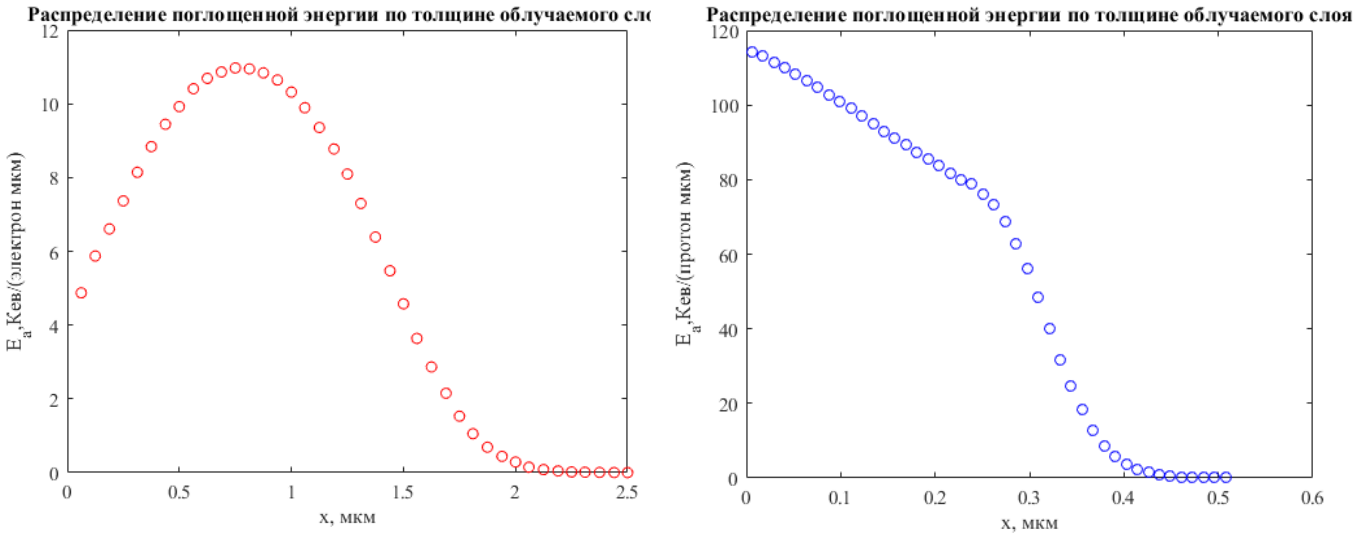

Рисунок 1. Распределение плотности поглощённой энергии по толщине облучаемого слоя. Слева: для электронов, справа: для протонов.

2. Полученные результаты

В качестве потоков были взяты следующие комбинации:

- нулевые потоки электронов и протонов;
- ненулевой поток протонов, нулевой поток электронов;
- нулевой поток протонов, ненулевой поток электронов;
- минимальные ненулевые потоки электронов и протонов;
- максимальные потоки электронов и протонов.

Нулевые потоки ($\varphi_e = \varphi_p = 0$):

Для нулевых потоков (рис. 2), как и ожидалось, температура меняется несильно ввиду малой интенсивности теплового сброса, однако и стационарной

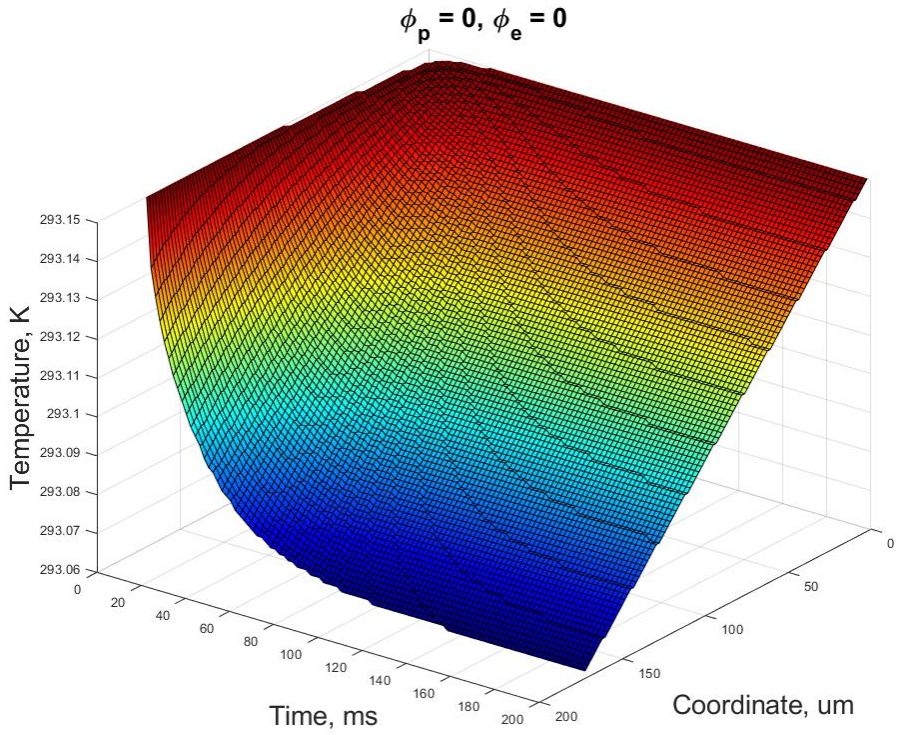

Рисунок 2. Распределение температуры и его изменение со временем для $\varphi_e = \varphi_p = 0$.

картины получиться не удастся ввиду того, что тепловой сброс присутствует постоянно.

Ненулевой поток протонов, нулевой поток электронов ($\varphi_e = 0$):

Случай максимального потока протонов ($\varphi_p = 10^{12}$):

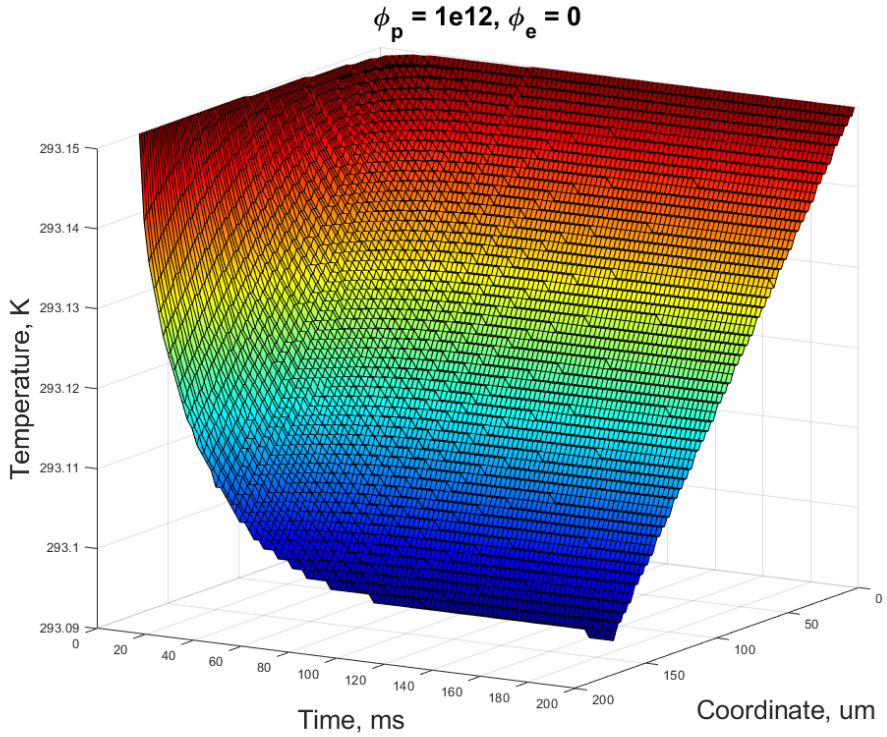

Рисунок 3. Распределение температуры и его изменение со временем для $\varphi_e = 0, \varphi_p = 10^{12}$.

