МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав.кафедрой Соловейчик Ю.Г.
(фамилия, имя, отчество)

(подпись)

«___»_____2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Судьяровой Марии Максимовны

(фамилия, имя, отчество студента – автора работы)

Разработка методов моделирования трехмерных нестационарных электромагнитных полей в проводящих средах, порождаемых индукционным витком

(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 01.03.02. Прикладная математика и информатика
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ	**Автор выпускной квалификационной работы**
Вагин Д.В. (фамилия, имя, отчество)	Судьярова М.М. (фамилия, И.О.)
к.т.н., доцент (ученая степень, ученое звание)	ФПМИ, ПМ-83 (факультет, группа)
(подпись, дата)	(подпись, дата)

Новосибирск, 2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Соловейчик Ю.Г.
(фамилия, имя, отчество)

«21» марта 2022 г.

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Судьяровой Марии Максимовне***
(фамилия, имя, отчество студента)

Направление подготовки *01.03.02. Прикладная математика и информатика*

Факультет Прикладной математики и информатики

Тема ***Разработка методов моделирования трехмерных нестационарных электромагнитных полей в проводящих средах, порождаемых индукционным витком***

Исходные данные (или цель работы):

Целью является разработка метода моделирования трехмерных нестационарных электромагнитных полей в проводящих средах, порождаемых индукционным витком, а именно разработка программного решения и проведение исследований для задачи моделирования электромагнитного поля с помощью решения параболической начально-краевой задачи методом конечных элементов в декартовой системе координат, где векторные базисные функции первого порядка на параллелепипедах, трехслойная неявная схема для аппроксимации по времени; и для двухмерной краевой задачи для петли в цилиндрической системе координат, где базисные функции билинейные на прямоугольниках, аппроксимация по времени – трехслойная неявная схема.

Структурные части работы:

1. Изучение теоретического материала по теме электромагнитной разведки

2. Разработка генераторов сеток для решения двухмерной и трехмерной задач.

3. Реализация программ для решения двухмерной и трехмерной задач (задача на выделение).

4. Тестирование программной разработки и анализ полученных результатов решения.

5. Проведение исследований для двухмерной и трехмерной задач на разработанной программе.

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Вагин Д.В.	*Судьярова М.М.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
к.т.н., доцент	*ФПМИ, ПМ-83*
(ученая степень, ученое звание)	(факультет, группа)
21.03.2022 г.	*21.03.2022 г.*
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № 1502/2 от «21 » *марта 2022* г.

ВКР сдана в ГЭК № _____, тема сверена с данными приказа

_____.
(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

Задорожный А.Г.
(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчет 93 с., 24 рис., 5 табл., 10 источников, 1 приложение.

МЕТОД КОНЕЧНЫХ ЭЛЕМЕНТОВ, ВЕКТОРНЫЕ БАЗИСНЫЕ ФУНКЦИИ ПЕРВОГО ПОРЯДКА, ПАРАЛЛЕЛЕПИПЕДЫ, ПЕТЛЯ, ЦИЛИНДРИЧЕСКАЯ СИСТЕМА КООРДИНАТ, БИЛИНЕЙНЫЕ БАЗИСНЫЕ ФУНКЦИИ НА ПРЯМОУГОЛЬНИКАХ

Цель работы – разработка метода моделирования трехмерных нестационарных электромагнитных полей в проводящих средах, порождаемых индукционным витком, а именно разработка программного решения и проведение исследований для задачи моделирования электромагнитного поля с помощью решения параболической начально-краевой задачи методом конечных элементов в декартовой системе координат, где реберные базисные функции первого порядка на параллелепипедах, трехслойная неявная схема для аппроксимации по времени; и для двухмерной краевой задачи для петли в цилиндрической системе координат, где базисные функции билинейные на прямоугольниках, аппроксимация по времени – трехслойная неявная схема.

В ходе работы были изучены научные материалы по теме электромагнитной разведки, разработаны генераторы сеток для решения двухмерной и трехмерной задач, создана реализация программ для решения двухмерной и трехмерной задач. Были проведены тестирование программной реализации и исследования для двухмерной и трехмерной задач на разработанной программе, проанализированы полученные результаты решения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В самом широком смысле наука геофизика – это приложение физики к исследованиям Земли, Луны и других планет. Однако обычно определение «геофизика» используется более ограниченным образом, применяясь исключительно к Земле. Прикладная геофизика предоставляет широкий спектр очень полезных и мощных инструментов, которые при правильном использовании в правильных ситуациях дают полезную информацию [4].

Целью геофизической разведки является обнаружение и, по возможности, измерение размеров и физических свойств подземных геологических структур или тел. Так, при разведке нефти в основном ищется структурная информация из-за связи нефти с особенностями, такими как антиклинали в осадочных породах. В горной геофизике упор делается на выявление и определение физических свойств. Хотя минеральные рудные тела дают отчетливые и поддающиеся измерению геофизические признаки, они часто имеют неправильную форму и залегают в породах сложной структуры, что затрудняет или делает невозможной точную количественную интерпретацию. На крупных строительных площадках часто требуется варьирование глубины коренных пород, а механические свойства вскрышных пород могут иметь важное значение, когда приходится рассчитывать способность выдерживать породами большие нагрузки.

Геофизическое исследование состоит из набора измерений, обычно собираемых по систематической схеме над земной поверхностью по суше, морю или воздуху, или вертикально в скважине. Измерения могут касаться пространственных вариаций статических силовых полей — градиентов электрического, гравитационного или магнитного «потенциала» — или характеристик волновых полей, в частности времени пробега упругих (сейсмических) волн, а также амплитудных и фазовых искажений электромагнитных волн. На эти силовые и волновые поля влияют физические свойства и структура подповерхностных пород. Поскольку физические свойства в значительной степени определяются литологией,

смена физических свойств часто соответствует геологическим границам, и, таким образом, любая структурная проблема сводится к интерпретации полей на поверхности с точки зрения этих смен. Легкость, с которой это можно сделать, зависит от многих факторов, особое значение из которых имеют сложность строения и степень контраста физических свойств пород. Понятно, что при выборе геофизического метода, который будет использоваться для изучения, проблема контрастирующих свойств подземных пород и их однородность в пределах особого образования являются важными факторами, которые следует учитывать.

Свойствами горных пород, которые наиболее широко используются в геофизических исследованиях геологоразведки являются упругость, электрическая проводимость, плотность, магнитная восприимчивость и остаточная намагниченность, а также электрическая поляризуемость. В меньшей степени используются такие свойства, как степень радиоактивности.

Вся материя имеет гравитационный эффект и горизонтальные изменения плотности внутри земли, что производит небольшие, но часто измеримые изменения гравитации над поверхностью. Точно так же многие горные породы содержат небольшое количество магнитных минералов и, следовательно, показывают степень намагниченности. Различия в интенсивности намагниченности между горными породами, возникающими из различия в магнитной восприимчивости или постоянной намагниченности, дают рост пространственному изменению результирующего магнитного поля, опять же измеримого над земной поверхностью. По формам поверхностей гравитационных или магнитных полей можно делать выводы о структуре недр [1].

Гравитационные и магнитные исследования используют естественные силовые поля. Большинство сейсмических и электрических (в том числе электромагнитных) методов, которые связаны с упругими и электрическими свойствами горных пород, требуют введения энергии в землю. Также практически возможно подавать энергию в землю с помощью индукции, используя катушку, по которой течет переменный ток с частотой тысяч или около того циклов в секунду. Однако

прямого электрического контакта с землей нет. В этих так называемых электромагнитных методах переменное магнитное поле, создаваемое передающей катушкой, вызывает вихревые токи в хороших проводниках в земле, проявляющиеся на поверхности как вторичные поля, которые могут быть измерены с помощью поисковой катушки [2].

Электромагнитные методы разведки основаны на измерении магнитных полей, связанных с переменными токами, наведенными в подземные проводники первичными магнитными полями. В большинстве методов первичное или индуцирующее поле создается искусственно путем пропускания переменного тока через катушку или петлю. Основное преимущество электромагнитных методов заключается в том, что многие системы можно использовать с самолета, т.к. проводящие заземляющие соединения не нужны. Основное их применение – поиск металлических руд, особенно сульфидов. Они являются относительно хорошими проводниками, в отличие от вмещающих пород, и, таким образом, могут быть обнаружены, несмотря на их локализованный характер и неправильную форму.

Основной принцип легче всего проиллюстрировать на примере двухкатушечной системы, показанной на рисунке 1.

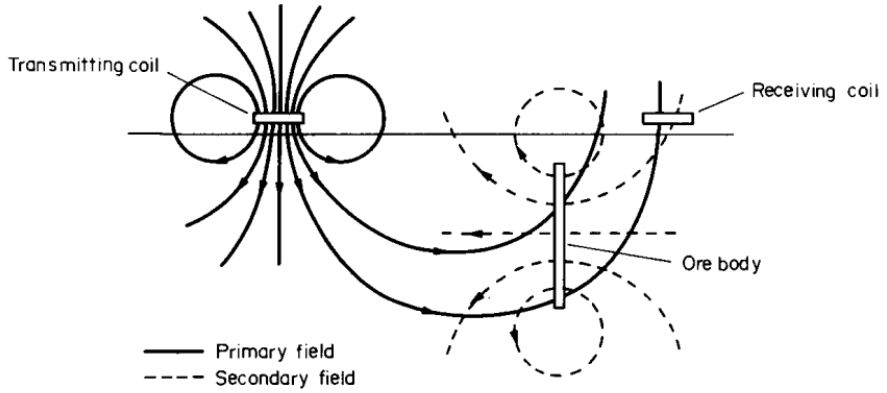

Рисунок 1 – Принцип двухкатушечной электромагнитной системы

Переменный ток, обычно с частотой от нескольких сотен до нескольких тысяч герц проходит через передающую катушку. Таким образом, в проводящей руде индуцируются вихревые токи в теле (и в незначительной степени часто во

вмещающей породе). На рисунке рудное тело представлено в виде вертикального пласта, а индуцированный ток имеет более или менее кольцеобразную форму в плоскости листа. Для проведения обследования две катушки разносятся на фиксированное расстояние в области исследования, по возможности на одной высоте.

Электромагнитные методы могут быть адаптированы в методы с применением самолетов и вертолетов. В таких методах, где используется движущийся источник, источник и приемник могут быть установлены на одном и том же самолете (например, на двух законцовках крыла). Хотя некоторые магнитные исследования проводятся на поверхности, самолет является эффективным средством для покрытия больших площадей и записи больших объемов данных, в отличие от гравиметрических исследований, для которых требуется платформа без движения. Таким образом легче достичь адекватный охват площади по сравнению с гравиметрической съемкой [3].

1. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

1.1. ПОСТАНОВКА ЗАДАЧИ

Моделирование электромагнитного поля с помощью решения параболической начально-краевой задачи методом конечных элементов в декартовой системе координат, используя векторные базисные функции первого порядка на параллелепипедах и трехслойную неявную схему для аппроксимации по времени.

1.2. ТРЕХМЕРНАЯ ЗАДАЧА

Уравнение имеет вид:

$$\frac{1}{\mu}\operatorname{rot}\operatorname{rot}\vec{\mathbf{A}} + \sigma\frac{\partial\vec{\mathbf{A}}}{\partial t} = (\sigma - \sigma_o)\overrightarrow{\mathbf{E}_o}, \tag{1}$$

где σ_o – удельная проводимость среды нормального поля, а $\overrightarrow{\mathbf{E}_o}$ получаем как производную решения двухмерной задачи:

$$\overrightarrow{\mathbf{E}_o} = (E_x{}^0\ E_y{}^0, E_z{}^0) = (-\frac{\partial A_x^{\varphi}}{\partial t}, -\frac{\partial A_y^{\varphi}}{\partial t}, -\frac{\partial A_z^{\varphi}}{\partial t}), \tag{2}$$

где

$$\begin{cases} \frac{\partial A_x^j}{\partial t} = \frac{A_x^j - A_x^{j-1}}{\Delta t}, \\ \frac{\partial A_y^j}{\partial t} = \frac{A_y^j - A_y^{j-1}}{\Delta t}, \\ \frac{\partial A_z^j}{\partial t} = \frac{A_z^j - A_z^{j-1}}{\Delta t}, \end{cases} \tag{3}$$

(или для трёхслойной временной модели формула (3) может быть преобразована в вычисление производной по времени по трёхслойной неявной схеме)

и

$$\begin{cases} A_x^{\varphi}(x,y,z,t) = -\frac{y}{r}A^{\varphi}(r,z,t), \\ A_y^{\varphi}(x,y,z,t) = \frac{x}{r}A^{\varphi}(r,z,t), \\ A_z^{\varphi}(x,y,z,t) = 0, \end{cases} \quad (4)$$

где

$$r = \sqrt{x^2 + y^2}. \quad (5)$$

Для второго временного слоя ищем решение при помощи двухслойной неявной схемы, затем воспользуемся трехслойной.

1.2.1. ДИСКРЕТИЗАЦИЯ ПО ВРЕМЕНИ

1.2.1.1. ДВУХСЛОЙНАЯ НЕЯВНАЯ СХЕМА

Представим искомое решение на интервале (t_{j-1}, t_j) в следующем виде:

$$\vec{\mathbf{A}}(x,y,z,t) \approx \vec{\mathbf{A}}^{j-1}(x,y,z)\eta_1^j(t) + \vec{\mathbf{A}}^{j}(x,y,z)\eta_0^j(t). \quad (6)$$

Функции $\eta_1^j(t), \eta_0^j(t)$ - это базисные полиномы Лагранжа, которые могут быть записаны в виде:

$$\eta_1^j(t) = -\frac{t-t_j}{\Delta t}, \quad (7)$$

$$\eta_0^j(t) = \frac{t-t_{j-1}}{\Delta t}, \quad (8)$$

где

$$\Delta t = t_j - t_{j-1}. \quad (9)$$

Вычислим производные $\frac{\partial}{\partial t}\eta_i^j(t)$ при $t = t_j$:

$$\frac{d\eta_1^j(t)}{dt} = -\frac{1}{\Delta t}, \quad (10)$$

$$\frac{d\eta_0^j(t)}{dt} = \frac{1}{\Delta t}. \quad (11)$$

Получаем:

$$\frac{1}{\mu}\text{rot rot}\,\vec{\mathbf{A}}^j + \sigma\left(\vec{\mathbf{A}}^{j-1}(x,y,z)\left(-\frac{1}{\Delta t}\right) + \vec{\mathbf{A}}^j(x,y,z)\left(\frac{1}{\Delta t}\right)\right) = (\sigma - \sigma_0)\vec{\mathbf{E}}^j. \quad (12)$$

Перенесем вправо известные значения $\vec{\mathbf{A}}$:

$$\frac{1}{\mu}\text{rot rot}\,\vec{\mathbf{A}}^j + \sigma\vec{\mathbf{A}}^j(x,y,z)\left(\frac{1}{\Delta t}\right) = (\sigma - \sigma_0)\vec{\mathbf{E}}^j + \frac{\sigma}{\Delta t}\vec{\mathbf{A}}^{j-1}(x,y,z) \quad (13)$$

Выполним конечноэлементную аппроксимацию и получим СЛАУ следующего вида:

$$\left(\frac{\sigma}{\Delta t}M + G\right)q^j = (\sigma - \sigma_0)\vec{\mathbf{E}}^j + \frac{\sigma}{\Delta t}Mq^{j-1} \quad (14)$$

1.2.1.2. ТРЕХСЛОЙНАЯ НЕЯВНАЯ СХЕМА

Представим искомое решение на интервале (t_{j-1}, t_j) в следующем виде [9]:

$$\vec{\mathbf{A}}(x,y,z,t) \approx \vec{\mathbf{A}}^{j-2}(x,y,z)\eta_2^j(t) + \vec{\mathbf{A}}^{j-1}(x,y,z)\eta_1^j(t) + \vec{\mathbf{A}}^j(x,y,z)\eta_0^j(t). \quad (15)$$

Функции $\eta_2^j(t),\ \eta_1^j(t), \eta_0^j(t)$ - это базисные квадратичные полиномы Лагранжа, которые могут быть записаны в виде:

$$\eta_2^j(t) = \frac{(t-t_{j-1})(t-t_j)}{\Delta t_1 \Delta t}, \tag{16}$$

$$\eta_1^j(t) - \frac{(t-t_{j-2})(t-t_j)}{\Delta t_1 \Delta t_0}, \tag{17}$$

$$\eta_0^j(t) = -\frac{(t-t_{j-2})(t-t_{j-1})}{\Delta t \Delta t_0}, \tag{18}$$

где

$$\Delta t = t_j - t_{j-2}, \tag{19}$$

$$\Delta t_0 = t_j - t_{j-1}. \tag{20}$$

$$\Delta t_1 = t_{j-1} - t_{j-2}, \tag{21}$$

Вычислим производные $\frac{\partial}{\partial t}\eta_i^j(t)$ при $t = t_j$:

$$\frac{d\eta_2^j(t)}{dt} = \frac{\Delta t_0}{\Delta t_1 \Delta t}, \tag{22}$$

$$\frac{d\eta_1^j(t)}{dt} = -\frac{\Delta t}{\Delta t_1 \Delta t_0}, \tag{23}$$

$$\frac{d\eta_0^j(t)}{dt} = \frac{\Delta t + \Delta t_0}{\Delta t \Delta t_0}. \tag{24}$$

Получаем:

$$\frac{1}{\mu}\operatorname{rot}\operatorname{rot}\vec{\mathbf{A}}^{j}+\ \sigma\left(\vec{\mathbf{A}}^{j-2}(x,y,z)\left(\frac{\Delta t_0}{\Delta t_1\Delta t}\right)+\vec{\mathbf{A}}^{j-1}(x,y,z)\left(-\frac{\Delta t}{\Delta t_1\Delta t_0}\right)+\right.$$

$$\left.+\vec{\mathbf{A}}^{j}(x,y,z)\left(\frac{\Delta t+\Delta t_0}{\Delta t\Delta t_0}\right)\right)=(\sigma-\sigma_0)\vec{\mathbf{E}}^{j}. \qquad (25)$$

Перенесем вправо известные значения $\vec{\mathbf{A}}$:

$$\frac{1}{\mu}\operatorname{rot}\operatorname{rot}\vec{\mathbf{A}}^{j}+\ \sigma\frac{\Delta t+\Delta t_0}{\Delta t\Delta t_0}\vec{\mathbf{A}}^{j}(x,y,z)=(\sigma-\sigma_0)\vec{\mathbf{E}}^{j}-\sigma\frac{\Delta t_0}{\Delta t_1\Delta t}\vec{\mathbf{A}}^{j-2}(x,y,z)+$$

$$+\sigma\frac{\Delta t}{\Delta t_1\Delta t_0}\vec{\mathbf{A}}^{j-1}(x,y,z). \qquad (26)$$

Выполним конечноэлементную аппроксимацию и получим СЛАУ следующего вида:

$$\left(\sigma\frac{\Delta t+\Delta t_0}{\Delta t\Delta t_0}M+G\right)q^{j}=(\sigma-\sigma_0)\vec{\mathbf{E}}^{j}-\sigma\frac{\Delta t_0}{\Delta t_1\Delta t}\ Mq^{j-2}+\sigma\frac{\Delta t}{\Delta t_1\Delta t_0}Mq^{j-1}\ . \qquad (27)$$

1.2.2. АНАЛИТИЧЕСКИЕ ВЫРАЖЕНИЯ ДЛЯ ВЫЧИСЛЕНИЯ ЛОКАЛЬНЫХ МАТРИЦ

Запишем вид локальных матриц [7]:

$$\widehat{M_{loc}}=\frac{h_x h_y h_z}{36}\begin{pmatrix}M & 0 & 0\\ 0 & M & 0\\ 0 & 0 & M\end{pmatrix}, \qquad (28)$$

$$\widehat{G\_loc}=\frac{1}{\mu}\begin{pmatrix}\frac{h_x h_y}{6h_z}G_1+\frac{h_x h_z}{6h_y}G_2 & -\frac{h_z}{6}G_2 & \frac{h_y}{6}G_3\\ -\frac{h_z}{6}G_2 & \frac{h_x h_y}{6h_z}G_1+\frac{h_y h_z}{6h_x}G_2 & -\frac{h_x}{6}G_1\\ \frac{h_y}{6}G_3^{T} & -\frac{h_x}{6}G_1 & \frac{h_x h_z}{6h_y}G_1+\frac{h_y h_z}{6h_x}G_2\end{pmatrix}, \qquad (29)$$

где

$$M = \begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 1 & 2 \\ 2 & 1 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{pmatrix}, \tag{30}$$

$$G_1 = \begin{pmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & -2 & 1 & 2 \end{pmatrix}, \tag{31}$$

$$G_2 = \begin{pmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{pmatrix}, \tag{32}$$

$$G_3 = \begin{pmatrix} -2 & 2 & -1 & 1 \\ -1 & 1 & -2 & 2 \\ 2 & -2 & 1 & -1 \\ 1 & -1 & 2 & -2 \end{pmatrix}. \tag{33}$$

1.3. ДВУХМЕРНАЯ ЗАДАЧА

МКЭ для двухмерной краевой задачи для петли

$$-\mathrm{div}\left(\frac{1}{\mu}\,\mathrm{grad}\,A_\varphi\right) + \frac{A_\varphi}{r^2} + \sigma\frac{\partial A_\varphi}{\partial t} = J_\varphi; \tag{34}$$