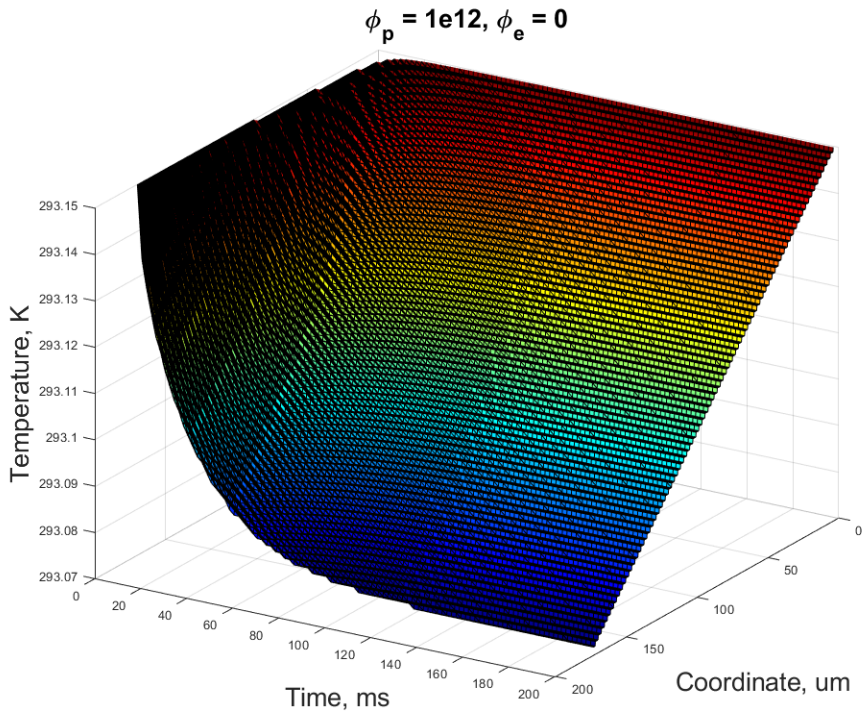

Рисунок 4. Распределение температуры и его изменение со временем для $\varphi_e = 0, \varphi_p = 10^{12}$. Увеличенное число точек пространства.

Даже максимальный поток протонов (рис. 3) не справляется с достаточным подогревом образца. Причина в том, что основная доля поглощаемой энергии протонов (рис. 1) приходится на тонкий приграничный слой и сразу же сбрасывается в виде теплового потока. Увеличение числа точек разбиения области до 1001 (рис. 4) картины не изменило.

Нулевой поток протонов, ненулевой поток электронов ($\varphi_p = 0$):

Случай максимального потока электронов ($\varphi_e = 10^{13}$):

Появляется возможность наблюдения за стационарным распределением тепла в облучаемом объекте. Также важно отметить то, что график стационарного распределения (рис. 6) заканчивается практически горизонтально, что говорит о малости радиационного сброса тепла, однако если бы он отсутствовал, то никакой стационарной картины не было бы. Принципиально также то, что увеличение числа точек разбиения области до 1001 (рис. 7) не меняет вида стационарного распределения, что говорит об оптимально подобранном шаге.

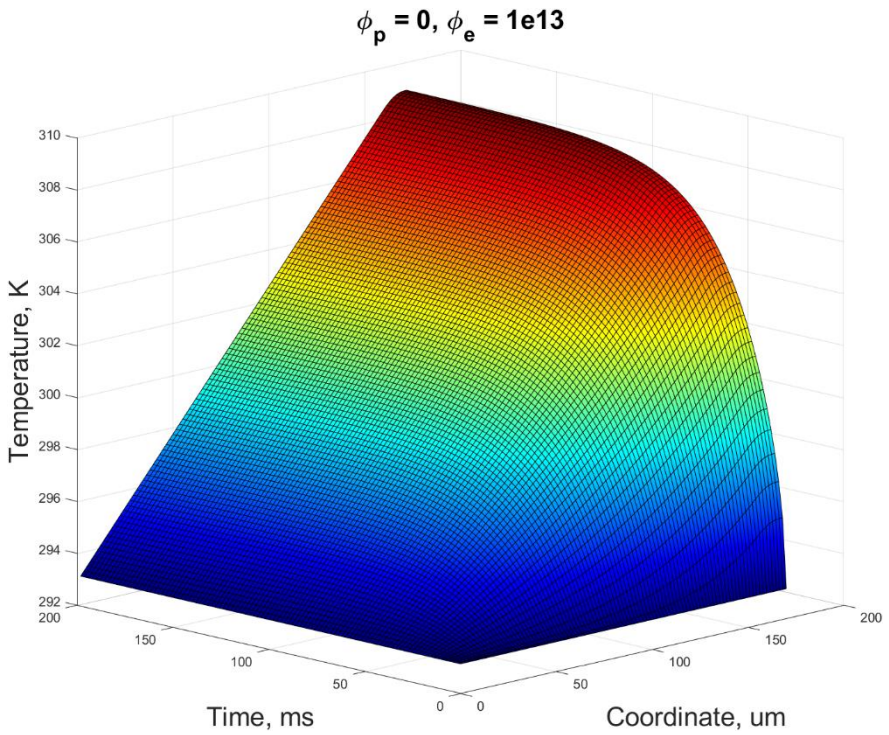

Рисунок 5. Распределение температуры и его изменение со временем для $\varphi_e = 10^{13}, \varphi_p = 0$.