в цилиндрической (r, z) системе координат [8]. Базисные функции билинейные на прямоугольниках. Аппроксимация по времени – трехслойная неявная схема. Тогда задача для функции u определяется дифференциальным уравнением, заданным в области Ω с границей $S = S_1 \cup S_2 \cup S_3 \cup S_4$ и краевым условием

$$u|_S = 0, \tag{35}$$

Перепишем дифференциальное уравнение в цилиндрических координатах:

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{1}{\mu}\frac{\partial A_\varphi}{\partial r}\right)-\frac{\partial}{\partial z}\left(\frac{1}{\mu}\frac{\partial A_\varphi}{\partial z}\right)+\frac{A_\varphi}{r^2}+\sigma\frac{\partial A_\varphi}{\partial t}=f. \quad (36)$$

1.3.1. ДИСКРЕТИЗАЦИЯ ПО ВРЕМЕНИ

1.3.1.1. ДВУХСЛОЙНАЯ НЕЯВНАЯ СХЕМА

Выполним конечноэлементную аппроксимацию и получим СЛАУ следующего вида:

$$\left(\frac{\sigma}{\Delta t_0}M+G+M^{\frac{1}{r^2}}\right)q^j=b^j+\frac{\sigma}{\Delta t_0}Mq^{j-1}. \quad (37)$$

1.3.1.2. ТРЕХСЛОЙНАЯ НЕЯВНАЯ СХЕМА

Выполним конечноэлементную аппроксимацию и получим СЛАУ следующего вида:

$$\left(\sigma\frac{\Delta t+\Delta t_0}{\Delta t\Delta t_0}M+G+M^{\frac{1}{r^2}}\right)q^j=b^j-\frac{\sigma\Delta t_0}{\Delta t\Delta t_1}Mq^{j-2}+\frac{\sigma\Delta t}{\Delta t_0\Delta t_1}Mq^{j-1}. \quad (38)$$

1.3.2. КОНЕЧНОЭЛЕМЕНТНАЯ ДИСКРЕТИЗАЦИЯ И ПЕРЕХОД К ЛОКАЛЬНЫМ МАТРИЦАМ

Приведем подстановку Галеркина эквивалентную исходной задаче. Невязка исходного уравнения равна

$$R(A_\varphi)=-\mathrm{div}\left(\frac{1}{\mu}\,\mathrm{grad}\,A_\varphi\right)+\frac{A_\varphi}{r^2}+\sigma\frac{\partial A_\varphi}{\partial t}-J_\varphi. \quad (39)$$

Она должна быть ортогональна пространству пробных функций Ф, т.е.

$$\int_{\Omega} \left(-\mathrm{div}\left(\frac{1}{\mu}\ \mathrm{grad}\ A_{\varphi}\right) + \frac{A_{\varphi}}{r^2} + \sigma\frac{\partial A_{\varphi}}{\partial t} - J_{\varphi}\right) v\ d\Omega = 0, \forall v \epsilon \Phi. \qquad (40)$$

Применим формулу Грина, распишем интеграл по границе с учетом краевых условий и потребуем, чтобы $\Phi = H_0^1$. Учтем точечный источник, который представляет петлю, то есть учтем то, что правая часть равна J_{φ} в узле-источнике. Будем считать, что других распределенных источников нет. Получаем вариационное уравнение вида:

$$\int_{\Omega} \frac{1}{\mu}\ \mathrm{grad} A_{\varphi}\ \mathrm{grad}\ v_0\ d\Omega + \int_{\Omega} \frac{A_{\varphi}}{r^2} v_0 d\Omega + \int_{S_3} \beta A_{\varphi} v_0 dS = J_{\varphi}\ v_0(R_{и}, Z_{и}) +$$

$$\int_{S_2} \theta v_0 dS + \int_{S_3} \beta A_{\varphi_{\beta}} v_0 dS, \forall v_0 \epsilon H_0^1. \qquad (41)$$

Представим $v_0^h\ \epsilon\ V_0^h$ в виде:

$$v_0^h = \sum_{i \epsilon N_0} q_i^v \psi_i, \qquad (42)$$

$$A_{\varphi}{}^h = \sum_{j=1}^{n} q_j \psi_j, \qquad (43)$$

где $\{\psi_i\}$ – базис V^h (пространство, аппроксимирующее H^1), N_0 – множество индексов i таких, что ψ_i являются базисными функциями пространств V_0^h, V_g^h.

Ячейки дискретизации - прямоугольники $\Omega_{ps} = [r_p, r_{p+1}] \times [z_s, z_{s+1}]$. Приведем билинейные базисные функции в цилиндрических координатах. Одномерные линейные функции на $[r_p, r_{p+1}]$, $[z_s, z_{s+1}]$ имеют вид:

$$R_1(r) = \frac{r_{p+1} - r}{h_r}, \qquad (44)$$

$$R_2(r) = \frac{r - r_p}{h_r}, \tag{45}$$

$$Z_1(z) = \frac{z_{s+1} - z}{h_z}, \tag{46}$$

$$Z_2(z) = \frac{z - z_s}{h_z}, \tag{47}$$

где

$$h_r = r_{p+1} - r_p, \tag{48}$$

$$h_z = z_{s+1} - z_s. \tag{49}$$

Тогда локальные базисные функции запишем как:

$$\widehat{\psi}_1(r, z) = R_1(r)Z_1(z), \tag{50}$$

$$\widehat{\psi}_2(r, z) = R_2(r)Z_1(z), \tag{51}$$

$$\widehat{\psi}_3(r, z) = R_1(r)Z_2(z), \tag{52}$$

$$\widehat{\psi}_4(r, z) = R_2(r)Z_2(z). \tag{53}$$

Компоненты локальных матриц выглядят следующим образом:

$$\widehat{G}_{ij} = \int_{\Omega_k} \frac{\overline{1}}{\mu} \left(\frac{\partial \widehat{\psi}_i}{\partial r} \frac{\partial \widehat{\psi}_j}{\partial r} + \frac{\partial \widehat{\psi}_j}{\partial r} \frac{\partial \widehat{\psi}_j}{\partial r} \right) r dr dz, \tag{54}$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{ij} = \int_{\Omega_k} \frac{1}{r^2} \widehat{\psi}_i \widehat{\psi}_j r dr dz. \quad (55)$$

Разложим $\frac{1}{r^2}$ по базисным функциям:

$$\frac{1}{r^2} = \sum_{k=1}^{n} \frac{1}{r_k^2} \psi_k, \quad (56)$$

Укажем аналитические выражения для вычисления элементов локальных матриц. Запишем вспомогательные интегралы:

$$\int_{z_s}^{z_s+h_z} \left(\frac{dZ_1}{dz}\right)^2 dz = \frac{1}{h_z}, \quad (57)$$

$$\int_{z_s}^{z_s+h_z} \left(\frac{dZ_2}{dz}\right)^2 dz = \frac{1}{h_z}, \quad (58)$$

$$\int_{z_s}^{z_s+h_z} \frac{dZ_1}{dz}\frac{dZ_2}{dz} dz = -\frac{1}{h_z}, \quad (59)$$

$$\int_{z_s}^{z_s+h_z} Z_1 Z_2 dz = \frac{h_z}{6}, \quad (60)$$

$$\int_{z_s}^{z_s+h_z} (Z_1)^2 dz = \frac{h_z}{3}, \quad (61)$$

$$\int_{z_s}^{z_s+h_z} (Z_2)^2 dz = \frac{h_z}{3}, \quad (62)$$

$$\int_{r_p}^{r_p+h_r} \left(\frac{dR_1}{dr}\right)^2 r dr = \left(\frac{r_p}{h_r} + \frac{1}{2}\right), \quad (63)$$

$$\int_{r_p}^{r_p+h_r} \left(\frac{dR_2}{dr}\right)^2 r dr = \left(\frac{r_p}{h_r} + \frac{1}{2}\right), \quad (64)$$

$$\int_{r_p}^{r_p+h_r} \frac{dR_1}{dr}\frac{dR_2}{dr} rdr = \left(-\frac{r_p}{h_r} - \frac{1}{2}\right), \quad (65)$$

$$\int_{r_p}^{r_p+h_r} R_1 R_2 rdr = \left(\frac{h_r r_p}{6} + \frac{{h_r}^2}{12}\right), \quad (66)$$

$$\int_{r_p}^{r_p+h_r} (R_1)^2 rdr = \left(\frac{h_r r_p}{3} + \frac{{h_r}^2}{12}\right), \quad (67)$$

$$\int_{r_p}^{r_p+h_r} (R_2)^2 rdr = \left(\frac{h_r r_p}{3} + \frac{{h_r}^2}{4}\right). \quad (68)$$

Вычислим компоненты матрицы жесткости (матрица симметрична, поэтому вычислим только нижний треугольник и диагональные элементы). Приведем итоговый результат:

$$\widehat{G}_{11} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\int_{z_s}^{z_s+h_z}\left(\left(\frac{\partial\widehat{\psi}_1}{\partial r}\right)^2 + \left(\frac{\partial\widehat{\psi}_1}{\partial r}\right)^2\right) rdrdz = \frac{\overline{1}}{\mu}\left(\frac{h_z r_p}{3h_r} + \frac{h_z}{6} + \frac{h_r r_p}{3h_z} + \frac{{h_r}^2}{12h_z}\right), \quad (69)$$

$$\widehat{G}_{12} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\frac{dR_1}{dr}\frac{dR_2}{dr} rdr \int_{z_s}^{z_s+h_z} Z_1^2\, dz + \int_{r_p}^{r_p+h_r} R_1 R_2 rdr \int_{z_s}^{z_s+h_z}\left(\frac{dZ_1}{dz}\right)^2 dz = =$$

$$\frac{\overline{1}}{\mu}\left(-\frac{h_z r_p}{3h_r} - \frac{h_z}{6} + \frac{h_r r_p}{6h_z} + \frac{{h_r}^2}{12h_z}\right), \quad (70)$$

$$\widehat{G}_{13} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\left(\frac{dR_1}{dr}\right)^2 rdr \int_{z_s}^{z_s+h_z} Z_1 Z_2\, dz + \int_{r_p}^{r_p+h_r} R_1^2 rdr \int_{z_s}^{z_s+h_z}\frac{dZ_1}{dz}\frac{dZ_2}{dz} dz =$$

$$= \frac{\overline{1}}{\mu}\left(\frac{h_z r_p}{6h_r} + \frac{h_z}{12} - \frac{h_r r_p}{3h_z} - \frac{{h_r}^2}{12h_z}\right), \quad (71)$$

$$\widehat{G}_{14} = \widehat{G}_{23} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\frac{dR_1}{dr}\frac{dR_2}{dr} rdr \int_{z_s}^{z_s+h_z} Z_1 Z_2\, dz +$$

$$+ \int_{r_p}^{r_p+h_r} R_1 R_2 rdr \int_{z_s}^{z_s+h_z}\frac{dZ_1}{dz}\frac{dZ_2}{dz} dz = \frac{\overline{1}}{\mu}\left(-\frac{h_z r_p}{6h_r} - \frac{h_z}{12} - \frac{h_r r_p}{6h_z} - \frac{{h_r}^2}{12h_z}\right), \quad (72)$$

$$\widehat{G}_{22} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\left(\frac{dR_2}{dr}\right)^2 rdr\int_{z_s}^{z_s+h_z} Z_1^2\,dz + \int_{r_p}^{r_p+h_r} R_2^2 rdr\int_{z_s}^{z_s+h_z}\left(\frac{dZ_1}{dz}\right)^2 dz = =$$

$$\frac{\overline{1}}{\mu}\left(\frac{h_z r_p}{3h_r} + \frac{h_z}{6} + \frac{h_r r_p}{3h_z} + \frac{{h_r}^2}{4h_z}\right), \quad (73)$$

$$\widehat{G}_{24} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\left(\frac{dR_2}{dr}\right)^2 rdr\int_{z_s}^{z_s+h_z} Z_1 Z_2\,dz + \int_{r_p}^{r_p+h_r} R_2^2 rdr\int_{z_s}^{z_s+h_z}\frac{dZ_1}{dz}\frac{dZ_2}{dz}dz =$$

$$= \frac{\overline{1}}{\mu}\left(\frac{h_z r_p}{6h_r} + \frac{h_z}{12} - \frac{h_r r_p}{3h_z} - \frac{{h_r}^2}{4h_z}\right), \quad (74)$$

$$\widehat{G}_{33} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\left(\frac{dR_1}{dr}\right)^2 rdr\int_{z_s}^{z_s+h_z} Z_2^2\,dz + \int_{r_p}^{r_p+h_r} R_1^2 rdr\int_{z_s}^{z_s+h_z}\left(\frac{dZ_2}{dz}\right)^2 dz = =$$

$$\frac{\overline{1}}{\mu}\left(\frac{h_z r_p}{3h_r} + \frac{h_z}{6} + \frac{h_r r_p}{3h_z} + \frac{{h_r}^2}{12h_z}\right), \quad (75)$$

$$\widehat{G}_{34} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\frac{dR_1}{dr}\frac{dR_2}{dr} rdr\int_{z_s}^{z_s+h_z} Z_2^2\,dz + \int_{r_p}^{r_p+h_r} R_1 R_2 rdr\int_{z_s}^{z_s+h_z}\left(\frac{dZ_2}{dz}\right)^2 dz =$$

$$= \frac{\overline{1}}{\mu}\left(-\frac{h_z r_p}{3h_r} - \frac{h_z}{6} + \frac{h_r r_p}{6h_z} + \frac{{h_r}^2}{12h_z}\right), \quad (76)$$

$$\widehat{G}_{44} = \frac{\overline{1}}{\mu}\int_{r_p}^{r_p+h_r}\left(\frac{dR_2}{dr}\right)^2 rdr\int_{z_s}^{z_s+h_z} Z_2^2\,dz + \int_{r_p}^{r_p+h_r} R_2^2 rdr\int_{z_s}^{z_s+h_z}\left(\frac{dZ_2}{dz}\right)^2 dz =$$

$$= \frac{\overline{1}}{\mu}\left(\frac{h_z r_p}{3h_r} + \frac{h_z}{6} + \frac{h_r r_p}{3h_z} + \frac{{h_r}^2}{4h_z}\right), \quad (77)$$

$$\widehat{G} = \frac{\frac{\overline{1}}{\mu}\frac{h_z r_p}{h_r}}{6}\begin{bmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{bmatrix} + \frac{\frac{\overline{1}}{\mu}h_z}{12}\begin{bmatrix} 2 & -2 & 1 & -1 \\ -2 & 2 & -1 & 1 \\ 1 & -1 & 2 & -2 \\ -1 & 1 & -2 & 2 \end{bmatrix} +$$

$$+\frac{\frac{\overline{1}}{\mu}\frac{h_r r_p}{h_z}}{6}\begin{bmatrix} 2 & 1 & -2 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & -1 & 2 & 1 \\ -1 & -2 & 1 & 2 \end{bmatrix} + \frac{\frac{\overline{1}}{\mu}\frac{{h_r}^2}{h_z}}{12}\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 3 & -1 & -3 \\ -1 & -1 & 1 & 1 \\ -1 & -3 & 1 & 3 \end{bmatrix}. \quad (78)$$

Запишем элементы матрицы массы.

$$\widehat{M}^{\frac{1}{r^2}}_{ij} = \int_{\Omega_k} \frac{1}{r^2} \widehat{\psi}_i \widehat{\psi}_j r dr dz = \sum_{k=1}^{n} \frac{1}{r_k^2} \int_{\Omega_k} \widehat{\psi}_k \widehat{\psi}_i \widehat{\psi}_j r dr dz. \quad (79)$$

Вычислим вспомогательные интегралы:

$$\int_{z_s}^{z_s+h_z} Z_1^2 Z_2 dz = \frac{h_z}{12}, \quad (80)$$

$$\int_{z_s}^{z_s+h_z} Z_1 Z_2^2 dz = \frac{h_z}{12}, \quad (81)$$

$$\int_{z_s}^{z_s+h_z} (Z_1)^3 dz = \frac{h_z}{4}, \quad (82)$$

$$\int_{z_s}^{z_s+h_z} (Z_2)^3 dz = \frac{h_z}{4}, \quad (83)$$

$$\int_{r_p}^{r_p+h_r} R_1^2 R_2 r dr = h_r \left(\frac{r_p}{12} + \frac{h_r}{30}\right), \quad (84)$$

$$\int_{r_p}^{r_p+h_r} R_1 R_2^2 r dr = h_r \left(\frac{r_p}{12} + \frac{h_r}{20}\right), \quad (85)$$

$$\int_{r_p}^{r_p+h_r} (R_1)^3 r dr = h_r \left(\frac{r_p}{4} + \frac{h_r}{20}\right), \quad (86)$$

$$\int_{r_p}^{r_p+h_r} (R_2)^3 r dr = h_r \left(\frac{r_p}{4} + \frac{h_r}{5}\right). \quad (87)$$

Тогда компоненты матрицы $\widehat{M}^{\frac{1}{r^2}}$ имеют вид:

$$\widehat{M}^{\frac{1}{r^2}}_{11} = \widehat{M}^{\frac{1}{r^2}}_{33} = \frac{h_r h_z}{4} \left(\frac{1}{r^2}(r_p, z_s) \left(\frac{r_p}{3} + \frac{h_r}{15}\right) + \frac{1}{r^2}(r_{p+1}, z_s) \left(\frac{r_p}{9} + \frac{2h_r}{45}\right) \right), \quad (88)$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{13} = \widehat{M}^{\frac{1}{r^2}}{}_{31} = \frac{h_r h_z}{12}\left(\frac{1}{r^2}(r_p, z_s)\left(\frac{r_p}{2} + \frac{h_r}{10}\right) + \frac{1}{r^2}(r_{p+1}, z_s)\left(\frac{r_p}{6} + \frac{h_r}{15}\right)\right), \quad (89)$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{14} = \widehat{M}^{\frac{1}{r^2}}{}_{41} = \frac{h_r h_z}{12}\left(\frac{1}{r^2}(r_p, z_s)\left(\frac{r_p}{6} + \frac{h_r}{15}\right) + \frac{1}{r^2}(r_{p+1}, z_s)\left(\frac{r_p}{6} + \frac{h_r}{10}\right)\right), \quad (90)$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{21} = \widehat{M}^{\frac{1}{r^2}}{}_{43} = \widehat{M}^{\frac{1}{r^2}}{}_{34} = \widehat{M}^{\frac{1}{r^2}}{}_{12} = \frac{h_r h_z}{4}\left(\frac{1}{r^2}(r_p, z_s)\left(\frac{r_p}{9} + \frac{2h_r}{45}\right) + \frac{1}{r^2}(r_{p+1}, z_s)\left(\frac{r_p}{9} + \right.\right.$$
$$\left.\left. + \frac{h_r}{15}\right)\right), \quad (91)$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{22} = \widehat{M}^{\frac{1}{r^2}}{}_{44} = \frac{h_r h_z}{4}\left(\frac{1}{r^2}(r_p, z_s)\left(\frac{r_p}{9} + \frac{h_r}{15}\right) + \frac{1}{r^2}(r_{p+1}, z_s)\left(\frac{r_p}{3} + \frac{4h_r}{15}\right)\right), \quad (92)$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{23} = \widehat{M}^{\frac{1}{r^2}}{}_{32} = \frac{h_r h_z}{12}\left(\frac{1}{r^2}(r_p, z_s)\left(\frac{r_p}{6} + \frac{h_r}{15}\right) + \frac{1}{r^2}(r_{p+1}, z_s)\left(\frac{r_p}{6} + \frac{2h_r}{10}\right)\right), \quad (93)$$

$$\widehat{M}^{\frac{1}{r^2}}{}_{24} = \widehat{M}^{\frac{1}{r^2}}{}_{42} = \frac{h_r h_z}{12}\left(\frac{1}{r^2}(r_p, z_s)\left(\frac{r_p}{6} + \frac{h_r}{10}\right) + \frac{1}{r^2}(r_{p+1}, z_s)\left(\frac{r_p}{2} + \frac{2h_r}{5}\right)\right). \quad (94)$$

А компоненты локальной матрицы $\widehat{M}$:

$$\widehat{M}_{11} = \widehat{M}_{33} = \frac{h_r h_z}{4}\left(\frac{4r_p}{9} + \frac{h_r}{9}\right), \quad (95)$$

$$\widehat{M}_{13} = \widehat{M}_{31} = \frac{h_r h_z}{12}\left(\frac{2r_p}{3} + \frac{h_r}{6}\right), \quad (96)$$

$$\widehat{M}_{14} = \widehat{M}_{41} = \frac{h_r h_z}{12}\left(\frac{r_p}{3} + \frac{h_r}{6}\right), \quad (97)$$

$$\widehat{M}_{21} = \widehat{M}_{43} = \widehat{M}_{34} = \widehat{M}_{12} = \frac{h_r h_z}{4}\left(\frac{2r_p}{9} + \frac{h_r}{9}\right), \quad (98)$$

$$\widehat{M}_{22} = \widehat{M}_{44} = \frac{h_r h_z}{4}\left(\frac{4r_p}{9} + \frac{h_r}{3}\right), \quad (99)$$

$$\widehat{M}_{23} = \widehat{M}_{32} = \frac{h_r h_z}{12}\left(\frac{r_p}{3} + \frac{h_r}{6}\right), \quad (100)$$

$$\widehat{M}_{24} = \widehat{M}_{42} = \frac{h_r h_z}{12}\left(\frac{2r_p}{3} + \frac{h_r}{2}\right). \quad (101)$$

1.4. ГЕНЕРАЦИЯ СЕТКИ

1.4.1. РАЗРАБОТКА ГЕНЕРАТОРА НЕРАВНОМЕРНОЙ СЕТКИ ДЛЯ СЛОИСТОЙ СРЕДЫ ДЛЯ РЕШЕНИЯ ДВУМЕРНОЙ ЗАДАЧИ

Опишем общий принцип работы генератора сетки, так как свойства искомого решения и его близость к точному решению зависят от выбора сетки [10]. На вход программе подаются по r левая и правая и по z нижняя и верхняя границы, начальный шаг разбиения сетки и коэффициент растяжения по каждой оси. Считываются нижние и верхние границы слоя, номер материалов в каждом, координата источника тока. Начиная с этой точки, генератор в соответствие с шагом и коэффициентом растяжения строит сетку влево и вправо для r и вниз и вверх для z. Таким образом получается плотная сетка в области источника тока и разреженная в удалении от него. Так как моделируется горизонтально слоистая среда, необходимо учитывать то, что некоторые разбиения по оси z должны совпадать с заданными уровнями смены слоев. То есть, если при построении сетки новый слой начинается с z = 5000, то невозможно представить сетку вида z = […, 4988, 4995, 5025, …].

Чтобы избежать такой ситуации генератор проверяет, в каком слое было создано предыдущее и новое деления. Если в разных, и новое деление не равно границе, тогда происходит перерасчет следующим образом: вычисляется шаг от предыдущего деления до границы, и если он меньше, чем шаг между предыдущим делением и предшествующим его делением (в примере, приведенном выше, это z = 4988 и z = 4995), то значение предыдущего деления (в примере z = 4995)

изменяется на значение границы, запоминается новый шаг, и с него же продолжается построение сетки в новом слое, иначе предыдущее деление не меняется, новое равно границе, и с полученным шагом продолжается построение сетки в новом слое. Аналогично проверяется выход значений делений за заданные границы осей.

1.4.2. РАЗРАБОТКА ГЕНЕРАТОРА НЕРАВНОМЕРНОЙ СЕТКИ ДЛЯ СЛОИСТОЙ СРЕДЫ ДЛЯ РЕШЕНИЯ ТРЕХМЕРНОЙ ЗАДАЧИ

Опишем общий принцип работы генератора сетки. Генерируется трехмерный объект в декартовых координатах, который располагается в некотором трехмерном электромагнитном поле. При разбиении трехмерного тела трудно наглядно представить расположение элементов в дискретной модели, поэтому, вероятно, более удобным из двух типов элементов – тетраэдров и параллелепипедов – является последний [5]. На вход программе подаются x, y, z границы объекта и трехмерной сетки, начальный шаг разбиения сетки и коэффициент растяжения по каждой оси в объекте и за его пределами. Сетка строится по всем осям с растяжением от объекта, создавая уплотнение в его области. Проверка на выход значений делений за границы осей проверяется аналогично описанному выше алгоритму генерирования двумерной сетки.

2. ОПИСАНИЕ РАЗРАБОТАННОЙ ПРОГРАММЫ

Программа получает на вход данные через набор файлов и выводит результаты решения по временным слоям двухмерной задачи – координаты точек и A_φ – в файл "res.txt", трехмерной задачи – координаты точек и $|\bar{A}|$ – в файл "result_object.txt", значения B_z нормального, аномального и полного полей в приемнике по временным слоям в файл "in_reciever.txt".

2.1. ВХОДНЫЕ ДАННЫЕ

Таблица 1 – Файлы входных данных

Наименование файла	Содержимое файла	Описание
1	2	3
"istoc.txt"	5 0	Координаты точечного источника тока r и z соответственно.
"layers.txt"	3 -10000 -10 2 -10 0 1 0 10000 0	Количество слоёв в среде, для каждого слоя координаты границ в порядке возрастания по z, номер материала.
"gen_net.txt"	0.1 10000 10000 -10000 0.1 1.1 0.1 1.1	Левые и правые границы поля по r и z, начальный шаг, коэффициент растяжения для оси по r и по z.
"material.txt"	3 0.0001256637 0.000001 0.0001256637 0.1 0.0001256637 0.01	Количество наборов материалов, для каждого набора материалов - μ и σ.

Окончание таблицы 1

1	2	3
"obj_material.txt"	0.0001256637 1000	μ и σ для объекта.
"time.txt"	30 0 1E-5 1.1	Количество временных слоев, начальное время, шаг по времени, коэффициент растяжения.
"gen_obj.txt"	10 30 1 1 10 30 1 1 -30 -10 1 1 1 -7000 7000 1.2 -7000 7000 1.2 -10000 10000 1.2	Левые и правые границы объекта, начальный шаг, коэффициент растяжения по x, y, z соответственно. Номер слоя, в котором расположен объект. Левые и правые границы трехмерного поля, коэффициент растяжения по x, y, z.
"reciever.txt"	20 20 0	Координаты x, y, z приемника.

2.2. ОПИСАНИЕ ПРОЦЕДУР И ФУНКЦИЙ ПРОГРАММЫ

Таблица 2 – Описание процедур и функций программы

Наименование подпрограммы	Назначение подпрограммы
1	2
double gamma(double r)	Вычисление $\frac{1}{r^2}$ для двухмерной задачи.
void Input()	Считывание всех необходимых данных из файлов.
void Clear_and_resize();	Очистка и инициализация векторов.

Продолжение таблицы 2

1	2
void multiply(vector<vector<double>>& A, vector<double>& x, vector<double>& res)	Умножение матрицы на вектор.
void Generate_Portrait()	Генерация портрета.
void Assemble_Locals(int el_id)	Внесение локальных матриц в глобальную СЛАУ для двухмерной задачи.
void Generate_2D_Net(string net_file)	Генерация двухмерной сетки и конечных элементов.
void Read_Layers(string obj_file)	Формирование слоистой среды.
void Get_G()	Формирование локальной матрицы G для двухмерной задачи.
void Get_M()	Формирование локальных матриц M и $M^{\frac{1}{r^2}}$ для двухмерной задачи.
void Locals(int el_id, double t);	Вызов процедур вычисления локальных матриц для двухмерной задачи.
void Find_field();	Временной цикл поиска нормального поля.
void Get_KR1(double t);	Учет первых краевых для двухмерной задачи.
int Get_Num_Layer(double x0, double y0, double x1, double y1);	Определение слоя, к которому относится конечный элемент двухмерной задачи.
double Get_solution(int time_layer,double r, double z, double variable, bool isB, bool two_layer);	Поиск A_φ, B_z нормального поля, E в точке.

Продолжение таблицы 2

1	2
void Generate_3D_Net(string net_file, bool go_to_z_up);	Генерация трехмерной сетки и конечных элементов.
void Middles();	Поиск середин векторов.
void Find_in_object();	Временной цикл поиска анномального поля.
void Locals_3D(int el_id, int time_layer, bool two_layers);	Вызов процедур вычисления локальных матриц для трехмерной задачи.
void Get_G_3D();	Формирование локальной матрицы G для трехмерной задачи.
void Get_M_3D()	Формирование локальной матрицы M для трехмерной задачи.
void Get_b_3D(int time_layer, int el_id, bool two_layers);	Формирование локального вектора правой части для трехмерной задачи.
void Assemble_Locals_3D(int el_id);	Внесение локальных матриц в глобальную СЛАУ для трехмерной задачи.
int Get_Num_Layer_3D(double z0, double z1);	Определение слоя, к которому относится конечный элемент трехмерной задачи.
double Get_solution_3D(double t, double x, double y, double z, int i, bool B);	Поиск A, B_z аномального поля в точке.
void Make_grid(bool go_to_z_up);	Формирование узлов трехмерной сетки.
void Make_edges();	Формирование ребер.
void Make_obj_elems();	Формирование конечных элементов для трехмерной задачи.
void Make_profile();	Перевод из строчно-столбцового формата в профильный.

Окончание таблицы 2

1	2
void Get_KR1_3D();	Учет первых краевых для трехмерной задачи.
void Make_centers();	Формирование центров конечных элементов трехмерной задачи.
void LOS_LU(int t);	Итерационный решатель ЛОС для двухмерной задачи.
void FactLU(vector<double>& L, vector<double>& U, vector<double>& D);	LU разложение.
void Direct(vector<double>& L, vector<double>& D, vector<double>& y, vector<double>& b);	Прямой алгоритм.
void Reverse(vector<double>& U, vector<double>& x, vector<double>& y);	Обратный алгоритм.
double Norm(vector<double>& x);	Вычисление нормы вектора.
double mult(const vector<double>& a, const vector<double>& b);	Умножение вектора на вектор.
void Ax(vector<double>& x, vector<double>& y);	Умножение вектора на матрицу решения A.
void Without_selection();	Функция для исследования решения задачи без выделения поля.
void LUDec();	LU разложение для профильной матрицы.
void Direct_prof();	Прямой алгоритм для профильной матрицы.
void Reverse_prof();	Обратный алгоритм для профильной матрицы.

3. ТЕСТИРОВАНИЕ

3.1. ТЕСТИРОВАНИЕ НА ПОРЯДОК АППРОКСИМАЦИИ ПО ПРОСТРАНСТВУ

Проведем тестирование разработанной программы на порядок аппроксимации по пространству.

Проведем тесты на небольшой трехмерной сетке, параметры которой:

$x \in [0, 5]$, $h = 1$, $d = 1$;

$y \in [0, 5]$, $h = 1$, $d = 1$;

$z \in [0, 5]$, $h = 1$, $d = 1$;

$\mu = 1$, $\sigma = 1$

Временная сетка: $t_0 = 0$, $h = 0.5$, $d = 1$, 5 временных слоёв. Для первого и второго (для трёхслойной схемы) временных слоев значения A зададим аналитически.

Результаты тестирования на полиномах приведены в таблице 3 и таблице 4.

Таблица 3 – Тестирование на порядок аппроксимации по пространству для двухслойной схемы по времени

Полином/Время	Среднеквадратичная погрешность			
	t = 0.5	t = 1	t = 1.5	t = 2
A = (5,5,5)	5.46428e-14	5.94814e-14	6.1078e-14	6.42135e-14
A = (yt, zt, xt)	1.91403e-14	3.86813e-14	5.71481e-14	8.77839e-14
A = (y^2t, z^2t, x^2t)	0.61668	0.27823	0.86405	2.61436

Таблица 4 – Тестирование на порядок аппроксимации по пространству для трёхслойной схемы по времени

Полином/Время	Среднеквадратичная погрешность			
	t = 0.5	t = 1	t = 1.5	t = 2
A = (5,5,5)	-	4.08466e-14	4.28448e-14	4.38908e-14
A = (yt, zt, xt)	-	3.14412e-14	4.34926e-14	6.50551e-14
A = (y^2t, z^2t, x^2t)	-	0.40200	1.14127	2.55285

По пространству используется трилинейный базис, следовательно, теоретический порядок аппроксимации – первый. Практически для двухслойной и трёхслойной временных схем также получен первый порядок аппроксимации по пространству.

3.2. ТЕСТИРОВАНИЕ НА ПОРЯДОК СХОДИМОСТИ ПО ПРОСТРАНСТВУ

Подробим сетку по пространству в 2 раза и 4 раза и определим порядок сходимости по погрешности во внутренних точках, используя трёхслойную временную схему.

Таблица 5 – Тестирование на порядок сходимости по пространству

Среднеквадратичная погрешность для A = (y^2t, z^2t, x^2t)			
Время	t = 1	t = 1.5	t = 2
h = 1	0.40200	1.14127	2.55285
h = 0.5	0.00768	0.53663	1.74683
h = 0.25	0.00191	0.54633	1.75306

Следовательно, порядок сходимости по пространству $k = \log_2 \frac{||A^* - A_h||}{||A^* - A_{\frac{h}{2}}||} = \log_2 \frac{0.00768}{0.00191} \approx 2$.

3.3. ТЕСТИРОВАНИЕ НА ПОРЯДОК АППРОКСИМАЦИИ ПО ВРЕМЕНИ

Проведем тестирование разработанной программы на порядок аппроксимации по времени.

Параметры сетки:

x ∈ [-100, 100], h = 1, d = 1.2;

y ∈ [-100, 100], h = 1, d = 1.2;

z ∈ [-100, 100], h = 1, d = 1.2;

С областью внутри:

x ∈ [10, 30], h = 1, d = 1

y ∈ [10, 30], h = 1, d = 1

z ∈ [-30, -10], h = 1, d = 1

μ = 1, σ = 1000

Среда имеет слоистую структуру, параметры слоёв:

L1 ∈ [-100, -10], μ = 4π10e-6, σ = 0.01

L2 ∈ [-10, 0], μ = 4π10e-6, σ = 0.1

L3 ∈ [0, 100], μ = 4π10e-6, σ = 0.000001

Основная временная сетка имеет шаг по времени ht = 0.001, коэффициент растяжения d = 1.3. При дроблении сетки в 2 раза $d_{1/2} = \sqrt{d}$, $ht_{1/2} = \frac{ht}{1+d_{1/2}}$, при дроблении сетки в 4 раза $d_{1/4} = \sqrt[4]{d}$, $ht_{1/4} = \frac{ht}{1+d_{1/4}+d_{\frac{1}{4}}^2+d_{1/4}^3}$.

Результаты тестирования на порядок аппроксимации по времени приведены на рисунке 2, рисунке 3 и рисунке 4.

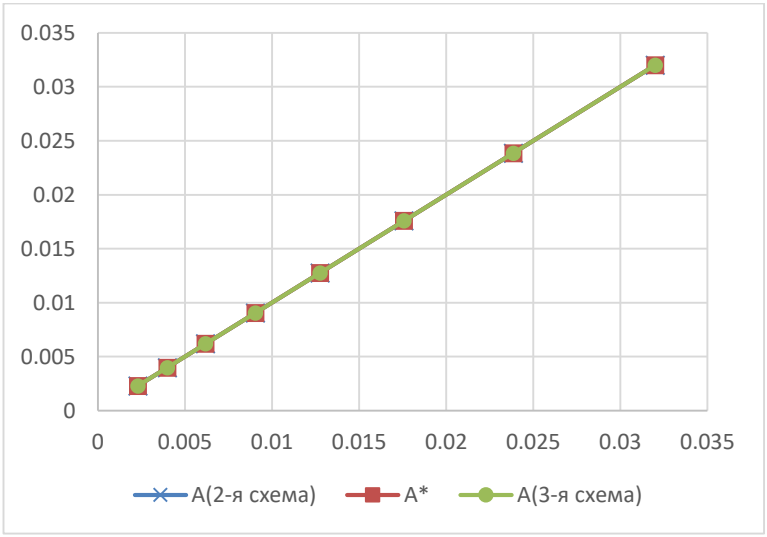

Рисунок 2 – A* = t

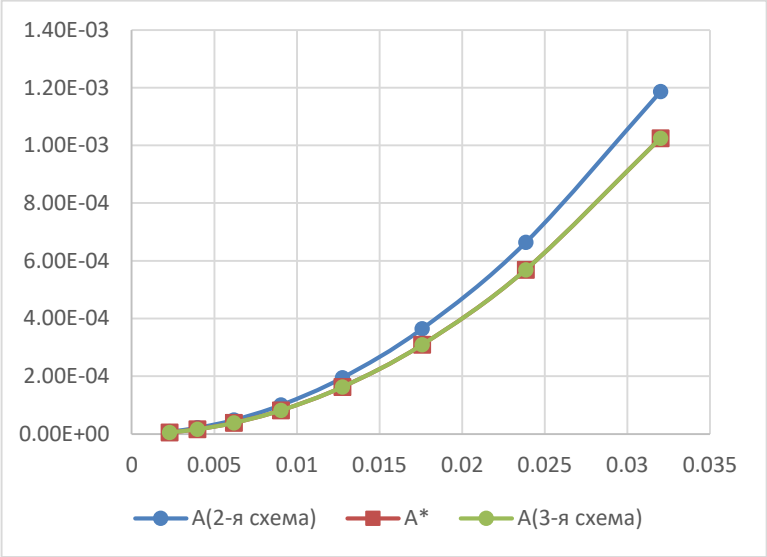

Рисунок 3 – $A^* = t^2$

Порядок аппроксимации двухслойной схемы = 1, что совпадает с теоретическим порядком аппроксимации.

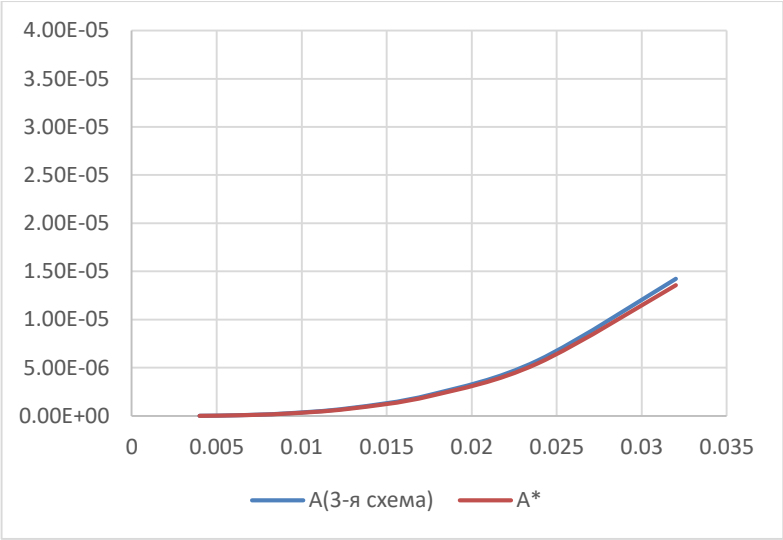

Рисунок 4 – $A^* = t^3$

Порядок аппроксимации трёхслойной схемы = 2, что совпадает с теоретическим порядком аппроксимации.

3.4. ТЕСТИРОВАНИЕ НА ПОРЯДОК СХОДИМОСТИ ПО ВРЕМЕНИ

Подробим сетку по времени в 2 раза и 4 раза и определим порядок сходимости двухслойной схемы по времени. Результаты отображены на рисунке 5.

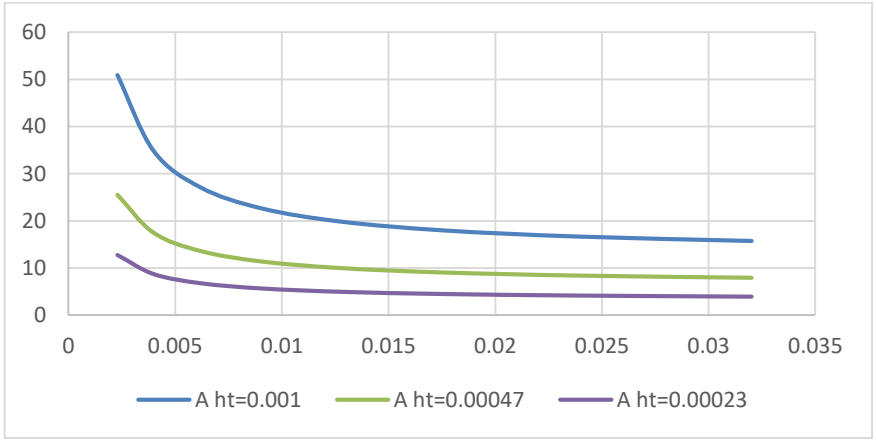

Рисунок 5 – Относительная погрешность при $A^* = t^2$ (%)

При дроблении сетки в 2 раза, погрешность падает также в 2 раза. Следовательно, порядок сходимости двухслойной схемы = 1.

Подробим сетку по времени в 2 раза и 4 раза и определим порядок сходимости трёхслойной схемы по времени. Результаты отображены на рисунке 6.

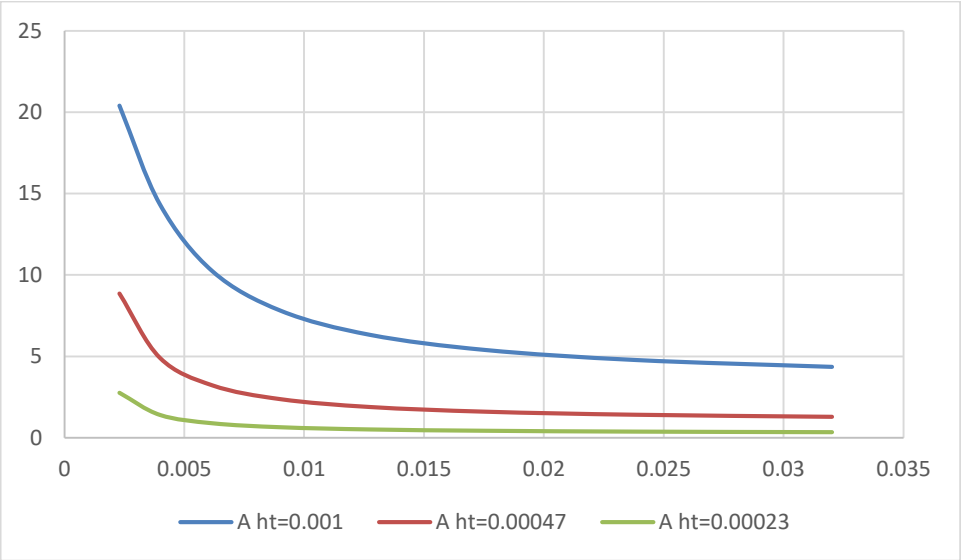

Рисунок 6 – Относительная погрешность при $A^* = t^3$ (%)

При дроблении сетки в 2 раза, погрешность падает в 3.7 раза. Следовательно, порядок сходимости трёхслойной схемы = 2.