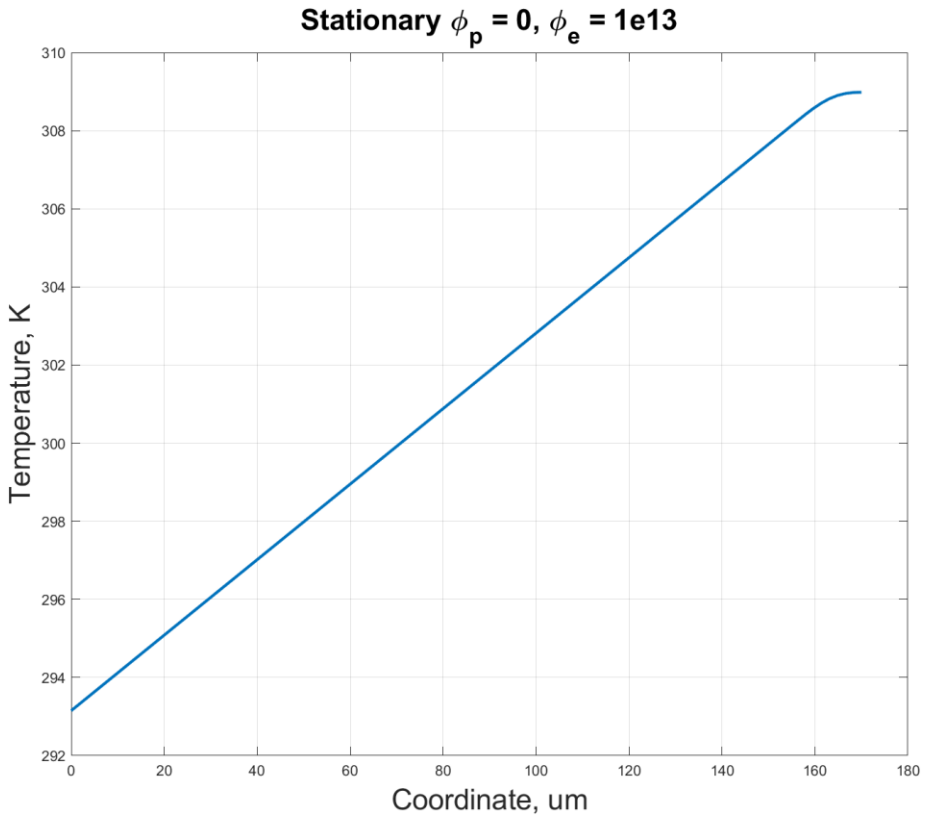

Рисунок 6. Стационарное распределение температуры для $\varphi_e = 10^{13}, \varphi_p = 0$.

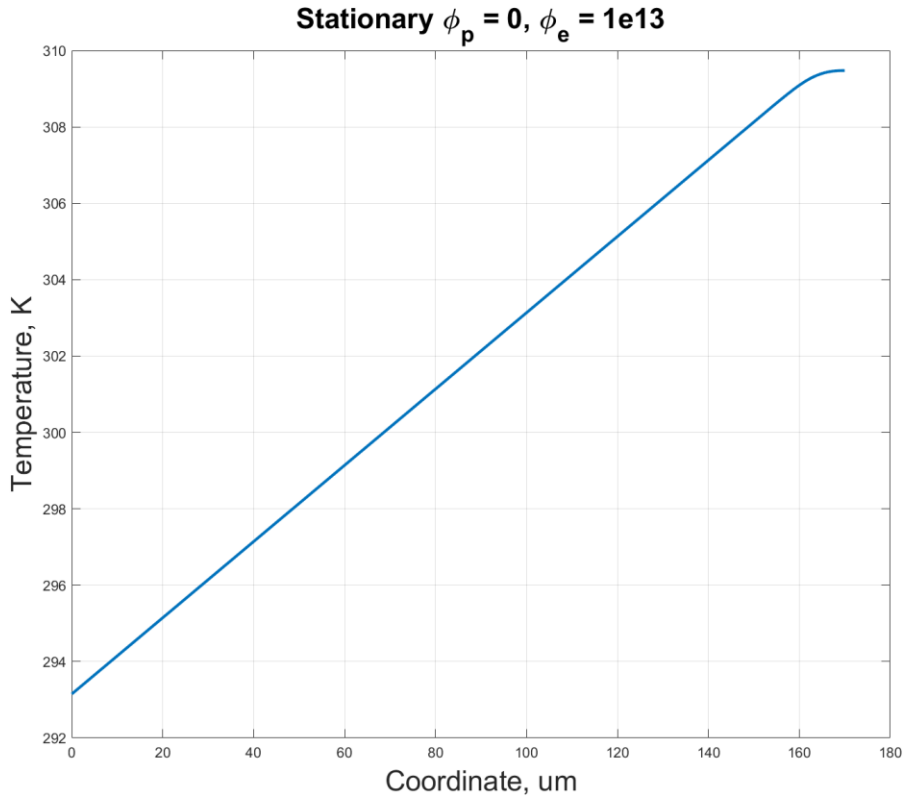

Рисунок 7. Стационарное распределение температуры для $\varphi_e = 10^{13}, \varphi_p = 0$. Увеличенное число точек пространства.

Случай минимального потока электронов ($\varphi_e = 10^{10}$):

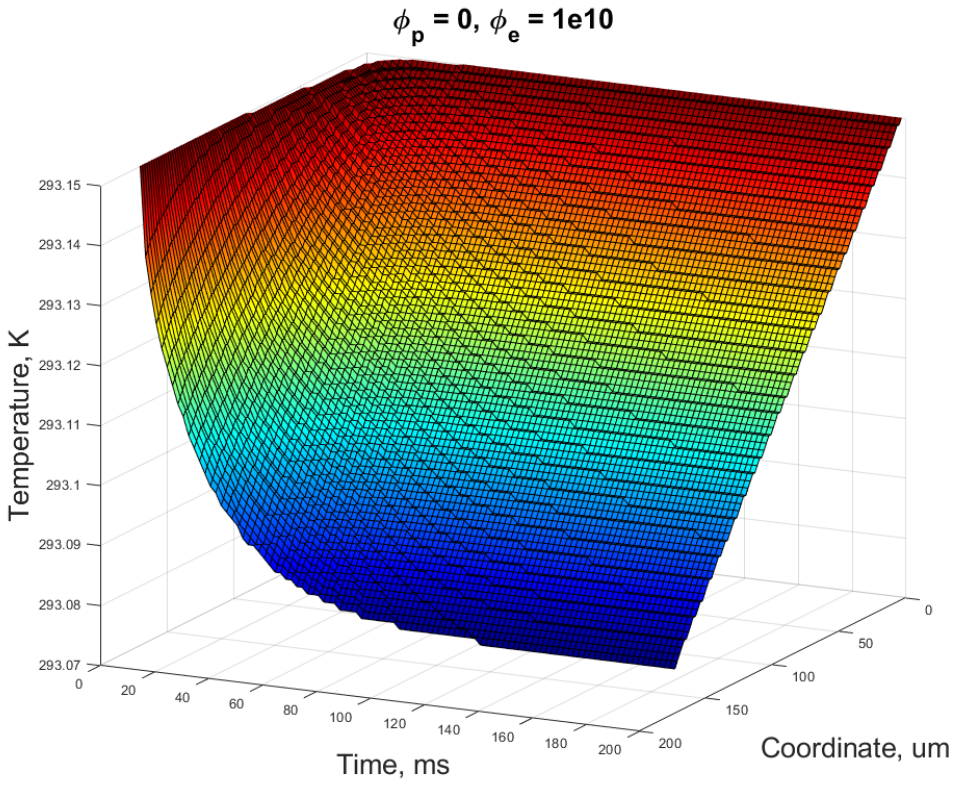

Рисунок 8. Распределение температуры и его изменение со временем для $\varphi_e = 10^{10}, \varphi_p = 0$.

Минимальный поток электронов (рис. 8) так же, как и поток протонов, не справляется с подогревом образца. На этот раз причина заключается в малости потока, что влечёт за собой малость функции-источника в дифференциальном уравнении (3).