4. ИССЛЕДОВАНИЯ

4.1. НОРМАЛЬНОЕ ПОЛЕ

Параметры среды:

r ∈ [0.1, 10000], h = 0.1, d = 1.1

z ∈ [-10000, 10000], h = 0.1, d = 1.1

Источник I (5, 0) с плотностью тока $J_{\varphi} = \frac{1}{4\pi R_{\text{источника}}}$.

Параметры слоёв:

L1 ∈ [-10000, -10], μ = 4π10e-6, σ = 0.01

L2 ∈ [-10, 0], μ = 4π10e-6, σ = 0.1

L3 ∈ [0, 10000], μ = 4π10e-6, σ = 0.000001

Временная шкала начинается с 0 с шагом 1e-05, с коэффициентом растяжения 1.1.

Верификация решения была проведена при помощи дробления начального шага сетки по пространству в 2 раза. Средняя относительная погрешность решения составила менее 2%.

Приведем на рисунках 7-12 иллюстрации (созданные при помощи программного пакета Surfer [6]) поля, полученного в результате решения двумерной задачи на некоторых временных слоях:

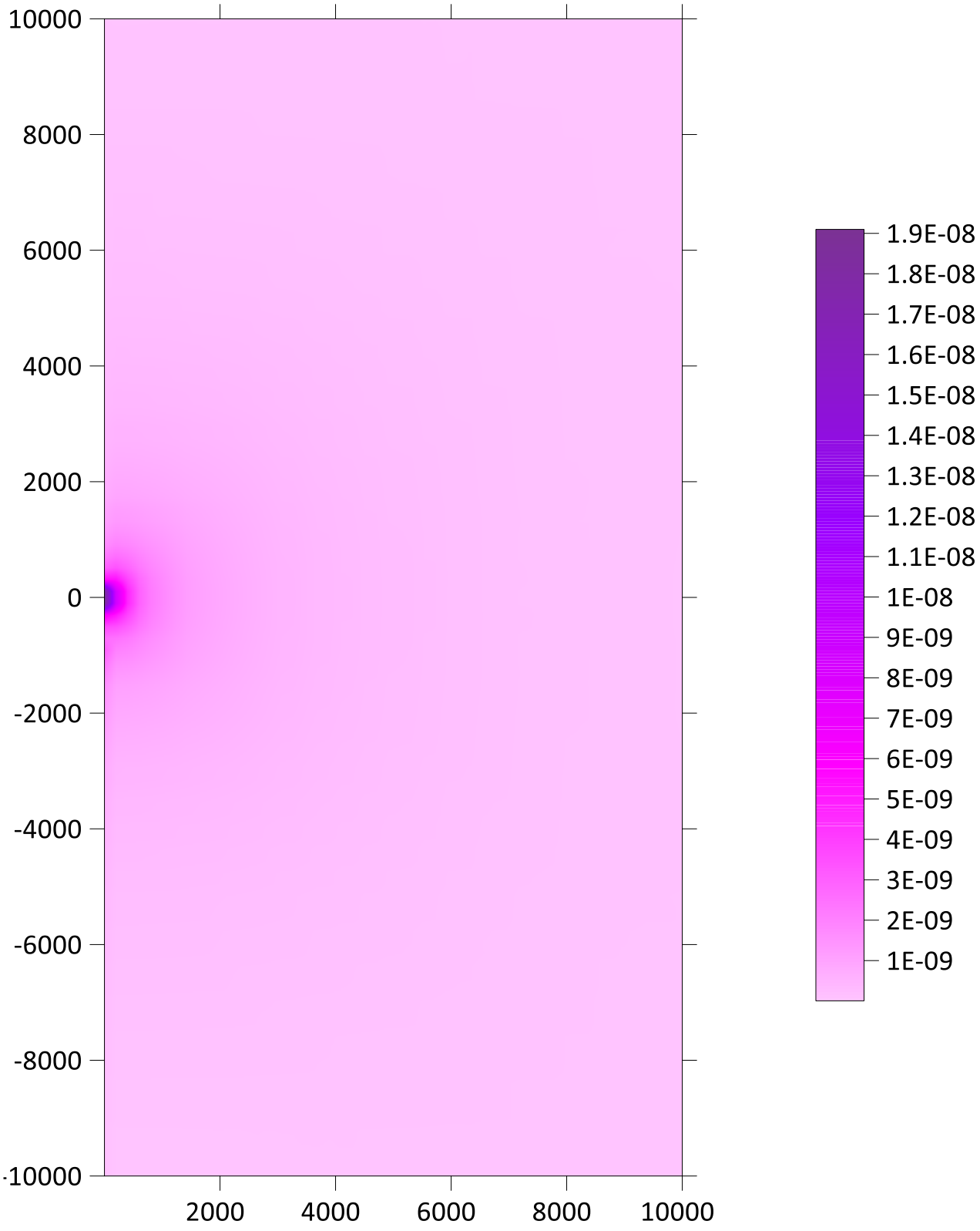

Рисунок 7 – t = 1e-05

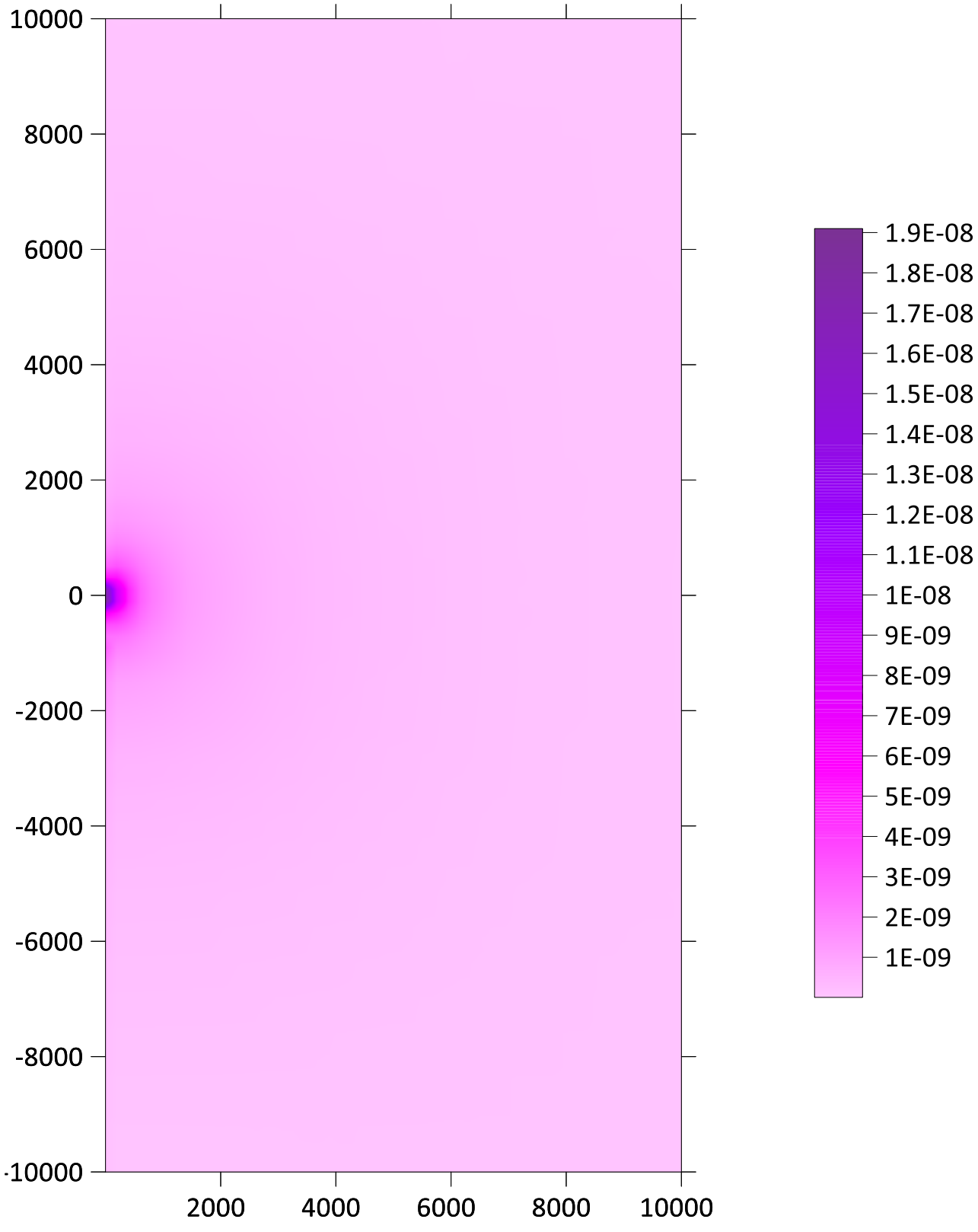

Рисунок 8 – t = 0.00015937424601

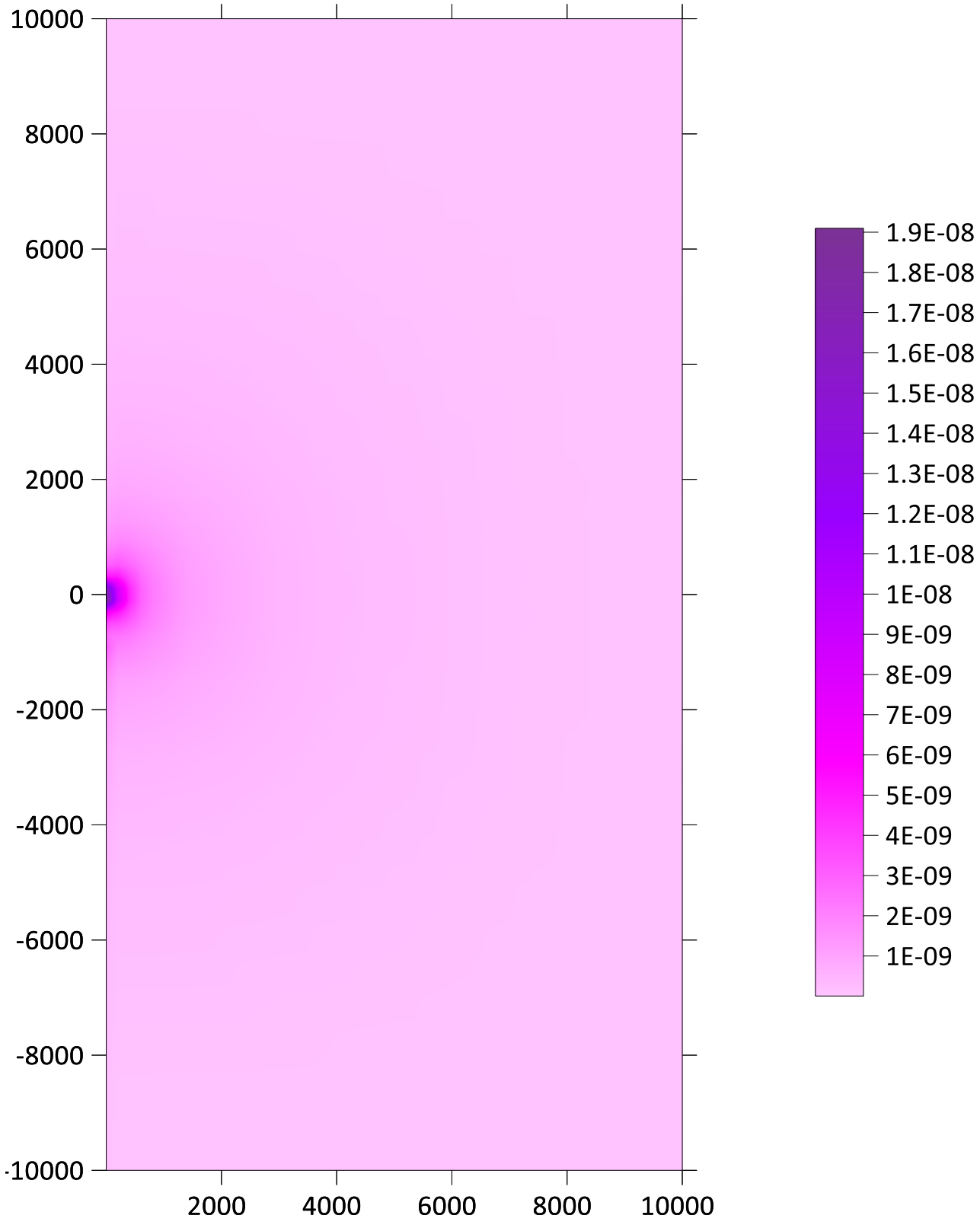

Рисунок 9 – t = 0.00057275

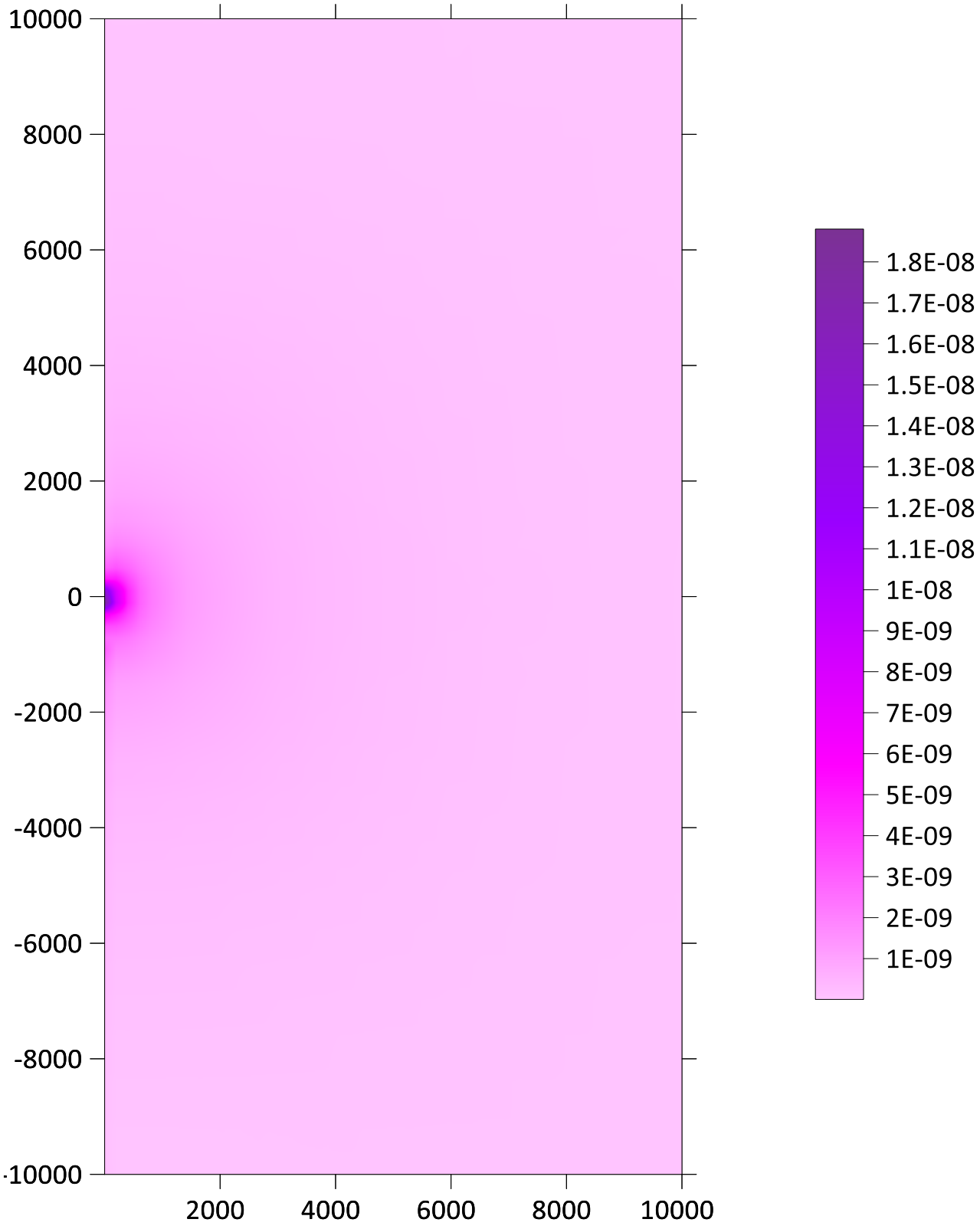

Рисунок 10 – t = 0.00164494022688864

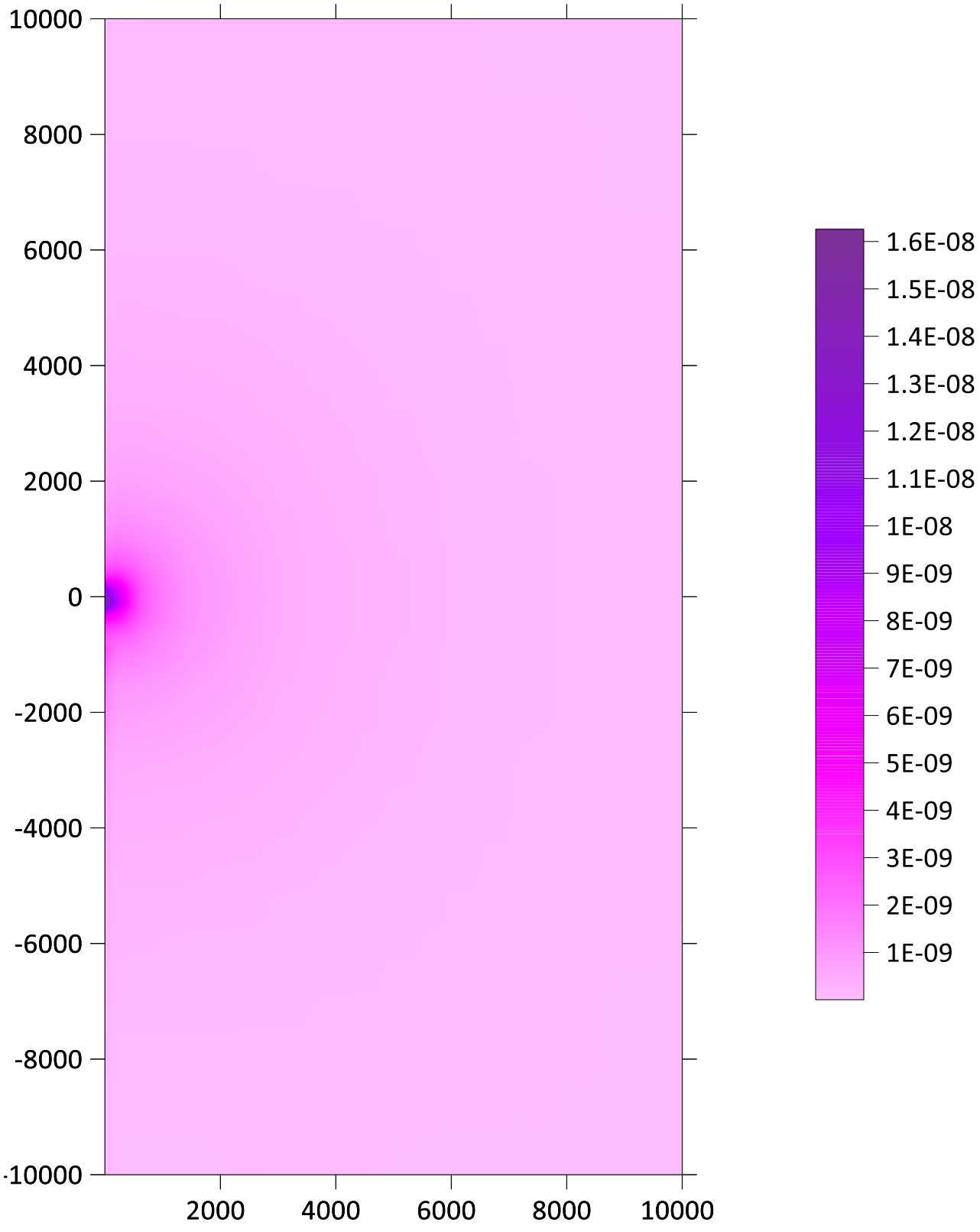

Рисунок 11 – t = 0.0044259255568176

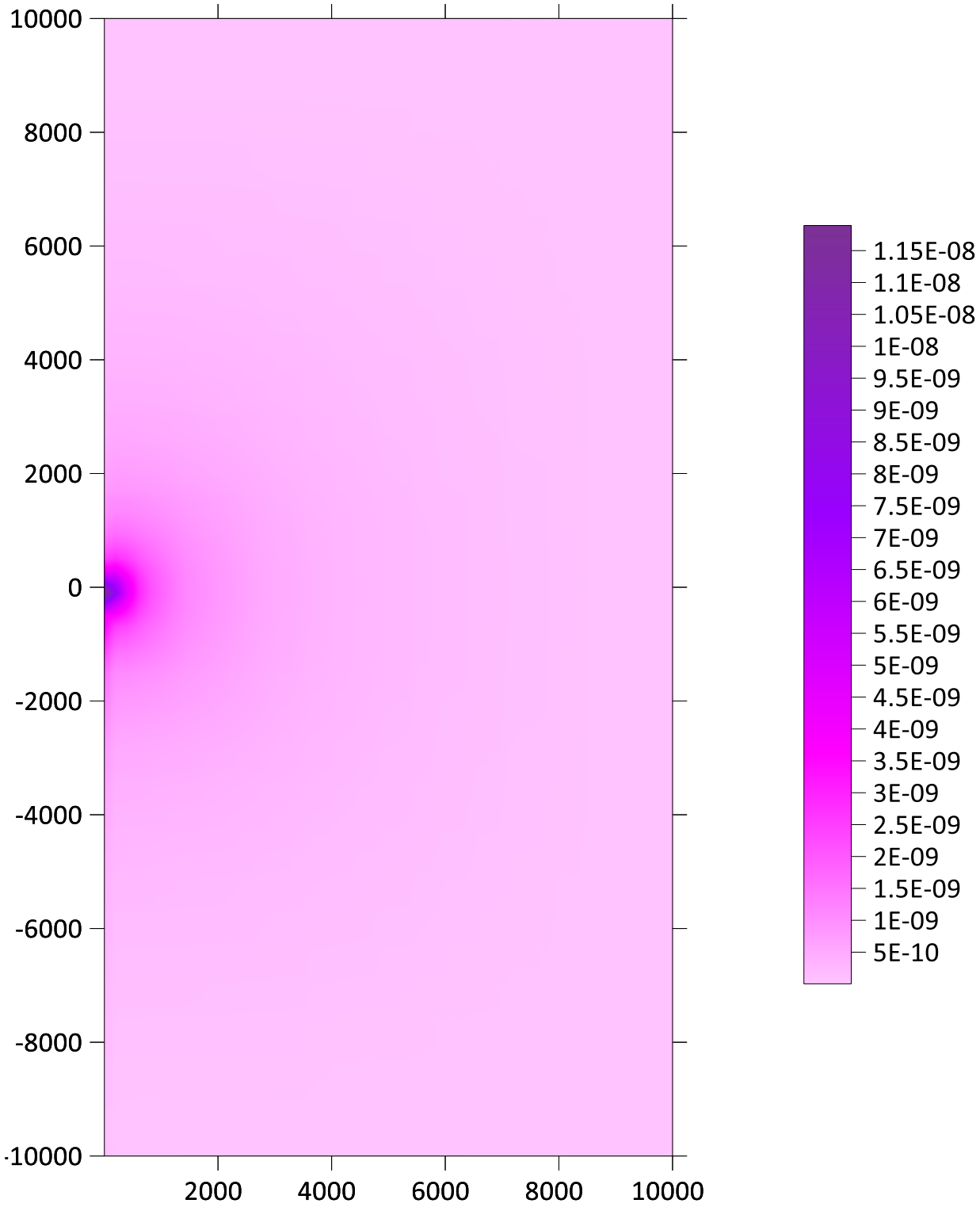

Рисунок 12 – t = 0.0116390852879695

По этим изображениям поля видно, как со временем уменьшается и распространяется поле.