Минимальные ненулевые потоки электронов и протонов ($\varphi_e = 10^{10}, \varphi_p = 10^9$):

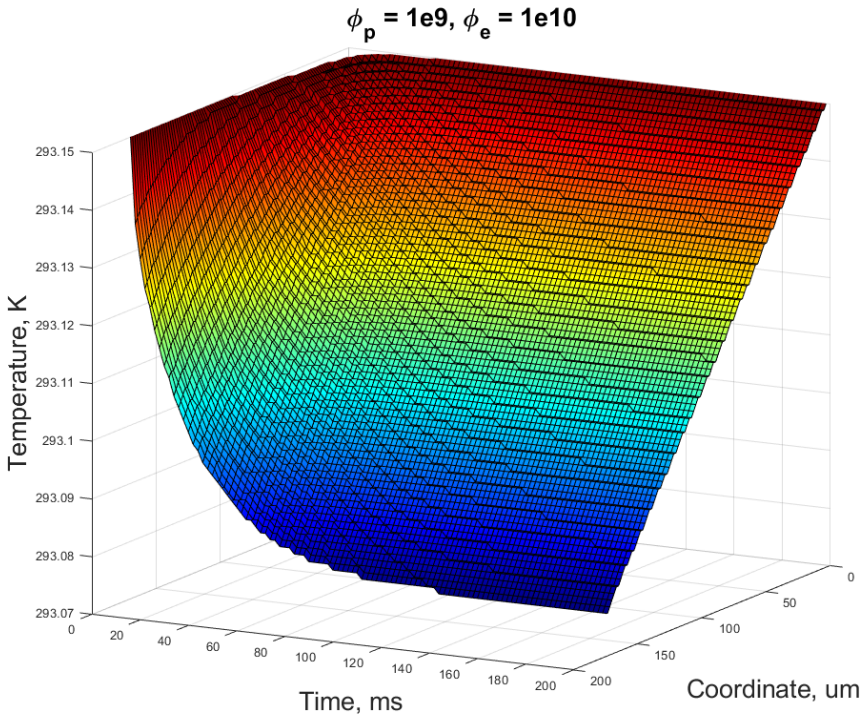

Рисунок 9. Распределение температуры и его изменение со временем для $\varphi_e = 10^{10}, \varphi_p = 10^9$.

В случае комбинации минимальных потоков, как, вероятно, и ожидалось проявляется недостаточный подогрев облучаемого объекта. В данном случае накладываются сразу две причины: быстрый сброс тепла от протонов и недостаточные потоки.

Максимальные ненулевые потоки электронов и протонов ($\varphi_e = 10^{13}, \varphi_p = 10^{12}$):

В случае максимальных совместных потоков (рис. 10, 11) распределение не сильно отличается от случая максимального потока электронов и нулевого потока протонов (рис. 5, 6). Причина та же: тепло от потока протонов тут же сбрасывается в виде излучения.

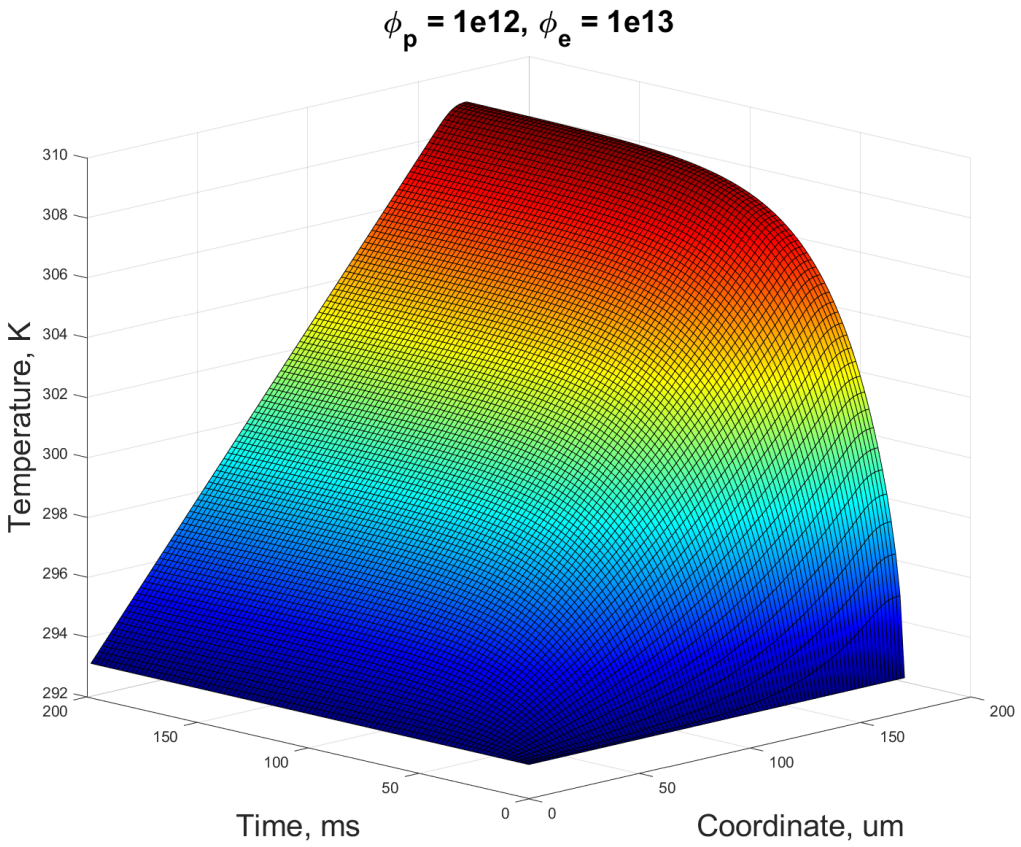

Рисунок 10. Распределение температуры и его изменение со временем для $\varphi_e = 10^{13}, \varphi_p = 10^{12}$.