4.2. АНОМАЛЬНОЕ ПОЛЕ

Опишем трехмерную сетку.

Параметры объекта:

$x \in [10, 30]$, $h = 1$, $d = 1$

$y \in [10, 30]$, $h = 1$, $d = 1$

$z \in [-30, -10]$, $h = 1$, $d = 1$

$\mu = 4\pi 10e{-6}$, $\sigma = 1000$

Параметры среды:

$x \in [-7000, 7000]$, $h = 1$, $d = 1.2$

$y \in [-7000, 7000]$, $h = 1$, $d = 1.2$

$z \in [-10000, 10000]$, $h = 1$, $d = 1.2$

Параметры слоёв:

$L1 \in [-10000, -10]$, $\mu = 4\pi 10e{-6}$, $\sigma = 0.01$

$L2 \in [-10, 0]$, $\mu = 4\pi 10e{-6}$, $\sigma = 0.1$

$L3 \in [0, 10000]$, $\mu = 4\pi 10e{-6}$, $\sigma = 0.000001$

Приемник P (20, 20, 0).

Верификация решения была проведена при помощи дробления начального шага сетки по пространству в 2 раза. Средняя относительная погрешность решения при проявлении аномального поля составила менее 5%.

Посмотрим на значения $|\vec{\mathbf{A}}|$ в центрах конечных элементов трехмерной задачи при срезе по y по временным слоям в центре объекта и вблизи него на некоторых временных слоях на рисунках 13-18:

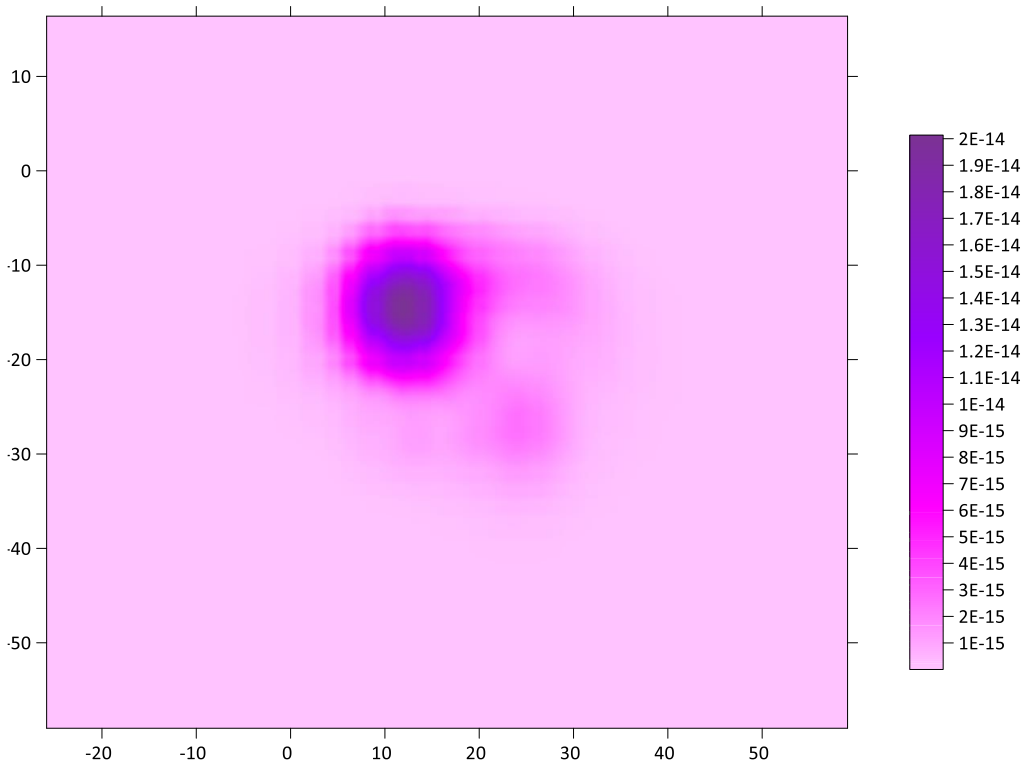

Рисунок 13 – t = 1e-05

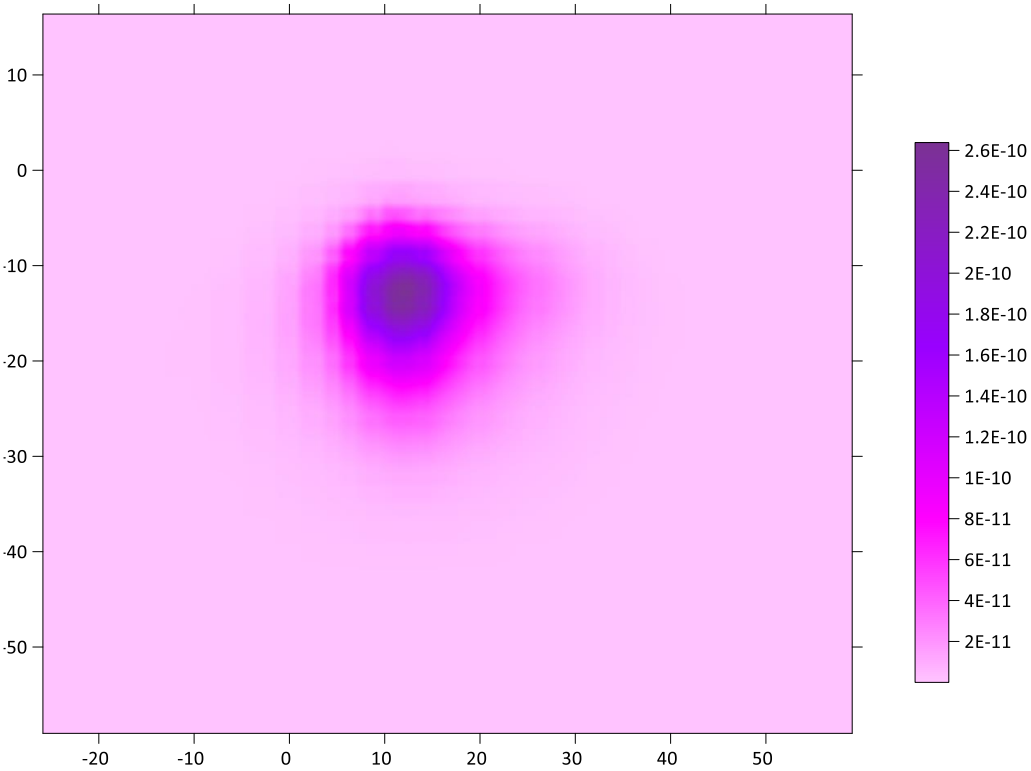

Рисунок 14 – t = 0.00015937424601

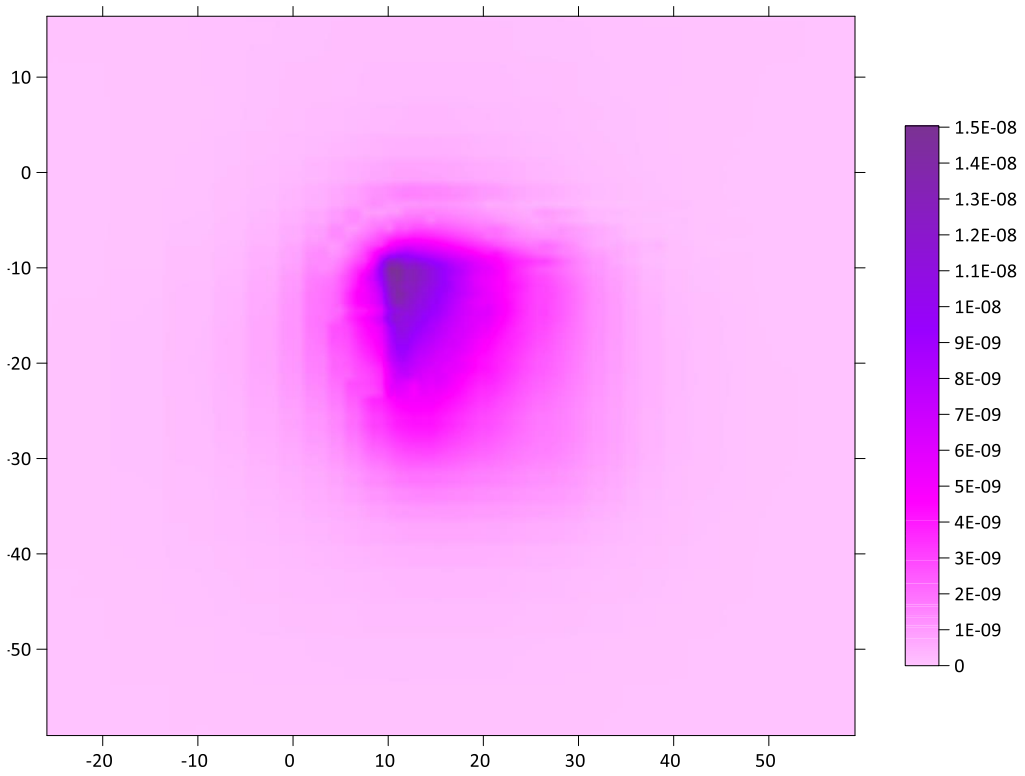

Рисунок 15 – t = 0.00057274999493256

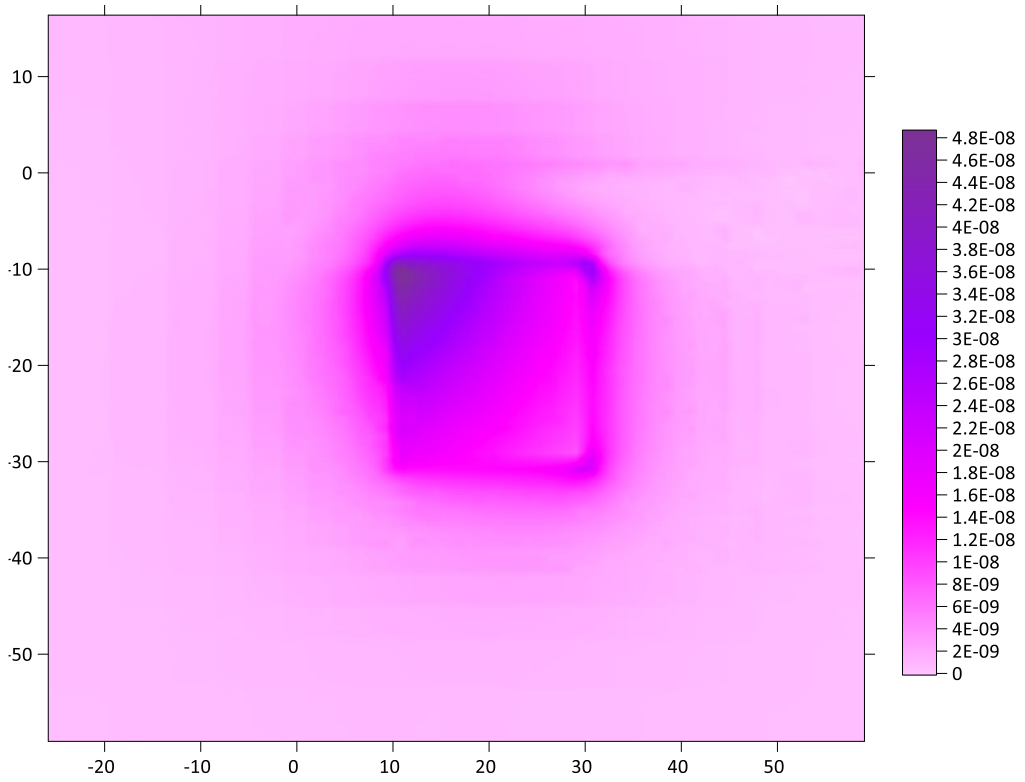

Рисунок 16 – t = 0.00164494022688864

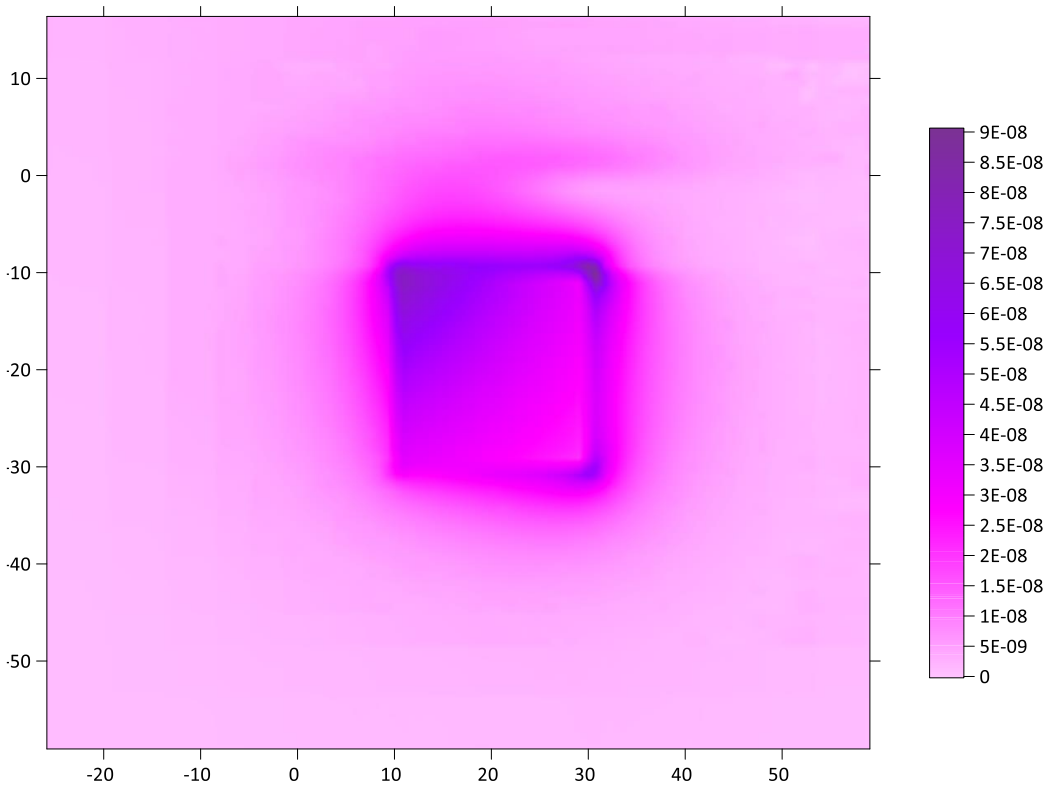

Рисунок 17 – t = 0.0044259255568176

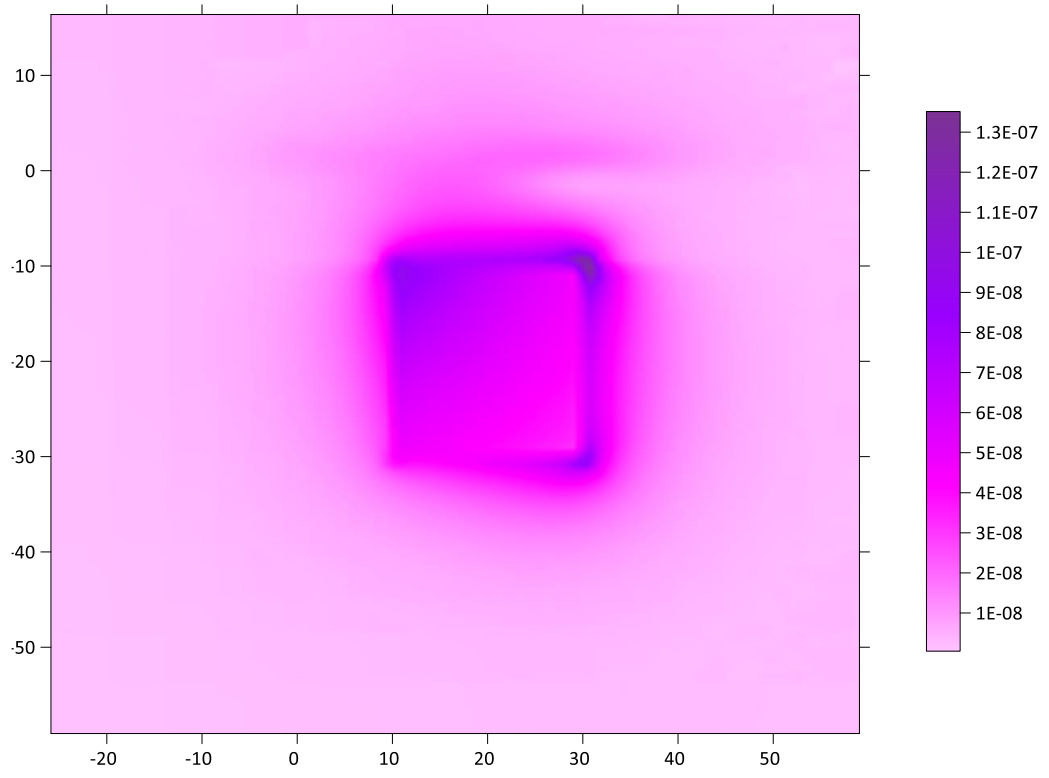

Рисунок 18 – t = 0.0116390852879695

4.3. ПОЛНОЕ ПОЛЕ

Выведем значения B_z в приемнике в логарифмической шкале по времени на рисунке 19:

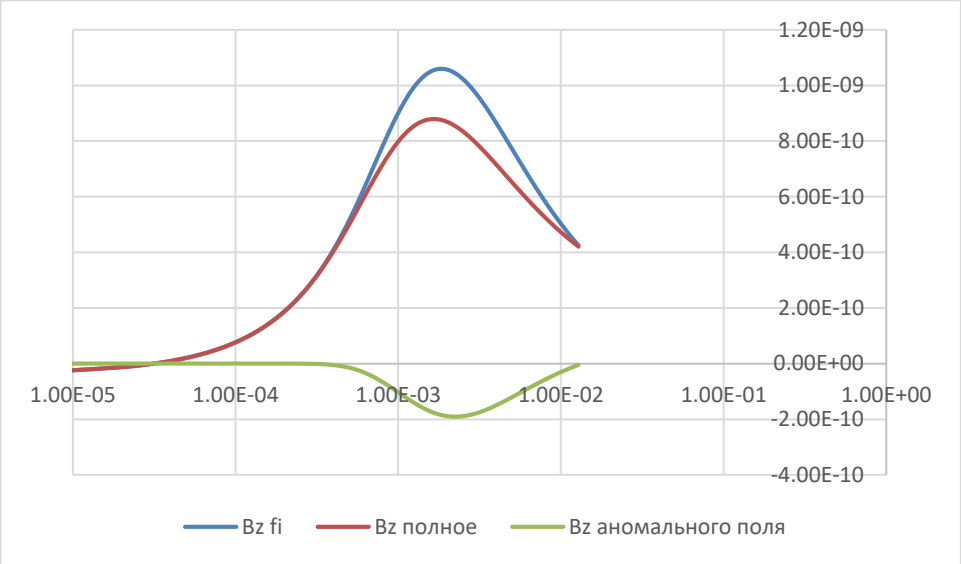

Рисунок 19 – B_z

Выведем значения $|B_z|$ в приемнике в логарифмической шкале по времени и самим значениям на рисунке 20:

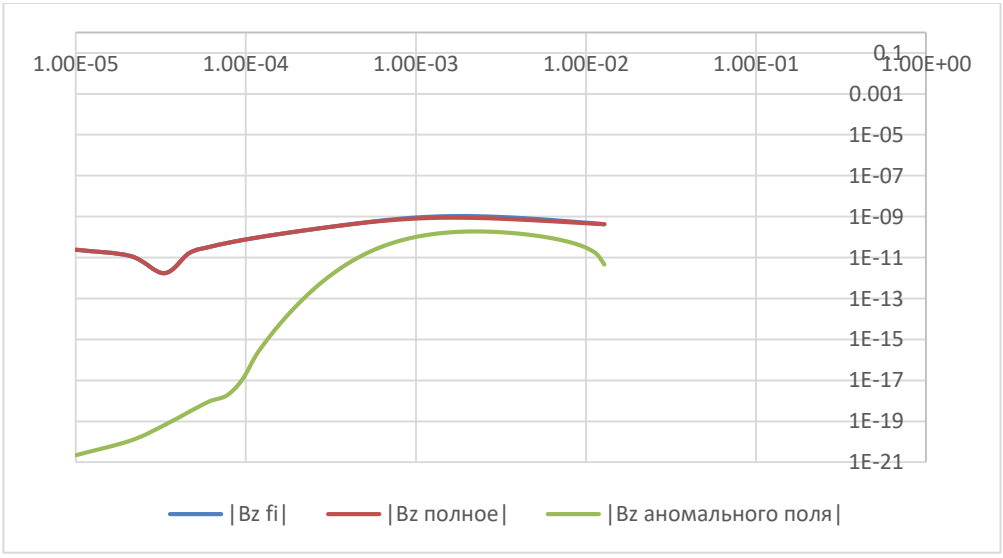

Рисунок 20 – $|B_z|$

B_z вычислялось следующим образом:

$$B_z = (\mathrm{rot}\,\vec{\mathbf{A}}_{3D})_z = \frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y} = \frac{\partial \sum_{j=5}^{8} q_j \vec{\psi}_j}{\partial x} - \frac{\partial \sum_{j=1}^{4} q_j \vec{\psi}_j}{\partial y} = \frac{\sum_{j=5}^{8} q_j \, \partial \vec{\psi}_j}{\partial x} - \frac{\sum_{j=1}^{4} q_j \, \partial \vec{\psi}_j}{\partial y}, \quad (102)$$

где

$$\partial \hat{\psi}_1(x,y,z) = \begin{pmatrix} \frac{-Z_1}{h_y} \\ 0 \\ 0 \end{pmatrix}, \quad (103)$$

$$\partial \hat{\psi}_2(x,y,z) = \begin{pmatrix} \frac{Z_1}{h_y} \\ 0 \\ 0 \end{pmatrix}, \quad (104)$$

$$\partial \hat{\psi}_3(x,y,z) = \begin{pmatrix} \frac{-Z_2}{h_y} \\ 0 \\ 0 \end{pmatrix}, \quad (105)$$

$$\partial \hat{\psi}_4(x,y,z) = \begin{pmatrix} \frac{Z_2}{h_y} \\ 0 \\ 0 \end{pmatrix}, \quad (106)$$

$$\partial \hat{\psi}_5(x,y,z) = \begin{pmatrix} 0 \\ \frac{-Z_1}{h_x} \\ 0 \end{pmatrix}, \quad (107)$$

$$\partial \hat{\psi}_6(x,y,z) = \begin{pmatrix} 0 \\ \frac{Z_1}{h_x} \\ 0 \end{pmatrix}, \quad (108)$$

$$\partial \hat{\psi}_7(x,y,z) = \begin{pmatrix} 0 \\ \frac{-Z_2}{h_x} \\ 0 \end{pmatrix}, \quad (109)$$

$$\partial\widehat{\psi}_4(x,y,z) = \begin{pmatrix} \frac{Z_2}{h_y} \\ 0 \\ 0 \end{pmatrix}, \quad (110)$$

$$\partial\widehat{\psi}_8(x,y,z) = \begin{pmatrix} 0 \\ \frac{Z_2}{h_x} \\ 0 \end{pmatrix}, \quad (111)$$

$$B_{z\varphi} = (\mathrm{rot}\, A_\varphi)_z = \frac{\partial(rA_\varphi)}{r\,\partial r} = \frac{1}{r}\left(A_\varphi + r\frac{\partial A_\varphi}{\partial r}\right) = \frac{A_\varphi}{r} + \frac{\partial A_\varphi}{\partial r} = \frac{\sum_{j=1}^{4} q_j\psi_j}{r} + \frac{\sum_{j=1}^{4} q_j\,\partial\psi_{j_i}}{\partial r}, \quad (112)$$

где одномерные линейные функции на $[r_p, r_{p+1}]$, $[z_s, z_{s+1}]$ имеют вид:

$$R_1(r) = \frac{r_{p+1}-r}{h_r}, \quad (113)$$

$$R_2(r) = \frac{r-r_p}{h_r}, \quad (114)$$

$$Z_1(z) = \frac{z_{s+1}-z}{h_z}, \quad (115)$$

$$Z_2(z) = \frac{z-z_s}{h_z}, \quad (116)$$

где $h_r = r_{p+1} - r_p$, $h_z = z_{s+1} - z_s$.

Тогда локальные базисные функции запишем как:

$$\widehat{\psi}_1(r,z) = \frac{-Z_1(z)}{h_r}, \quad (117)$$

$$\widehat{\psi}_2(r,z) = \frac{Z_1(z)}{h_r}, \quad (118)$$

$$\widehat{\psi}_3(r,z) = \frac{-Z_2(z)}{h_r}, \quad (119)$$

$$\widehat{\psi}_4(r,z) = \frac{Z_2(z)}{h_r}. \quad (120)$$

4.4. ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ВРЕМЕННЫХ СХЕМ

Выведем значения относительной погрешности (в процентах), полученные при решении двумерной задачи двухслойной схемой, на рисунке 21:

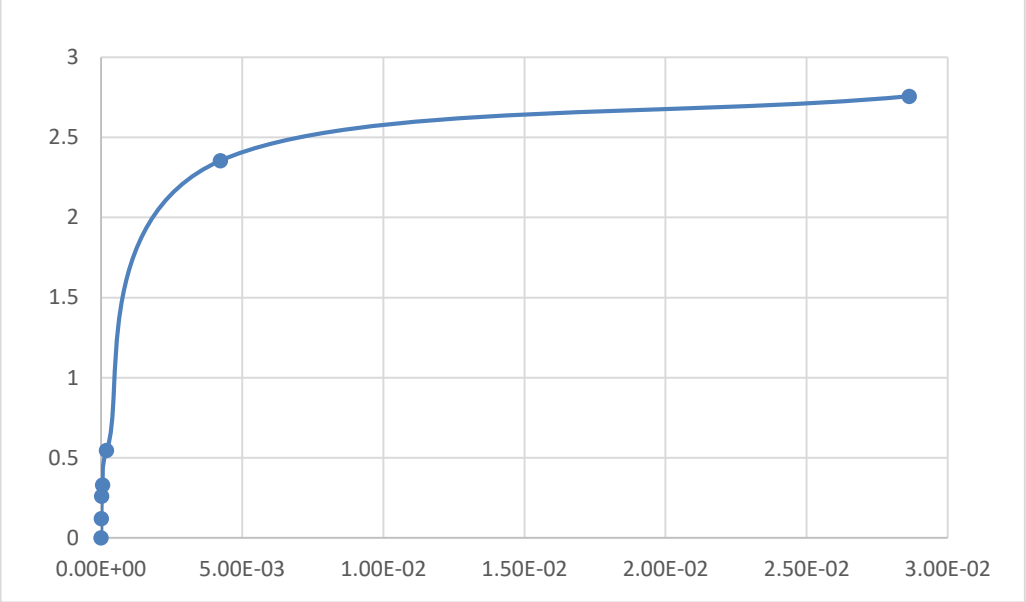

Рисунок 21 – Средняя относительная погрешность решения двумерной задачи двухслойной схемой относительной трехслойной схемы по времени

Посмотрим, как зависит погрешность от шага по времени для решения двумерной задачи двухслойной схемой по времени, на рисунке 22:

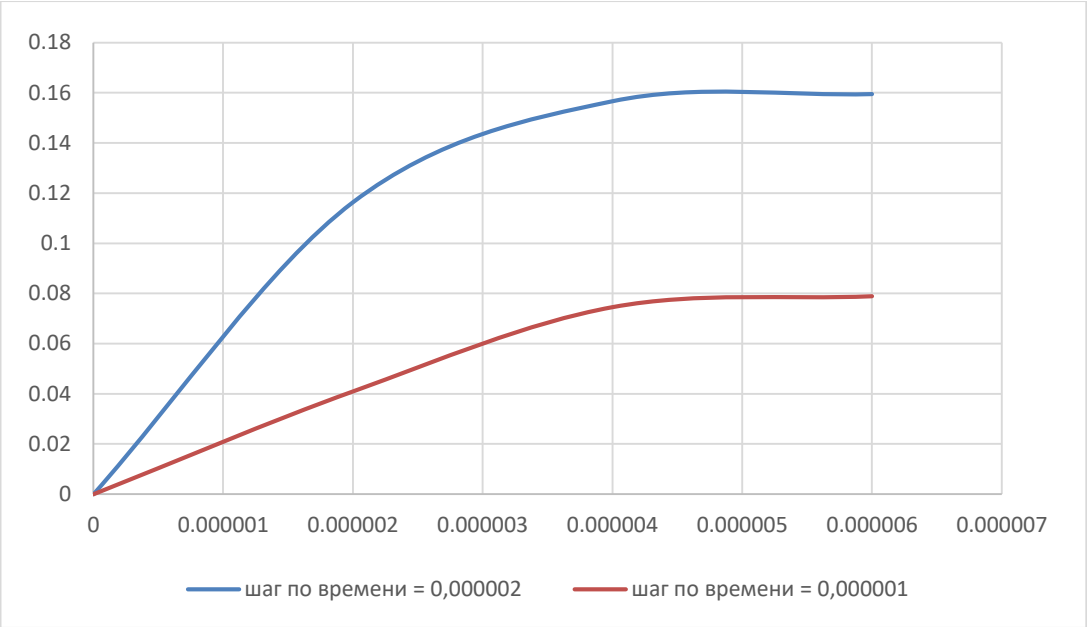

Рисунок 22 – Средняя относительная погрешность решения двумерной задачи двухслойной схемой относительной трехслойной схемы по времени

Таким образом, можно сделать вывод, что для двумерной задачи, решая двухслойной схемой, уменьшение шага по времени ведет к приближению к результату, полученному трехслойной схемой.

Выведем значения относительной погрешности (в процентах), полученные при решении трехмерной задачи двухслойной схемой на рисунке 23:

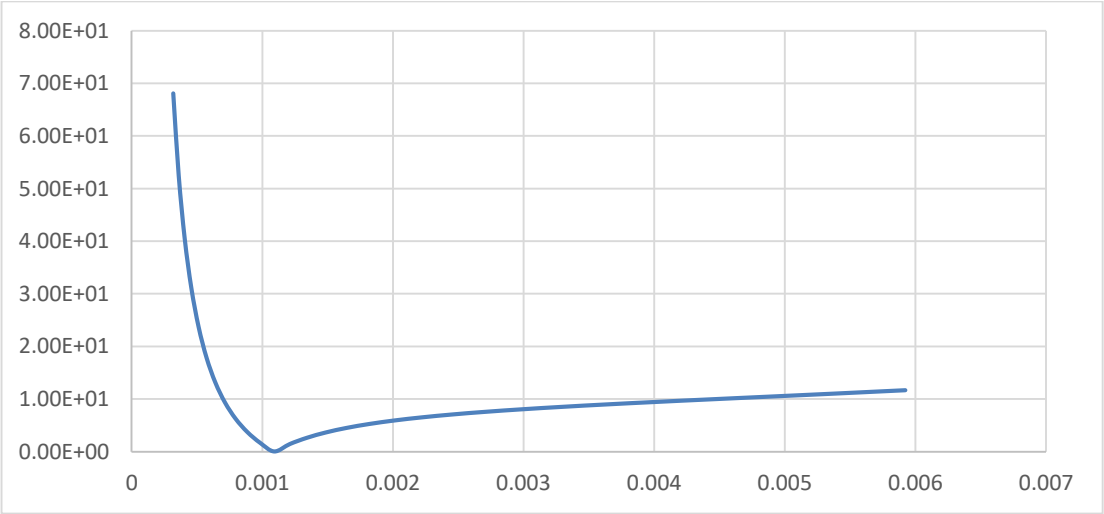

Рисунок 23 – Средняя относительная погрешность решения трехмерной задачи двухслойной схемой относительной трехслойной схемы по времени

Большая погрешность наблюдается при очень малых (почти нулевых) значениях аномального поля (до проявления аномалии), затем наблюдается медленно возрастающая низкая погрешность решения, полученного двухмерной схемой, относительно решения, полученного трехмерной схемой по времени.

Посмотрим, как зависит погрешность от шага по времени для решения трехмерной задачи двухслойной схемой по времени на рисунке 24:

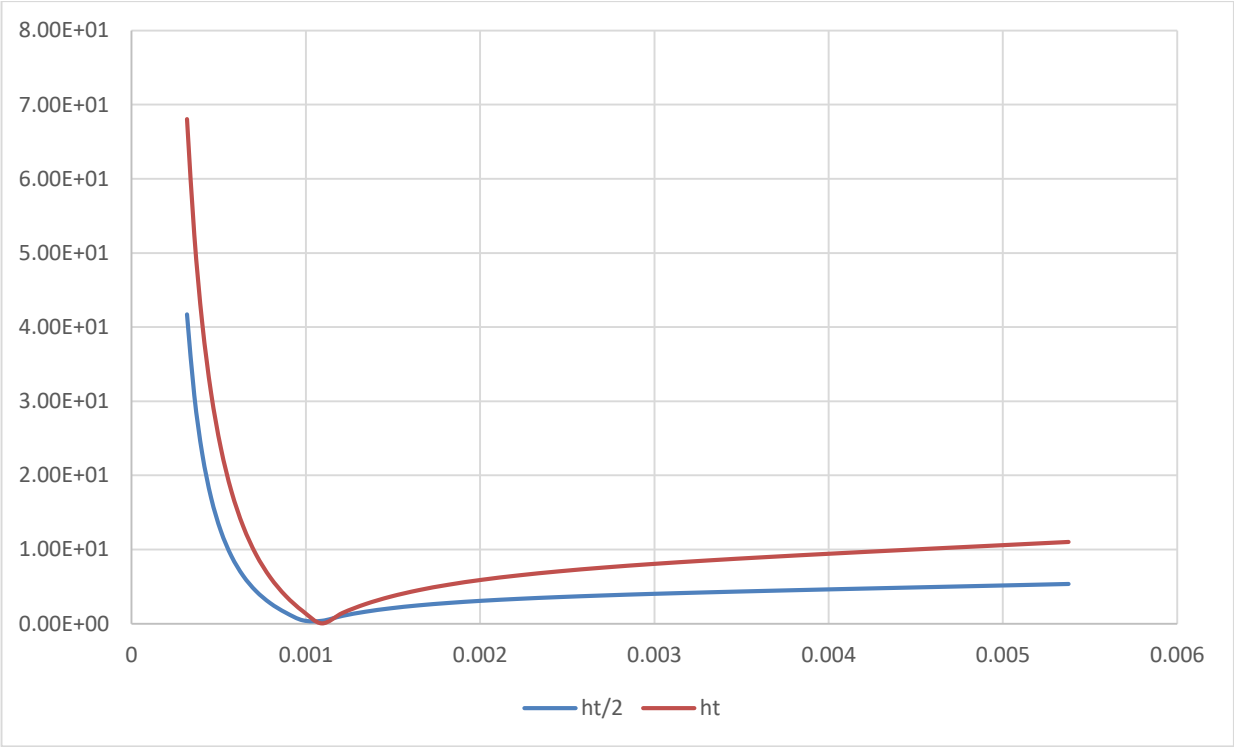

Рисунок 24 – Средняя относительная погрешность решения трехмерной задачи двухслойной схемой относительной трехслойной схемы по времени

Таким образом, можно сделать вывод, что для трехмерной задачи, решая двухслойной схемой, уменьшение шага по времени ведет к приближению к результату, полученному трехслойной схемой.

Погрешность полученного решения двухмерной и трехмерной задач падает приблизительно в 2 раза при дроблении шага в 2 раза, что согласуется с первым порядком сходимости временной двухслойной схемы.

4.5. РЕШЕНИЕ ЗАДАЧИ БЕЗ ВЫДЕЛЕНИЯ ПОЛЯ

Проведем исследование, в котором на трехмерной сетке зададим и индукционную петлю в виде квадрата, с центром в нуле координат и площадью равной площади круга, образуемого петлей в цилиндрических координатах, и объект.

На первом временном слое будем решать

$$\frac{1}{\mu}\mathrm{rot}\,\mathrm{rot}\,\vec{\mathbf{A}} = \vec{\mathrm{J}}, \tag{121}$$

затем

$$\frac{1}{\mu}\mathrm{rot}\,\mathrm{rot}\,\vec{\mathbf{A}} + \sigma\frac{\partial\vec{\mathbf{A}}}{\partial \mathrm{t}} = 0. \tag{122}$$

Параметры трехмерного поля идентичны параметрам полей, приведенных в предыдущих исследованиях, на всей внешней границе заданы первые нулевые краевые условия.

Получаем длину стороны квадрата $\mathrm{l} \approx 8.86$. Разобьем каждую сторону на 25 отрезков, определим центр каждого отрезка и конечный элемент, в котором он расположен, и внесем соответствующий вклад в вектор правой части СЛАУ:

$$\vec{\mathrm{J}} = \int \mathrm{J}\overline{\psi_{\mathrm{s}}}\mathrm{dS}. \tag{123}$$

В результате сформировалась СЛАУ, не поддающаяся решению обычными итерационными методами решения (ЛОС, МСГ), поэтому формат хранения матрицы был переведен из строчно-столбцового в профильный. Попытка получить решение при помощи прямого метода решения СЛАУ LU не привела к результату в связи с огромными затратами памяти. Также был проведен поиск решения при помощи улучшенного итерационного решателя. Из-за очень долгого итерационного процесса решение получено не было.

ЗАКЛЮЧЕНИЕ

Таким образом, был реализован метод моделирования трехмерных нестационарных электромагнитных полей в проводящих средах, порождаемых индукционным витком. При помощи реализованной программы было получено решение параболической начально-краевой задачи методом конечных элементов в декартовой системе координат с использованием векторных базисных функций первого порядка на параллелепипедах и трехслойной неявной схемы для аппроксимации по времени. Также было найдено решение для двухмерной краевой задачи для петли в цилиндрической системе координат, где скалярные базисные функции билинейные на прямоугольниках, а аппроксимация по времени – трехслойная неявная схема.

Проведенные исследования демонстрируют полученные решения - нормальное и аномальное поля.

СПИСОК ЛИТЕРАТУРЫ

1. Griffiths D.H., King R.F. Applied Geophysics for Geologists and Engineers. The elements of Geophysical Prospecting – 1981. – P. 1-5.
2. Griffiths D.H., King R.F. Applied Geophysics for Geologists and Engineers. The elements of Geophysical Prospecting – 1981. – P. 112-124.
3. Lillie R.J. Whole earth geophysics: an introductory textbook for geologists and geophysicists - Prentice Hall Upper Saddle River, New Jersey 07458, 1999. – P. 293-296.
4. Reynolds J.M. An Introduction to Applied and environmental Geophysics – 2011. P. 1-3.
5. Segerlind L.J. Applied Finite Element Analysis – Michigan State University, 1976. – P. 19.
6. Иванова И.А., Чеканцев В.А. Решение геологических задач с применением программного Р47 пакета Surfer: практикум для выполнения учебно-научных работ студентами направления «Прикладная геология» – Томск: Изд-во Томского политехнического университета, 2008. – С. 5-14.
7. Персова М.Г., Рояк М.Э., Соловейчик Ю.Г. Метод конечных элементов для решения скалярных и векторных задач – Новосибирск: НГТУ, 2007. – С. 743-772.
8. Персова М.Г., Рояк М.Э., Соловейчик Ю.Г. Метод конечных элементов для решения скалярных и векторных задач – Новосибирск: НГТУ, 2007. – С. 401-414.
9. Рояк М.Э., Соловейчик Ю.Г., Шурина Э.П. Сеточные методы решения краевых задач математической физики – Новосибирск: НГТУ, 1998. – С. 113-114.
10. Самарский А.А. Теория разностных схем – Москва, 1997. – С. 68.