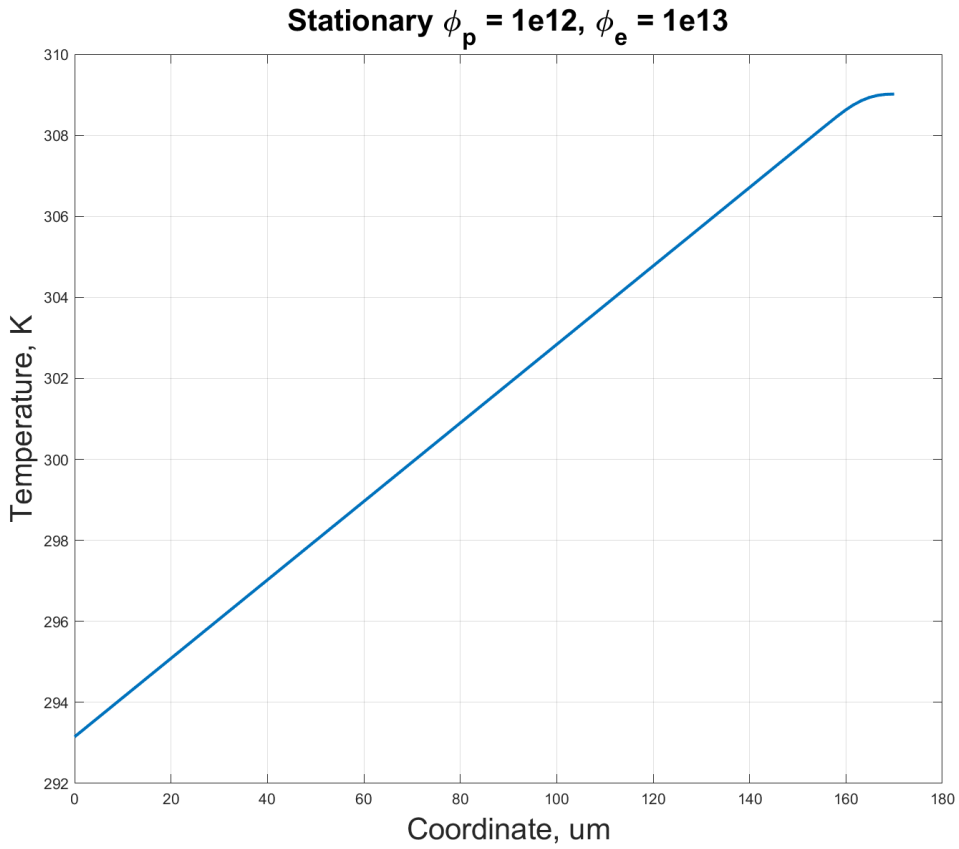

Рисунок 11. Стационарное распределение температуры для $\varphi_e = 10^{13}, \varphi_p = 10^{12}$.

Для наглядности были проведены расчёты с увеличенным суммарным потоком, чтобы проследить, как изменится результат ($\varphi_e = 10^{16}, \varphi_p = 10^{15}$):

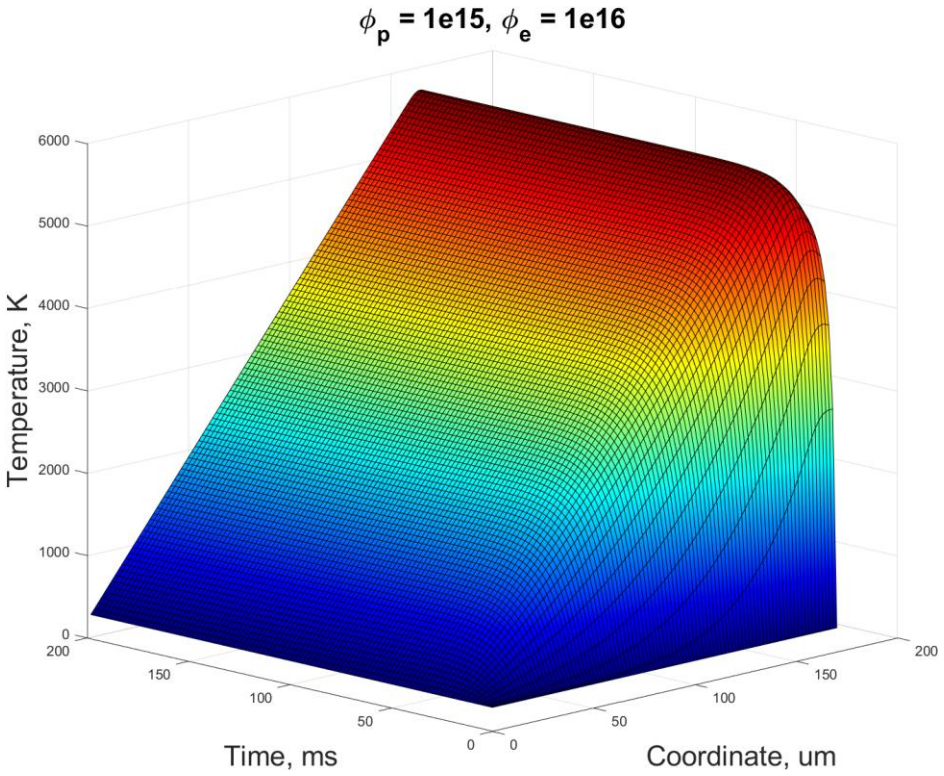

Рисунок 12. Распределение температуры и его изменение со временем для $\varphi_e = 10^{16}, \varphi_p = 10^{15}$.

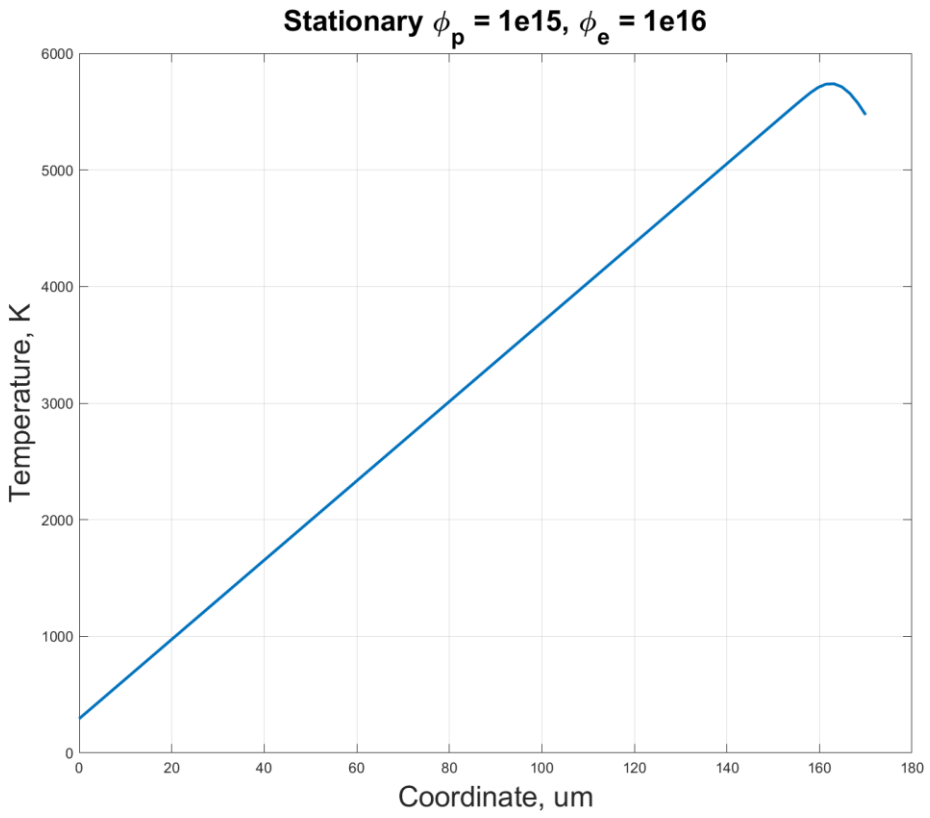

Рисунок 13. Стационарное распределение температуры для $\varphi_e = 10^{16}, \varphi_p = 10^{15}$.

Несмотря на практическую недостоверность полученных результатов для увеличенных потоков (рис. 12, 13), заключающуюся в том, что стекло (облучаемый объект) плавится при намного меньших температурах, полученная картина стационарного распределения выявляет тот факт, что максимум распределения температуры из-за теплового сброса несколько смещается от границы облучаемой поверхности.