ПРИЛОЖЕНИЕ

```
#include <fstream>
#include <iostream>
#include <vector>
#include <iomanip>
#include <set>
#include <algorithm>
#include <iterator>

using namespace std;

struct node //структура узла для 2D сетки
{
	double r;
	double z;
};

struct node_3D //структура узла для 3D сетки
{
	double x;
	double y;
	double z;
};

struct edge // структура ребра
{
	node_3D beg, end, middle;
};

struct otrez // структура ребра
{
	node_3D beg, end, middle;
	double l;
};


struct material
{
	double mu;
	double lambda;
	double sigma;
};

struct element
{
	vector<int> num; //локальные узлы
	int mater;
};

struct layer {
	double y0, y1;
	unsigned int n_mat;
};

static const double G1[4][4] =
{
	{2.0, 1.0, -2.0, -1.0},
	{1.0, 2.0, -1.0, -2.0},
	{-2.0, -1.0, 2.0, 1.0},
	{-1.0, -2.0, 1.0, 2.0}
};
```

```
static const double G2[4][4] =
{
	{2.0, -2.0, 1.0, -1.0},
	{-2.0, 2.0, -1.0, 1.0},
	{1.0, -1.0, 2.0, -2.0},
	{-1.0, 1.0, -2.0, 2.0}
};

static const double G3[4][4] =
{
	{-2.0, 2.0, -1.0, 1.0},
	{-1.0, 1.0, -2.0, 2.0},
	{2.0, -2.0, 1.0, -1.0},
	{1.0, -1.0, 2.0, -2.0}
};

static const double G3T[4][4] =
{
	{-2.0, -1.0, 2.0, 1.0},
	{2.0, 1.0, -2.0, -1.0},
	{-1.0, -2.0, 1.0, 2.0},
	{1.0, 2.0, -1.0, -2.0}
};

static const double M[4][4] =
{
	{4.0, 2.0, 2.0, 1.0},
	{2.0, 4.0, 1.0, 2.0},
	{2.0, 1.0, 4.0, 2.0},
	{1.0, 2.0, 2.0, 4.0}
};

vector<otrez> kvadro;
vector<node> nodes;          // все узлы в порядке глобальной нумерации
vector<element> elems;       // все элементы в прядке глобальной нумерации
vector<material> materials; // все материалы по индексам
vector<int>KR1;    // краевые 1 рода
vector<int> KR1_3D;    // краевые 1 рода
vector<layer> layers;        //слои в среде
vector<edge> edges;          // все ребра в порядке глобальной нумерации
vector<element> obj_elems;      //элементы 3-ой задачи

class KURS
{
public:
	vector<double> di, al, au, b_loc, b, time, M2, M1, temp, p, z, r, Ar, y, L, D,
U, ar, az, al_profile, au_profile, ia_profile, di_profile;
	vector<int> ja, ia;
	vector<node> Arz;
	vector<vector<double>> A_loc, G_loc, M_loc, M_loc_g, x0, x0_3D;
	set <double> r_s, z_s;
	int N, Kel, time_number, istoc, num_r, num_z, n_layers, maxIter = 100000, iter =
0, Nmat, rasm; //rasm = 4 или 12 в зависимости от 2 и 3сти
	double eps = 1E-16, normr0, normR = 0, normB, hr, hz, rp, lambda, sigma, zs, t0,
t1, t2, nu0, nu1, nu2, istoc_r, istoc_z, h_r, h_z, d_r, d_z, r_left, r_right, z_top,
z_bottom, usel;

	///----------------общие функции и процедуры-----------------------------------
--------
	double gamma(double r); //гамма
	void Input(); //считывание всех необходимых данных
	void Clear_and_resize(); //инициализация векторов
	void multiply(vector<vector<double>>& A, vector<double>& x, vector<double>&
res);
```

```
	//------------------для двумерной задачи---------------------------------------
----------
	void Generate_Portrait(); // генерация портрета
	void Assemble_Locals(int el_id); // внесение локальных в глобальную СЛАУ
	void Generate_2D_Net(string net_file); //генерация сетки и конечных элементов
для 2-мерной задачи
	void Read_Layers(string obj_file); //формируем слоистую среду
	void Get_G(); //получение локальной матрицы G
	void Get_M();//получение локальных матриц M
	void Locals(int el_id, double t); // получение локальной матрицы A
	void Find_field(); //цикл по времени
	void Get_KR1(double t); // учет первых краевых
	int  Get_Num_Layer(double x0, double y0, double x1, double y1); //определяем к
какому слою относится элемент

	//--------------------для трехмерной задачи---------------------------------
----------
	double Get_solution(int time_layer, double r, double z, double variable, bool
isB, bool two_layer);
	void Generate_3D_Net(string net_file, bool go_to_z_up); //генерация сетки и ко-
нечных элементов
	void Middles();
	void Find_in_object(); //цикл по времени
	void Locals_3D(int el_id, int time_layer, bool two_layers);//dsx
	void Get_G_3D(); //получение локальной матрицы G
	void Get_M_3D();//получение локальных матриц M
	void Get_b_3D(int time_layer, int el_id, bool two_layers); // получение локаль-
ного b
	void Assemble_Locals_3D(int el_id); // внесение локальных A, b  в глобальную
СЛАУ
	void Generate_Portrait_3D();
	int Get_Num_Layer_3D(double z0, double z1);
	double Get_solution_3D(double t, double x, double y, double z, int i, bool B);
	void Make_grid(bool go_to_z_up);
	void Make_edges();
	void Make_obj_elems();
	void Get_KR1_3D();
	void Make_centers();
	void diag_preconditioning(vector<double>& x0);
	void Make_profile();
	void Without_selection();
	double h_x, h_y, h_obj_z, d_x, d_y, d_obj_z, x_left, x_right, y_left, y_right,
z_obj_top, z_obj_bottom, hx, hy,
		hz_o, x_obj_left, x_obj_right, h_obj_x, d_obj_x, y_obj_left, y_obj_right,
h_obj_y, d_obj_y, z_sr_left, z_sr_right,
		d_sr_z, reciever_x, reciever_y, reciever_z, time_h, time_beg, time_d;
	int num_x, num_y, num_obj_z, num_edges, num_elems, lay_obj;
	vector<double> ax, ay, a_obj_z, F;
	vector <node_3D> centers;
	set <double> x_s, y_s, z_obj_s; //границы КЭ

	//--------------------для решателя-----------------------------------------
----------
	void LOS_LU(int t);
	void LOS_LU_3D(int t);
	void FactLU(vector<double>& L, vector<double>& U, vector<double>& D);
	void Direct(vector<double>& L, vector<double>& D, vector<double>& y, vector<dou-
ble>& b);
	void Reverse(vector<double>& U, vector<double>& x, vector<double>& y);
	double Norm(vector<double>& x);
	double mult(const vector<double>& a, const vector<doubl
	void LUDec();
	void Direct_prof();
	void Reverse_prof();e>& b);
	void Ax(vector<double>& x, vector<double>& y);
```

```
//---------------------блок общих функций и процедур----------------------
double KURS::gamma(double r) // значение гамма всегда равно 1/r^2
{
	return 1.0 / (r * r);
}

void KURS::Input() // чтение данных
{
	ifstream in;

	in.open("istoc.txt");
	in >> istoc_r >> istoc_z;
	in.close();

	Read_Layers("layers");
	Generate_2D_Net("gen_net");

	for (int i = 0; i < N; i++)
		if (nodes[i].r == r_right || nodes[i].z == z_top || nodes[i].z == z_bot-
tom)
			KR1.push_back(i);

	in.open("material.txt");
	in >> Nmat;
	materials.resize(Nmat + 1);
	for (int i = 0; i < Nmat; i++)
	{
		in >> materials[i].mu;
		materials[i].lambda = 1. / materials[i].mu;
		in >> materials[i].sigma;
	}

	in.close();

	in.open("obj_material.txt"); //задаем параметры материалов для объекта
	in >> materials[Nmat].mu >> materials[Nmat].sigma;
	materials[Nmat].lambda = 1. / materials[Nmat].mu;
	in.close();

	in.open("time.txt");
	in >> time_number >> time_beg >> time_h >> time_d; //считываем количество вре-
менных слоев, начало по времени, шаг и коэф
	time.resize(time_number);
	time[0] = time_beg;
	for (int i = 1; i < time_number; i++) //считываем временные слои
		time[i] = time[i - 1] + time_h * pow(time_d, i - 1);
	in.close();

	Generate_3D_Net("gen_obj", false); //генерируем сетку для объекта
	Middles();

	in.open("reciever.txt");
	in >> reciever_x >> reciever_y >> reciever_z;
	in.close();
}

void KURS::Clear_and_resize()
{
	au.clear();
	au.resize(ia[N]);
	al.clear();
	al.resize(ia[N]);
	di.clear();
	di.resize(N);
	b.clear();
```

```
	b.resize(N);
	ja.resize(ia[N]);
	G_loc.resize(rasm);
	M_loc.resize(rasm);
	A_loc.resize(rasm);
	b_loc.clear();
	b_loc.resize(rasm);
	M_loc_g.resize(rasm);
	for (int i = 0; i < rasm; i++)
	{
		A_loc[i].resize(rasm);
		M_loc[i].resize(rasm);
		M_loc_g[i].resize(rasm);
		G_loc[i].resize(rasm);
	}
	M1.resize(N);
	M2.resize(N);
}

void KURS::multiply(vector<vector<double>>& A, vector<double>& x, vector<double>& res)
{
	for (int i = 0; i < rasm; i++)
	{
		res[i] = 0.;
		for (int j = 0; j < rasm; j++)
			res[i] += A[i][j] * x[j];
	}
}
//---------------------------------------------------------------------


//-------------блок функций и процедур для 2D задачи--------------------
void KURS::Generate_2D_Net(string net_file)
{
	ifstream fin(net_file + ".txt");
	fin >> r_left >> r_right >> z_top >> z_bottom >> h_r >> d_r >> h_z >> d_z;
//считываем границы области, шаги по r и z, коэффициент растяжения

   //строим сетку по r
	num_r = 0; //обнуляем число элементов по r
	double h = h_r / d_r;
	bool flag = true;
	ar.push_back(istoc_r);
	double usel = istoc_r;
	for (int i = 0; usel > r_left; i++) //идем влево от источника
	{
		flag = true;
		h = h * d_r;
		usel = ar[i] - h;
		if (usel < r_left)
		{
			flag = false;
			usel = r_left;
			if (abs(ar[i] - usel) < abs(ar[i] - ar[i - 1]))
			{
				ar[i] = usel;
				h = abs(ar[i] - ar[i - 1]) / d_r;
				i--;
			}
			else
				ar.push_back(usel);
		}
		if (flag == true)
			ar.push_back(usel);
	}
```

```
	reverse(ar.begin(), ar.end());

	h = h_r / d_r;
	for (int i = ar.size() - 1; usel < r_right; i++) //идем вправо от источника
	{
		flag = true;
		h = h * d_r;
		usel = ar[i] + h;
		if (usel > r_right)
		{
			usel = r_right;
			flag = false;
			if (abs(ar[i] - usel) < abs(ar[i] - ar[i - 1]))
			{
				ar[i] = usel;
				h = abs(ar[i] - ar[i - 1]) / d_r;
				i--;
			}
			else
				ar.push_back(usel);
		}
		if (flag == true)
			ar.push_back(usel);
	}

	num_r = ar.size();

	for (int j = 0; j < num_r; j++)
		r_s.insert(ar[j]);

	//строим сетку по z
	num_z = 0; //обнуляем число элементов по z

	az.push_back(istoc_z);
	usel = istoc_z;
	h = h_z / d_z;
	for (int i = 0; usel > z_bottom; i++) //идем вниз от источника
	{
		flag = true;
		h = h * d_z;
		usel = az[i] - h;
		if (i != 0)
		{
			for (int j = 0; j < n_layers; j++) //проеряем значения в окрестно-
стях границ слоев
				if (layers[j].y0 > usel && az[i] > layers[j].y0)
				{
					flag = false;
					usel = layers[j].y0;
					if (abs(az[i] - usel) < abs(az[i] - az[i - 1]))
					{
						az[i] = usel;
						h = abs(az[i] - az[i - 1]) / d_z;
						i--;
					}
					else
						az.push_back(usel);
				}
		}
		if (flag == true)
			az.push_back(usel);
	}

	reverse(az.begin(), az.end());
```

```
h = h_z / d_z;
for (int i = az.size() - 1; usel < z_top; i++) //идем вверх от источника
{
	flag = true;
	h = h * d_z;
	usel = az[i] + h;
	if (usel > z_top)
	{
		usel = z_top;
		flag = false;
		if (abs(az[i] - usel) < abs(az[i] - az[i - 1]))
		{
			az[i] = usel;
			h = abs(az[i] - az[i - 1]) / d_z;
			i--;
		}
		else
			az.push_back(usel);
	}
	if (flag == true)
		az.push_back(usel);
}

num_z = az.size();

//обработка по z
for (int j = 0; j < num_z; j++)
	z_s.insert(az[j]);

// Формирование узлов
Kel = (num_r - 1) * (num_z - 1); //количество конечных элементов
N = num_r * num_z; //число узлов
nodes.resize(N);
int k = 0;
for (set<double>::iterator j = z_s.begin(); j != z_s.end(); j++)
	for (set<double>::iterator i = r_s.begin(); i != r_s.end(); i++)
	{
		nodes[k].r = *i;
		nodes[k].z = *j;
		if (nodes[k].r == istoc_r && nodes[k].z == istoc_z)
			istoc = k;
		k++;
	}

// Получаем КЭ
int ku = 0;
elems.resize(Kel);
for (int i = 0; i < Kel; i++)
	elems[i].num.resize(4);
set<double>::iterator j_it = z_s.begin();
for (unsigned int j = 0; j < num_z - 1; j++)
{
	set<double>::iterator i_it = r_s.begin();
	for (unsigned int i = 0; i < num_r - 1; i++)
	{
		elems[ku].num[0] = num_r * j + i;
		elems[ku].num[1] = num_r * j + i + 1;
		elems[ku].num[2] = num_r * (j + 1) + i;
		elems[ku].num[3] = num_r * (j + 1) + i + 1;
		double x0 = *i_it, y0 = *j_it;
		set<double>::iterator k = j_it;
		k++;
		i_it++;
		double x1 = *i_it, y1 = *k;
		unsigned int num_area = Get_Num_Layer(x0, y0, x1, y1);
```

```
                elems[ku].mater = layers[num_area].n_mat;
                ku++;
            }
            j_it++;
        }
}
int KURS::Get_Num_Layer(double x0, double y0, double x1, double y1)
{
        for (int i = 0; i < n_layers; i++)
            if (layers[i].y0 <= y0 && layers[i].y1 >= y1)
                return i;
        return 0;
}

void KURS::Read_Layers(string obj_file)
{
        ifstream fin(obj_file + ".txt");
        layer tmp;
        fin >> n_layers;
        for (int i = 0; i < n_layers; i++)
        {
            fin >> tmp.y0 >> tmp.y1 >> tmp.n_mat;
            layers.push_back(tmp);
        }
}

void KURS::Get_G() // получение локальной G
{
        double a1 = (lambda * hz * rp) / (6 * hr),
            a2 = (lambda * hz) / (12),
            a3 = (lambda * hr * rp) / (6 * hz),
            a4 = (lambda * hr * hr) / (12 * hz);
        G_loc[0][0] = 2 * a1 + 2 * a2 + 2 * a3 + 1 * a4;
        G_loc[0][1] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
        G_loc[0][2] = 1 * a1 + 1 * a2 - 2 * a3 - 1 * a4;
        G_loc[0][3] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;

        G_loc[1][0] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
        G_loc[1][1] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
        G_loc[1][2] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
        G_loc[1][3] = 1 * a1 + 1 * a2 - 2 * a3 - 3 * a4;

        G_loc[2][0] = 1 * a1 + 1 * a2 - 2 * a3 - 1 * a4;
        G_loc[2][1] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
        G_loc[2][2] = 2 * a1 + 2 * a2 + 2 * a3 + 1 * a4;
        G_loc[2][3] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;

        G_loc[3][0] = -1 * a1 - 1 * a2 - 1 * a3 - 1 * a4;
        G_loc[3][1] = 1 * a1 + 1 * a2 - 2 * a3 - 3 * a4;
        G_loc[3][2] = -2 * a1 - 2 * a2 + 1 * a3 + 1 * a4;
        G_loc[3][3] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
}

void KURS::Get_M() // получение локальной M
{
        double g1 = gamma(rp), g2 = gamma(rp + hr);
        M_loc_g[0][0] = M_loc_g[2][2] = hr * hz / 4 * (
            g1 * (rp / 3 + hr / 15) +
            g2 * (rp / 9 + 2 * hr / 45));
        M_loc_g[0][2] = M_loc_g[2][0] = hr * hz / 12 * (
            g1 * (rp / 2 + hr / 10) +
            g2 * (rp / 6 + hr / 15));
        M_loc_g[0][3] = M_loc_g[3][0] = hr * hz / 12 * (
            g1 * (rp / 6 + hr / 15) +
            g2 * (rp / 6 + hr / 10));
```

```
	M_loc_g[1][0] = M_loc_g[3][2] = M_loc_g[2][3] = M_loc_g[0][1] =
		hr * hz / 4 * (
			g1 * (rp / 9 + 2 * hr / 45) +
			g2 * (rp / 9 + hr / 15));
	M_loc_g[1][1] = M_loc_g[3][3] = hr * hz / 4 * (
		g1 * (rp / 9 + hr / 15) +
		g2 * (rp / 3 + 4 * hr / 15));
	M_loc_g[1][2] = M_loc_g[2][1] = hr * hz / 12 * (
		g1 * (rp / 6 + hr / 15) +
		g2 * (rp / 6 + hr / 10));
	M_loc_g[1][3] = M_loc_g[3][1] = hr * hz / 12 * (
		g1 * (rp / 6 + hr / 10) +
		g2 * (rp / 2 + 2 * hr / 5));

	M_loc[0][0] = M_loc[2][2] = sigma * hr * hz / 4 * (4 * rp / 9 + hr / 9);
	M_loc[0][2] = M_loc[2][0] = sigma * hr * hz / 12 * (2 * rp / 3 + hr / 6);
	M_loc[0][3] = M_loc[3][0] = sigma * hr * hz / 12 * (rp / 3 + hr / 6);
	M_loc[1][0] = M_loc[3][2] = M_loc[2][3] = M_loc[0][1] = sigma * hr * hz / 4 * (2
* rp / 9 + hr / 9);
	M_loc[1][1] = M_loc[3][3] = sigma * hr * hz / 4 * (4 * rp / 9 + hr / 3);
	M_loc[1][2] = M_loc[2][1] = sigma * hr * hz / 12 * (rp / 3 + hr / 6);
	M_loc[1][3] = M_loc[3][1] = sigma * hr * hz / 12 * (2 * rp / 3 + hr / 2);
}

void KURS::Locals(int el_id, double t) // получение локальной матрицы A
{
	element el = elems[el_id];
	hr = nodes[el.num[1]].r - nodes[el.num[0]].r;
	hz = nodes[el.num[2]].z - nodes[el.num[0]].z;
	rp = nodes[el.num[0]].r;
	zs = nodes[el.num[0]].z;
	lambda = materials[el.mater].lambda;
	sigma = materials[el.mater].sigma;
	Get_G();
	Get_M();
}

void KURS::Generate_Portrait() // генерация портрета
{
	ia.resize(N + 1);
	ja.resize(rasm * rasm * Kel);
	vector<int> t1(rasm * rasm * Kel), t2(rasm * rasm * Kel), beg(N);
	int s = 0;
	for (int i = 0; i < N; i++)
		beg[i] = 0;
	for (int el = 0; el < Kel; el++)
	{
		for (int i = 0; i < rasm; i++)
		{
			int k = elems[el].num[i];
			for (int j = i + 1; j < rasm; j++)
			{
				int ind1 = k;
				int ind2 = elems[el].num[j];
				if (ind2 < ind1)
				{
					ind1 = ind2;
					ind2 = k;
				}
				int iaddr = beg[ind2];
				if (iaddr == 0)
				{
					s++;
					beg[ind2] = s;
					t1[s] = ind1;
```

```
                                t2[s] = 0;
                        }
                        else
                        {
                                while (t1[iaddr] < ind1 && t2[iaddr] > 0)
                                        iaddr = t2[iaddr];
                                if (t1[iaddr] > ind1)
                                {
                                        s++;
                                        t1[s] = t1[iaddr];
                                        t2[s] = t2[iaddr];
                                        t1[iaddr] = ind1;
                                        t2[iaddr] = s;
                                }
                                else if (t1[iaddr] < ind1)
                                {
                                        s++;
                                        t2[iaddr] = s;
                                        t1[s] = ind1;
                                        t2[s] = 0;
                                }
                        }
                }
        }
        }

        ia[0] = 0;
        for (int i = 0; i < N; i++)
        {
                ia[i + 1] = ia[i];
                int a = beg[i];
                while (a != 0)
                {
                        ja[ia[i + 1]] = t1[a];
                        ia[i + 1]++;
                        a = t2[a];
                }
        }
}

void KURS::Assemble_Locals(int el_id) // внесение локальных в глобальную СЛАУ
{
        vector<int> L = elems[el_id].num;
        int k = elems[el_id].num.size();
        for (int i = 0; i < k; i++)
                di[L[i]] += A_loc[i][i];

        for (int i = 0; i < rasm; i++)
                for (int j = 0; j < i; j++)
                        for (int k = ia[L[i]]; k < ia[L[i] + 1]; k++)
                                if (ja[k] == L[j])
                                {
                                        al[k] += A_loc[i][j];
                                        au[k] += A_loc[j][i];
                                        k++;
                                        break;
                                }
}

void KURS::Find_field()
{
        x0.resize(time_number);
        for (int i = 0; i < time_number; i++)
                x0[i].resize(N);
        temp.resize(N);
```

```
	//-------Инициализация решения на 0-м слое как решение стационарной задачи------
----------
	Clear_and_resize(); //очищаем вектора для каждого временного слоя

	for (int i = 0; i < Kel; i++)
	{
		Locals(i, time[0]);

		for (int i = 0; i < rasm; i++)
			for (int j = 0; j < rasm; j++)
				A_loc[i][j] = G_loc[i][j] + M_loc_g[i][j];

		Assemble_Locals(i); //сборка левой части

		for (int j = 0; j < rasm; j++)  //сборка вектора правой части
			b[elems[i].num[j]] += b_loc[j];
	}

	b[istoc] = b[istoc] + 1. / 2. / 3.14 / nodes[istoc].r; //задаем точечный источ-
ник в узле сетки

	Get_KR1(time[0]); //учет первых краевых

	LOS_LU(0);
	ofstream f("res.txt");
	f << "t = 0" << endl;
	for (int i = 0; i < N; i++)
		f << nodes[i].r << "\t" << nodes[i].z << "\t" << x0[0][i] << endl;
	f << endl;

	//-----------------------------------------------------------------------------
------------

	//----------------------Инициализация решения на 1-м слое---------------------
------------
	for (int t = 1; t < 3; t++)
	{
		Clear_and_resize(); //очищаем вектора для каждого временного слоя

		//считаем коэффициенты по времени для двухслойной неявной схемы
		t0 = time[t] - time[t - 1]; //t0
		nu0 = 1 / t0;

		for (int i = 0; i < Kel; i++)
		{
			Locals(i, time[t]);
			for (int i = 0; i < rasm; i++)
				for (int j = 0; j < rasm; j++)
					A_loc[i][j] = G_loc[i][j] + M_loc[i][j] * nu0 +
M_loc_g[i][j];

			Assemble_Locals(i); //сборка левой части

			for (int j = 0; j < 4; j++) //сборка вектора правой части
				temp[j] = x0[t - 1][elems[i].num[j]];
			multiply(M_loc, temp, M2);//M*x2
			for (int j = 0; j < 4; j++)