Заключение

Численное решение уравнения теплопроводности с радиационным сбросом тепла представляет собой весьма значимую и непростую задачу. Непростая эта задача ввиду того, что граничное условие на краю, где имеется радиационный сброс достаточно сложное. Однако решение этой задачи при помощи неявной разностной схемы может доставить трудности лишь при выводе различных коэффициентов (прогоночные коэффициенты, граничные значения). Сам процесс решения представляет собой простой метод прогонки.

Полученное решение для различных величин потока частиц показало ряд физических процессов, ответственных за установление распределения температуры:

- если поток приходится лишь на приграничную часть облучаемого объекта (как у протонов), то постоянный тепловой сброс будет мешать накоплению энергии объектом;
- если же поток пробирается несколько вглубь облучаемого объекта (как в случае электронов), то прогрев объекта становится возможен и стационарное распределение температуры устанавливается тогда, когда этот поток достаточен для поддержания температуры;
- увеличение потока показало важную особенность решения уравнения теплопроводности с радиационным тепловым сбросом: получаемое распределение имеет пик не у самой границы, а на некотором расстоянии от неё.

Приложение

1. Листинг программы решения поставленной задачи (C + +)

Файл *main*:

```
#include <iostream>
#include <string>
#include <direct.h>

#include "Eigen/Eigen"
#include "Namespaces\Phys.h"
#include "Classes\Shape.h"
#include "Classes\Material.h"
#include "Classes\Medium.h"
#include "Classes\NonStatThermalCondEq.h"

void nonStationaryTemperatureDistrib();


int main()
{
	setlocale(LC_ALL, "Russian"); // Отображение кириллицы в консоли
	std::cout.precision(6); // Точность вывода десятичных чисел
	std::srand(time(0)); // Синхронизация рандомайзера с текущем временем

	nonStationaryTemperatureDistrib();
}


void nonStationaryTemperatureDistrib()
{
	double duration = 200.0 * phys::ms();
	double mediumLength = 40.0 * phys::mm();
	double mediumWidth = 40.0 * phys::mm();
	double mediumHeight = 0.17 * phys::mm();

	Eigen::Vector3d mediumCenterLocation = Eigen::Vector3d(0.0, 0.0, mediumHeight / 2.0);

	double tableTemperature = (20.0 + 273.15) * phys::K();
	double protonEnergy = 30.0 * phys::KeV();
	double electronEnergy = 15.0 * phys::KeV();
	double protonFlowDensity = 0.0 / phys::s() / pow(phys::cm(), 2.0); // 0.0, 1e9, 1e10, 1e11, 1e12
	double electronFlowDensity = 1e13 / phys::s() / pow(phys::cm(), 2.0); // 0.0, 1e10, 1e11, 1e12,
1e13

	std::string inputDataFolder = "D:/Основные файлы/Учёба/Вуз/Вычислительная физика/8 семестр/Курсо-
вая/ThermalCond/InputData/";

	Shape* mediumShape = new Parallelepiped(mediumLength, mediumWidth, mediumHeight);
	Material* mediumMaterial = new Material("Glass");
	mediumMaterial->loadAbsorbedEnergyPerProtonData(inputDataFolder + "AbsEnDensPerProton.txt");
	mediumMaterial->loadAbsorbedEnergyPerElectronData(inputDataFolder + "AbsEnDensPerElectron.txt");
	Medium* medium = new Medium(mediumCenterLocation, mediumShape, mediumMaterial);

	std::vector<double> initCond;
	std::string boundCondType = "Dirichlet_Radiation";
	std::vector<double> leftBoundCond;
	std::vector<double> rightBoundCond;
	std::vector<std::vector<double>> sourceTerm;

	int numberOfSpatialNodes = 100;
	std::vector<double> spatialNodalGrid;
	for (int m = 0; m < numberOfSpatialNodes; m++)
	{
		spatialNodalGrid.push_back(0.0 + m * mediumHeight / (numberOfSpatialNodes - 1.0));
		initCond.push_back(tableTemperature);
	}

	int numberOfTimeNodes = 1000;
	std::vector<double> timeNodalGrid;
	std::vector<double> sourceTermAtT;
	for (int n = 0; n < numberOfTimeNodes; n++)
```

```
	{
		timeNodalGrid.push_back(0.0 + n * duration / (numberOfTimeNodes - 1.0));
		leftBoundCond.push_back(tableTemperature);
		rightBoundCond.push_back(tableTemperature);

		sourceTermAtT.clear();
		for (int m = 0; m < numberOfSpatialNodes; m++)
		{
			sourceTermAtT.push_back(mediumMaterial->getAbsorbedEnergyDensPerProton(spatial-
NodalGrid.at(m)) * protonFlowDensity +
				mediumMaterial->getAbsorbedEnergyDensPerElectron(spatialNodalGrid.at(m)) *
electronFlowDensity);
		}
		std::reverse(sourceTermAtT.begin(), sourceTermAtT.end());

		sourceTerm.push_back(sourceTermAtT);
	}

	std::string outputDataFolder = "D:/Основные файлы/Учёба/Вуз/Вычислительная физика/8 семестр/Курсо-
вая/ThermalCond/OutputData/";
	std::string sourceFolder = "0_1e13/";
	bool outputDataFolderCreated = _mkdir(outputDataFolder.c_str());
	bool sourceFolderCreated = _mkdir((outputDataFolder + sourceFolder).c_str());

	NonStatThermalCondEq* diffEq = new NonStatThermalCondEq(spatialNodalGrid, timeNodalGrid, initCond,
boundCondType, leftBoundCond, rightBoundCond, medium, sourceTerm);
	diffEq->recordSolution(outputDataFolder + sourceFolder);
}
```

Файл *Phys*:

```
#pragma once
#include <math.h>

namespace phys
{
	double kg();
	double g();
	double Da();

	double m();
	double cm();
	double mm();
	double um();
	double angstrem();

	double barn();

	double s();
	double ms();

	double J();
	double eV();
	double KeV();
	double MeV();

	double K();

	double stefanBoltzmanConstant();
}
```

```
#include "Phys.h"

namespace phys
{
	double kg()
	{
		return 1.0;
	}
	double g()
	{
		return 1e-3 * kg();
	}
	double Da()
```

```
        {
                return 1.66e-27 * kg();
        }

        double m()
        {
                return 1.0;
        }
        double cm()
        {
                return 1e-2 * m();
        }
        double mm()
        {
                return 1e-3 * m();
        }
        double um()
        {
                return 1e-6 * m();
        }
        double angstrem()
        {
                return 1e-10 * m();
        }

        double barn()
        {
                return 1e-28 * pow(m(), 2.0);
        }

        double s()
        {
                return 1.0;
        }
        double ms()
        {
                return 1e-3 * s();
        }

        double J()
        {
                return 1.0;
        }
        double eV()
        {
                return 1.6e-19 * J();
        }
        double KeV()
        {
                return 1e3 * eV();
        }
        double MeV()
        {
                return 1e6 * eV();
        }

        double K()
        {
                return 1.0;
        }

        double stefanBoltzmanConstant()
        {
                return 5.670374419 * 1e-8 * J() / s() / pow(m(), 2.0) / pow(K(), 4.0);
        }
}
```

Файл *Shape*:

```
#pragma once
#define _USE_MATH_DEFINES
#include <iostream>
#include <math.h>

#include "Eigen/Eigen"
```

```
class Shape
{
public:
	Shape();

	virtual Eigen::VectorXd getProportions() = 0;

protected:
	int dimensions;
};


class Parallelepiped : public Shape
{
public:
	Parallelepiped(double _a, double _b, double _c);

	Eigen::VectorXd getProportions();

private:
	double a, b, c;
};
```

```
#include "Shape.h"

Shape::Shape()
{
}


Parallelepiped::Parallelepiped(double _a, double _b, double _c)
{
	dimensions = 3;
	a = _a;
	b = _b;
	c = _c;
}

Eigen::VectorXd Parallelepiped::getProportions()
{
	Eigen::VectorXd proportions(dimensions);
	proportions << a, b, c;

	return proportions;
}
```

Файл *Material*:

```
#pragma once
#include <iostream>
#include <string>
#include <math.h>
#include <vector>
#include <fstream>

#include "Phys.h"

class Material
{
public:
	Material(std::string _materialName);

	double getDensity();
	double getConductivity();
	double getCapacity();

	void loadAbsorbedEnergyPerProtonData(std::string fileLocation);
	void loadAbsorbedEnergyPerElectronData(std::string fileLocation);
```

```
        double getAbsorbedEnergyDensPerProton(double depth);
        double getAbsorbedEnergyDensPerElectron(double depth);

private:
        std::string materialName;
        double density;
        double conductivity;
        double capacity;
        std::vector<double> depthsProton;
        std::vector<double> energyDensPerProton;
        std::vector<double> depthsElectron;
        std::vector<double> energyDensPerElectron;
};
```

```
#include "Material.h"

Material::Material(std::string _materialName)
{
        materialName = _materialName;

        if (_materialName == "Glass" || _materialName == "SiO2")
        {
                density = 2.5 * phys::g() / pow(phys::cm(), 3.0);
                conductivity = 0.8 * phys::J() / phys::s() / phys::m() / phys::K();
                capacity = 703 * phys::J() / phys::K() / phys::kg();
        }
}

double Material::getDensity()
{
        return density;
}

double Material::getConductivity()
{
        return conductivity;
}

double Material::getCapacity()
{
        return capacity;
}

void Material::loadAbsorbedEnergyPerProtonData(std::string fileLocation)
{
        std::ifstream inFile;
        inFile.open(fileLocation);

        double depth;
        double depthUnits = phys::um();
        double energyDens;
        double energyDensUnits = phys::KeV() / phys::um();
        if (inFile.is_open())
        {
                while (!inFile.eof())
                {
                        inFile >> depth >> energyDens;
                        depthsProton.push_back(depth * depthUnits);
                        energyDensPerProton.push_back(energyDens * energyDensUnits);
                }
                inFile.close();

                std::cout << "Файл Protons успешно прочитан" << std::endl;
        }
        else
        {
                std::cout << "Ошибка при прочтении файла Protons" << std::endl;
        }
}

void Material::loadAbsorbedEnergyPerElectronData(std::string fileLocation)
{
        std::ifstream inFile;
        inFile.open(fileLocation);

        double depth;
```

```
        double depthUnits = phys::g() / pow(phys::cm(), 2.0);
        double energyDens;
        double energyDensUnits = phys::MeV() * (phys::g() / pow(phys::cm(), 2.0));
        if (inFile.is_open())
        {
                while (!inFile.eof())
                {
                        inFile >> depth >> energyDens;
                        depthsElectron.push_back(depth * depthUnits / density);
                        energyDensPerElectron.push_back(energyDens * energyDensUnits * density);
                }
                inFile.close();

                std::cout << "Файл Electrons успешно прочитан" << std::endl;
        }
        else
        {
                std::cout << "Ошибка при прочтении файла Electrons" << std::endl;
        }
}

double Material::getAbsorbedEnergyDensPerProton(double depth)
{
        int index = 0;
        for (double depthProton : depthsProton)
        {
                if (depth > depthProton)
                {
                        index++;
                        continue;
                }
                else
                {
                        break;
                }
        }

        if (index >= depthsProton.size())
        {
                return 0.0;
        }
        else
        {
                return energyDensPerProton.at(index);
        }
}

double Material::getAbsorbedEnergyDensPerElectron(double depth)
{
        int index = 0;
        for (double depthElectron : depthsElectron)
        {
                if (depth > depthElectron)
                {
                        index++;
                        continue;
                }
                else
                {
                        break;
                }
        }

        if (index >= depthsElectron.size())
        {
                return 0.0;
        }
        else
        {
                return energyDensPerElectron.at(index);
        }
}
```

Файл *Medium*:

```
#pragma once
#include "Shape.h"
#include "Material.h"

class Medium
{
public:
	Medium(Eigen::Vector3d _mediumCenterLocation, Shape* _mediumShape, Material* _meduimMaterial);

	Material* getMediumMaterial();

private:
	Eigen::Vector3d mediumCenterLocation;
	Shape* mediumShape;
	Material* meduimMaterial;
};
```

```
#include "Medium.h"

Medium::Medium(Eigen::Vector3d _mediumCenterLocation, Shape* _mediumShape, Material* _meduimMaterial)
{
	mediumCenterLocation = _mediumCenterLocation;
	mediumShape = _mediumShape;
	meduimMaterial = _meduimMaterial;
}

Material* Medium::getMediumMaterial()
{
	return meduimMaterial;
}
```

Файл *NonStatThermalCondEq*:

```
#pragma once
#include <vector>

#include "Medium.h"

class NonStatThermalCondEq
{
public:
	NonStatThermalCondEq(std::vector<double> _spatialNodalGrid, std::vector<double> _timeNodalGrid,
std::vector<double> _initCond,
		std::string _boundCondType, std::vector<double> _leftBoundCond, std::vector<double>
_rightBoundCond, Medium* _medium, std::vector<std::vector<double>> _sourceTerm);

	void recordSolution(std::string filePath);

private:
	void sweepMethodDR(); // Dirichlet - Radiation flux

	double parameter;
	std::vector<double> spatialNodalGrid;
	std::vector<double> timeNodalGrid;
	std::vector<double> initCond;
	std::vector<double> boundCond;
	std::vector<double> leftBoundCond;
	std::vector<double> rightBoundCond;
	std::string boundCondType;
	std::vector<std::vector<double>> sourceTerm;
	Medium* medium;

	double spatialStep;
	double timeStep;
	int M;
	int N;
	std::vector<double> coefficientsA;
	std::vector<double> coefficientsB;
	std::vector<std::vector<double>> solution;
};
```

```
#include "NonStatThermalCondEq.h"

NonStatThermalCondEq::NonStatThermalCondEq(std::vector<double> _spatialNodalGrid, std::vector<double>
_timeNodalGrid, std::vector<double> _initCond,
	std::string _boundCondType, std::vector<double> _leftBoundCond, std::vector<double> _rightBound-
Cond, Medium* _medium, std::vector<std::vector<double>> _sourceTerm)
{
	spatialNodalGrid = _spatialNodalGrid;
	timeNodalGrid = _timeNodalGrid;
	initCond = _initCond;
	boundCondType = _boundCondType;
	leftBoundCond = _leftBoundCond;
	rightBoundCond = _rightBoundCond;
	medium = _medium;

	std::vector<double> sourceTermAtT;
	for (std::vector<double> sourceT : _sourceTerm)
	{
		sourceTermAtT.clear();
		for (double sourceX : sourceT)
		{
			sourceTermAtT.push_back(sourceX / (_medium->getMediumMaterial()->getDensity() *
_medium->getMediumMaterial()->getCapacity()));
		}
		sourceTerm.push_back(sourceTermAtT);
	}

	parameter = pow(_medium->getMediumMaterial()->getConductivity() / (_medium->getMediumMaterial()-
>getDensity() * _medium->getMediumMaterial()->getCapacity()), 0.5);
	spatialStep = _spatialNodalGrid.at(1) - _spatialNodalGrid.at(0);
	timeStep = _timeNodalGrid.at(1) - _timeNodalGrid.at(0);
	M = _spatialNodalGrid.size() - 1;
	N = _timeNodalGrid.size() - 1;

	solution.push_back(initCond);

	if (_boundCondType == "Dirichlet_Radiation")
	{
		sweepMethodDR();
	}
}

void NonStatThermalCondEq::sweepMethodDR()
{
	std::vector<double> solutionAtTimeStep;
	double solutionAtPoint;
	std::vector<double> coefficientsA;
	std::vector<double> coefficientsB;
	std::vector<double> reverseSolutionAtTimeStep;

	double factor = pow(parameter / spatialStep, 2.0) * timeStep;

	double A = 1.0;
	double B = 1.0;
	double C = 2.0 + 1.0 / factor;
	double FFactorSol = 1 / factor;
	double FFactorSource = spatialStep * spatialStep / parameter / parameter;
	double radiationCoef;

	for (int n = 1; n <= N; n++)
	{
		coefficientsA.clear();
		coefficientsB.clear();

		coefficientsA.push_back(0.0);
		coefficientsB.push_back(leftBoundCond.at(n));
		for (int m = 2; m <= M; m++)
		{
			coefficientsA.push_back(B / (C - A * coefficientsA.at(m - 2)));
			coefficientsB.push_back((A * coefficientsB.at(m - 2) + FFactorSol * solution.at(n
- 1).at(m - 1) +
				FFactorSource * sourceTerm.at(n - 1).at(m - 1)) / (C - A * coeffi-
cientsA.at(m - 2)));
		}

		radiationCoef = phys::stefanBoltzmanConstant() * spatialStep / medium->getMediumMate-
rial()->getConductivity() * pow(solution.at(n - 1).at(M), 3.0);
```

```
                solutionAtTimeStep.clear();
                reverseSolutionAtTimeStep.clear();
                reverseSolutionAtTimeStep.push_back((coefficientsB.at(M - 1) + (solution.at(n - 1).at(M) +
timeStep * sourceTerm.at(n).at(M)) / factor / 2.0 ) /
                        (1.0 + 0.5 / factor - coefficientsA.at(M - 1) + radiationCoef));
                int reverseIndex = 0;
                for (int m = M - 1; m >= 0; m--)
                {
                        reverseSolutionAtTimeStep.push_back(coefficientsA.at(m) * reverseSolutionAt-
TimeStep.at(reverseIndex) + coefficientsB.at(m));
                        reverseIndex++;
                }

                solutionAtTimeStep = reverseSolutionAtTimeStep;
                std::reverse(solutionAtTimeStep.begin(), solutionAtTimeStep.end());

                solution.push_back(solutionAtTimeStep);
        }
}

void NonStatThermalCondEq::recordSolution(std::string filePath)
{
        std::string fileName;

        for (int n = 0; n < timeNodalGrid.size(); n++)
        {
                fileName = std::to_string(timeNodalGrid.at(n)) + ".mat";

                std::ofstream outFile;
                outFile.open(filePath + fileName);
                for (int m = 0; m < spatialNodalGrid.size(); m++)
                {
                        outFile << spatialNodalGrid.at(m) << "\t" << solution.at(n).at(m) << "\n";
                }
                outFile.close();

                std::cout << "Файл " + fileName + " успешно создан" << std::endl;
        }
}
```

2. Листинг программы построения графиков (*Matlab*)

```
%% Declarations
clear all
close all

caseType = "0_1e13";
skipFiles = 10;
timeScale = 1e-3; % ms
spatialScale = 1e-6; % um

folder = "D:\Основные файлы\Учёба\Вуз\Вычислительная физика\8 семестр\Курсовая\";
inputDataFolder = "ThermalCond\OutputData\" + caseType + "\";
saveDataFolder = "Results\";

cd(folder + inputDataFolder);
files = dir('*.*');

nFiles = size(files, 1) - 2;
numberOfTimeNodes = nFiles;

c = lines;

SpatialGrid = cell(nFiles / skipFiles, 1);
TimeGrid = cell(nFiles / skipFiles, 1);
Solution = cell(nFiles / skipFiles, 1);
for iFile = 1:skipFiles:nFiles
    Data = load(files(iFile + 2).name, '-ascii');
```

```
    SpatialGrid{iFile} = Data(:, 1) / spatialScale;

    timeStr = extractBefore(files(iFile + 2).name, ".mat");
    timeStr = strrep(timeStr, ',', '.');
    time = str2double(timeStr);
    TimeGrid{iFile} = zeros(size(SpatialGrid{iFile}, 1), 1) + time / timeScale;

    Solution{iFile} = Data(:, 2);
end

C = [];
T = [];
X = [];
for i = 1:skipFiles:numberOfTimeNodes
    C = [C Solution{i}];
    T = [T TimeGrid{i}];
    X = [X SpatialGrid{i}];
end

%% Plot data

colormap jet

h1 = figure(1);
set(gcf, 'Position',  [400, 100, 1000, 800])
legendStr = [];
surf(X, T, C, 'FaceColor', 'interp')
grid on
view(-45, 15)
xlabel('Coordinate, um', 'FontSize', 20)
ylabel('Time, ms', 'FontSize', 20)
zlabel('Temperature, K', 'FontSize', 20)
title("\phi_p = " + extractBefore(caseType, '_') + ", \phi_e = " + extractAfter(ca-
seType, '_'), 'FontSize', 20)

h2 = figure(2)
set(gcf, 'Position',  [400, 100, 1000, 800])
plot(SpatialGrid{end}, Solution{end}, 'LineWidth', 2)
grid on
xlabel('Coordinate, um', 'FontSize', 20)
ylabel('Temperature, K', 'FontSize', 20)
title("Stationary \phi_p = " + extractBefore(caseType, '_') + ", \phi_e = " + ex-
tractAfter(caseType, '_'), 'FontSize', 20)

%% Save results
cd(folder);
mkdir(saveDataFolder)
cd(saveDataFolder);

exportgraphics(h1, "TempDistrib_" + caseType + "_" + num2str(numberOfTimeNodes) +
".png")
exportgraphics(h2, "Stationar tempDistrib_" + caseType + "_" + num2str(numberOfTime-
Nodes) + ".png")
```