				M2[j] = M2[j] * nu0;
			for (int j = 0; j < 4; j++)
				b[elems[i].num[j]] += b_loc[j] + M2[j];
		}

		Get_KR1(time[t]);

		LOS_LU(t);
```

```
			f << "t = " << time[t] << endl;
			for (int i = 0; i < N; i++)
				f << nodes[i].r << "\t" << nodes[i].z << "\t" << x0[t][i] << endl;
			f << endl;

		}
		//-------------------------------------------------------------------------------
------------

		//-------------------Инициализация решения на 3-м и посл. слоях-----------------
------------
		for (int t = 3; t < time_number; t++)
		{
			Clear_and_resize(); //очищаем вектора для каждого временного слоя

			//считаем коэффициенты по времени
			t0 = time[t] - time[t - 1]; //t0
			t1 = time[t] - time[t - 2]; //t
			t2 = time[t - 1] - time[t - 2]; //t1
			nu0 = (t1 + t0) / t1 / t0;
			nu1 = t1 / t2 / t0;
			nu2 = t0 / t2 / t1;

			for (int i = 0; i < Kel; i++)
			{
				Locals(i, time[t]);
				for (int i = 0; i < rasm; i++)
					for (int j = 0; j < rasm; j++)
						A_loc[i][j] = G_loc[i][j] + M_loc[i][j] * nu0 +
M_loc_g[i][j];

				Assemble_Locals(i); //сборка левой части

				//сборка вектора правой части F = b_loc(j)-nu2*M*x2+nu1*M*x1
				for (int j = 0; j < rasm; j++)
					temp[j] = x0[t - 2][elems[i].num[j]];
				multiply(M_loc, temp, M2);//M*x2
				for (int j = 0; j < rasm; j++)
					temp[j] = x0[t - 1][elems[i].num[j]];
				multiply(M_loc, temp, M1);//M*x1
				for (int j = 0; j < rasm; j++)
					b[elems[i].num[j]] += b_loc[j] - M2[j] * nu2 + M1[j] * nu1;
			}

			Get_KR1(time[t]);

			LOS_LU(t);
			f << "t = " << time[t] << endl;
			for (int i = 0; i < N; i++)
				f << nodes[i].r << "\t" << nodes[i].z << "\t" << x0[t][i] << endl;
			f << endl;
		}
		f.close();
}

void KURS::Get_KR1(double t) // учет первых краевых
{
		for (int j = 0; j < KR1.size(); j++)
		{
			int node_id = KR1[j];
			di[node_id] = 1;
			b[node_id] = 0.;
			for (int k = ia[node_id]; k < ia[node_id + 1]; k++)
				al[k] = 0;
```

```
			for (int k = 0; k < ja.size(); k++)
				if (ja[k] == node_id)
					au[k] = 0;
		}
	}
	//---------------------------------------------------------------------

	////-------------блок функций и процедур для 3D задачи--------------------

	double KURS::Get_solution(int time_layer, double r, double z, double variable, bool
	isB, bool two_layer)
	{
		bool finded = false;
		int fe_sol = 0;
		for (int i = 0; i < num_r * num_z && !finded; i++) //сделать разные переменные
	для N!!!
		{
			if (r >= nodes[elems[i].num[0]].r && r <= nodes[elems[i].num[1]].r &&
				z >= nodes[elems[i].num[0]].z && z <= nodes[elems[i].num[2]].z)
			{
				finded = true;
				fe_sol = i;
			}
		}

		// Если нашли, то решение будет линейной комбинацией базисных функций на соот-
	ветствующие веса
		// Вычисление шага
		double hr = fabs(nodes[elems[fe_sol].num[1]].r - nodes[elems[fe_sol].num[0]].r);
		double hz = fabs(nodes[elems[fe_sol].num[2]].z - nodes[elems[fe_sol].num[0]].z);
		double A0 = 0.0, A1 = 0.0, result = 0.0, A2 = 0.0;

		if (isB == false)
		{
			// Находим линейные одномерные функции
			double X1 = (nodes[elems[fe_sol].num[1]].r - r) / hr;
			double X2 = (r - nodes[elems[fe_sol].num[0]].r) / hr;
			double Y1 = (nodes[elems[fe_sol].num[2]].z - z) / hz;
			double Y2 = (z - nodes[elems[fe_sol].num[0]].z) / hz;

			// Находим значение билинейных базисных функций
			double psi[4];
			psi[0] = X1 * Y1;
			psi[1] = X2 * Y1;
			psi[2] = X1 * Y2;
			psi[3] = X2 * Y2;

			if (two_layer == true)
			{
				// Линейная комбинация базисных функций на веса
				for (int i = 0; i < 4; i++)
				{
					A0 += x0[time_layer - 1][elems[fe_sol].num[i]] * psi[i]; //на
	предыдущем слое
					A1 += x0[time_layer][elems[fe_sol].num[i]] * psi[i]; //на те-
	кущем слое
				}
				A0 = variable / r * A0;
				A1 = variable / r * A1;
				result = (A0 - A1) / t0; //t0 = t[i]-t[i-1], то есть шаг по времени
			}
			else
			{
				// Линейная комбинация базисных функций на веса
				for (int i = 0; i < 4; i++)
```

```
                    {
                          A0 += x0[time_layer - 1][elems[fe_sol].num[i]] * psi[i]; //на
предыдущем слое
                          A1 += x0[time_layer][elems[fe_sol].num[i]] * psi[i]; //на те-
кущем слое
                          A2 += x0[time_layer - 2][elems[fe_sol].num[i]] * psi[i];//на
2 слоя назад

                    }
                    A0 = variable / r * A0;
                    A1 = variable / r * A1;
                    A2 = variable / r * A2;
                    t0 = time[time_layer] - time[time_layer - 1]; //t0
                    t1 = time[time_layer] - time[time_layer - 2]; //t
                    t2 = time[time_layer - 1] - time[time_layer - 2]; //t1
                    nu0 = (t1 + t0) / t1 / t0;
                    nu1 = -t1 / t2 / t0;
                    nu2 = t0 / t2 / t1;
                    result = A2 * nu2 + A0 * nu1 + A1 * nu0; //t0 = t[i]-t[i-1], то
есть шаг по времени
              }
       }
       else
       {
              // Находим линейные одномерные функции
              double X1 = (nodes[elems[fe_sol].num[1]].r - r) / hr;
              double X2 = (r - nodes[elems[fe_sol].num[0]].r) / hr;
              double Y1 = (nodes[elems[fe_sol].num[2]].z - z) / hz;
              double Y2 = (z - nodes[elems[fe_sol].num[0]].z) / hz;

              // Находим значение билинейных базисных функций
              double psi_der[8];
              psi_der[0] = X1 * Y1;
              psi_der[1] = X2 * Y1;
              psi_der[2] = X1 * Y2;
              psi_der[3] = X2 * Y2;
              psi_der[4] = -Y1 / hr;
              psi_der[5] = Y1 / hr;
              psi_der[6] = -Y2 / hr;
              psi_der[7] = Y2 / hr;

              // Линейная комбинация базисных функций на веса
              for (int i = 0; i < 4; i++)
              {
                    A0 = A0 + x0[time_layer][elems[fe_sol].num[i]] * psi_der[i];
//считаем A
                    A1 += x0[time_layer][elems[fe_sol].num[i]] * psi_der[i + 4];
//Производная по A
              }
              result = A0 / r + A1;
       }
       return result;
}

void KURS::Middles()
{
       ofstream fout("xyz_coords.txt");
       double middle = 0.0, diff = 0.0;
       fout << edges.size() << endl;
       for (int i = 0; i < edges.size(); i++)
       {
              diff = edges[i].end.x - edges[i].beg.x;
              middle = edges[i].beg.x + diff / 2.;
              if (diff != 0)
              {
```

```
                        fout << middle << " " << edges[i].beg.y << " " << edges[i].beg.z <<
endl;
                        edges[i].middle.x = middle;
                        edges[i].middle.y = edges[i].beg.y;
                        edges[i].middle.z = edges[i].beg.z;
                }
                else
                {
                        diff = edges[i].end.y - edges[i].beg.y;
                        middle = edges[i].beg.y + diff / 2.;
                        if (diff != 0)
                        {
                                fout << edges[i].beg.x << " " << middle << " " <<
edges[i].beg.z << endl;
                                edges[i].middle.x = edges[i].beg.x;
                                edges[i].middle.y = middle;
                                edges[i].middle.z = edges[i].beg.z;
                        }
                        else
                        {
                                diff = edges[i].end.z - edges[i].beg.z;
                                middle = edges[i].beg.z + diff / 2.;
                                fout << edges[i].beg.x << " " << edges[i].beg.y << " " <<
middle << endl;
                                edges[i].middle.x = edges[i].beg.x;
                                edges[i].middle.y = edges[i].beg.y;
                                edges[i].middle.z = middle;
                        }
                }
        }
        fout.close();
}

void KURS::Generate_Portrait_3D() // генерация портрета
{
        N = num_edges;
        Kel = num_elems;
        ia.resize(N + 1);
        ja.resize(rasm * rasm * Kel);
        vector<int> t1(rasm * rasm * Kel);
        vector<int> t2(rasm * rasm * Kel);
        vector<int> beg(N);
        int s = 0;
        for (int i = 0; i < N; i++)
                beg[i] = 0;
        for (int el = 0; el < Kel; el++)
        {
                for (int i = 0; i < rasm; i++)
                {
                        int k = obj_elems[el].num[i];
                        for (int j = i + 1; j < rasm; j++)
                        {
                                int ind1 = k;
                                int ind2 = obj_elems[el].num[j];
                                if (ind2 < ind1)
                                {
                                        ind1 = ind2;
                                        ind2 = k;
                                }
                                int iaddr = beg[ind2];
                                if (iaddr == 0)
                                {
                                        s++;
                                        beg[ind2] = s;
                                        t1[s] = ind1;
```

```
                        t2[s] = 0;
                    }
                    else
                    {
                        while (t1[iaddr] < ind1 && t2[iaddr] > 0)
                            iaddr = t2[iaddr];
                        if (t1[iaddr] > ind1)
                        {
                            s++;
                            t1[s] = t1[iaddr];
                            t2[s] = t2[iaddr];
                            t1[iaddr] = ind1;
                            t2[iaddr] = s;
                        }
                        else if (t1[iaddr] < ind1)
                        {
                            s++;
                            t2[iaddr] = s;
                            t1[s] = ind1;
                            t2[s] = 0;
                        }
                    }
                }
            }
        }

        ia[0] = 0;
        for (int i = 0; i < N; i++)
        {
            ia[i + 1] = ia[i];
            int a = beg[i];
            while (a != 0)
            {
                ja[ia[i + 1]] = t1[a];
                ia[i + 1]++;
                a = t2[a];
            }
        }
}

void KURS::Make_grid(bool go_to_z_up)
{
        //строим сетку по x
        num_x = 0; //обнуляем число элементов по x
        double h = h_obj_x / d_x;
        bool flag = true;
        double usel = x_obj_left;

        ax.push_back(x_obj_left);
        for (int i = 0; usel > x_left; i++) //идем влево от источника
        {
            flag = true;
            h = h * d_x;
            usel = ax[i] - h;
            if (usel < x_left)
            {
                flag = false;
                usel = x_left;
                if (abs(ax[i] - usel) < abs(ax[i] - ax[i - 1]))
                {
                    ax[i] = usel;
                    h = abs(ax[i] - ax[i - 1]) / d_x;
                    i--;
                }
                else
```

```
                        ax.push_back(usel);
                }
                if (flag == true)
                        ax.push_back(usel);
        }

        reverse(ax.begin(), ax.end());

        usel = x_obj_left;
        h = h_obj_x / d_obj_x;
        for (int i = ax.size() - 1; usel < x_obj_right; i++) //идем вправо от начала ко-
ординат
        {
                flag = true;
                h = h * d_obj_x;
                usel = ax[i] + h;
                if (usel > x_obj_right)
                {
                        usel = x_obj_right;
                        flag = false;
                        if (abs(ax[i] - usel) < abs(ax[i] - ax[i - 1]))
                        {
                                ax[i] = usel;
                                h = abs(ax[i] - ax[i - 1]) / d_obj_x;
                                i--;
                        }
                        else
                                ax.push_back(usel);
                }
                if (flag == true)
                        ax.push_back(usel);
        }

        h = h / d_x;
        for (int i = ax.size() - 1; usel < x_right; i++) //идем вправо от объекта
        {
                flag = true;
                h = h * d_x;
                usel = ax[i] + h;
                if (usel > x_right)
                {
                        usel = x_right;
                        flag = false;
                        if (abs(ax[i] - usel) < abs(ax[i] - ax[i - 1]))
                        {
                                ax[i] = usel;
                                h = abs(ax[i] - ax[i - 1]) / d_x;
                                i--;
                        }
                        else
                                ax.push_back(usel);
                }
                if (flag == true)
                        ax.push_back(usel);
        }
        num_x = ax.size();

        for (int j = 0; j < num_x; j++)
                x_s.insert(ax[j]);

        //строим сетку по y
        num_y = 0; //обнуляем число элементов по y
        ay.push_back(y_obj_left);
        usel = y_obj_left;
        h = h_obj_y / d_y;
```

```
for (int i = 0; usel > y_left; i++) //идем влево от источника
{
	flag = true;
	h = h * d_y;
	usel = ay[i] - h;
	if (usel < y_left)
	{
		flag = false;
		usel = y_left;
		if (abs(ay[i] - usel) < abs(ay[i] - ay[i - 1]))
		{
			ay[i] = usel;
			h = abs(ay[i] - ay[i - 1]) / d_y;
			i--;
		}
		else
			ay.push_back(usel);
	}
	if (flag == true)
		ay.push_back(usel);
}

reverse(ay.begin(), ay.end());

h = h_obj_y / d_obj_y;
usel = y_obj_left;

for (int i = ay.size() - 1; usel < y_obj_right; i++) //идем от начала координат
{
	flag = true;
	h = h * d_obj_y;
	usel = ay[i] + h;
	if (usel > y_obj_right)
	{
		usel = y_obj_right;
		flag = false;
		if (abs(ay[i] - usel) < abs(ay[i] - ay[i - 1]))
		{
			ay[i] = usel;
			h = abs(ay[i] - ay[i - 1]) / d_obj_y;
			i--;
		}
		else
			ay.push_back(usel);
	}
	if (flag == true)
		ay.push_back(usel);
}

h = h / d_y;
for (int i = ay.size() - 1; usel < y_right; i++) //идем от объекта
{
	flag = true;
	h = h * d_y;
	usel = ay[i] + h;
	if (usel > y_right)
	{
		usel = y_right;
		flag = false;
		if (abs(ay[i] - usel) < abs(ay[i] - ay[i - 1]))
		{
			ay[i] = usel;
			h = abs(ay[i] - ay[i - 1]) / d_y;
			i--;
```

```
                }
                else
                        ay.push_back(usel);
        }
        if (flag == true)
                ay.push_back(usel);
    }

    num_y = ay.size();

    //обработка по y
    for (int j = 0; j < num_y; j++)
        y_s.insert(ay[j]);

    //для уплотнения сетки по ближе к источнику
    num_obj_z = 0; //обнуляем число элементов по z
    a_obj_z.push_back(z_obj_top);
    usel = z_obj_top;
    h = h_obj_z / d_obj_z;
    for (int i = 0; usel > z_obj_bottom; i++) //идем вверх от начала координат
    {
        flag = true;
        h = h * d_obj_z;
        usel = a_obj_z[i] - h;
        if (usel < z_obj_bottom)
        {
                usel = z_obj_bottom;
                flag = false;
                if (abs(a_obj_z[i] - usel) < abs(a_obj_z[i] - a_obj_z[i - 1]))
                {
                        a_obj_z[i] = usel;
                        h = abs(a_obj_z[i] - a_obj_z[i - 1]) / d_obj_z;
                        i--;
                }
                else
                        a_obj_z.push_back(usel);
        }
        if (flag == true)
                a_obj_z.push_back(usel);
    }

    h = h / d_sr_z;
    for (int i = a_obj_z.size() - 1; usel > z_sr_left; i++) //идем вверх от начала
координат
    {
        flag = true;
        h = h * d_sr_z;
        usel = a_obj_z[i] - h;
        for (int j = 0; j < n_layers; j++) //проеряем значения в окрестностях
границ слоев
                if (layers[j].y0 > usel && a_obj_z[i] > layers[j].y0)
                {
                        usel = layers[j].y0;
                        flag = false;
                        if (abs(a_obj_z[i] - usel) < abs(a_obj_z[i] - a_obj_z[i -
1]))
                        {
                                a_obj_z[i] = usel;
                                h = abs(a_obj_z[i] - a_obj_z[i - 1]) / d_sr_z;
                                i--;
                        }
                        else
                                a_obj_z.push_back(usel);
                }
        if (flag == true)
```

```
                a_obj_z.push_back(usel);
        }
        reverse(a_obj_z.begin(), a_obj_z.end());

        h = h_obj_z / d_sr_z;
        for (int i = a_obj_z.size() - 1; usel < z_sr_right; i++) //идем от начала коор-
динат
        {
                flag = true;
                h = h * d_sr_z;
                usel = a_obj_z[i] + h;
                for (int j = 0; j < n_layers; j++)
                        if (layers[j].y1 < usel && a_obj_z[i] < layers[j].y1)
                        {
                                usel = layers[j].y1;
                                flag = false;
                                if (abs(a_obj_z[i] - usel) < abs(a_obj_z[i] - a_obj_z[i -
1]))
                                {
                                        a_obj_z[i] = usel;
                                        h = abs(a_obj_z[i] - a_obj_z[i - 1]) / d_sr_z;
                                        i--;
                                }
                                else
                                        a_obj_z.push_back(usel);
                        }
                if (flag == true)
                        a_obj_z.push_back(usel);
        }

        num_obj_z = a_obj_z.size();

        //обработка по z
        for (int j = 0; j < num_obj_z; j++)
                z_obj_s.insert(a_obj_z[j]);
}
void KURS::Make_edges() //Формирование ребер
{
        num_edges = num_obj_z * (num_x - 1) * num_y + num_obj_z * (num_y - 1) * num_x +
(num_obj_z - 1) * num_y * num_x;
        edges.resize(num_edges);
        int l = 0;
        //x
        for (int i = 0; i < num_obj_z; i++)
                for (int j = 0; j < num_x - 1; j++)
                        for (int k = 0; k < num_y; k++)
                        {
                                edges[l].beg.x = ax[j];
                                edges[l].beg.y = ay[k];
                                edges[l].beg.z = a_obj_z[i];
                                edges[l].end.x = ax[j + 1];
                                edges[l].end.y = ay[k];
                                edges[l].end.z = a_obj_z[i];
                                l++;
                        }
        //y
        for (int i = 0; i < num_obj_z; i++)
                for (int j = 0; j < num_y - 1; j++)
                        for (int k = 0; k < num_x; k++)
                        {
                                edges[l].beg.x = ax[k];
                                edges[l].beg.y = ay[j];
                                edges[l].beg.z = a_obj_z[i];
                                edges[l].end.x = ax[k];
                                edges[l].end.y = ay[j + 1];
```

```
                        edges[l].end.z = a_obj_z[i];
                        l++;
                  }
      //z
      for (int i = 0; i < num_obj_z - 1; i++)
            for (int k = 0; k < num_y; k++)
                  for (int j = 0; j < num_x; j++)
                  {
                        edges[l].beg.x = ax[j];
                        edges[l].beg.y = ay[k];
                        edges[l].beg.z = a_obj_z[i];
                        edges[l].end.x = ax[j];
                        edges[l].end.y = ay[k];
                        edges[l].end.z = a_obj_z[i + 1];
                        l++;
                  }
}

int KURS::Get_Num_Layer_3D(double z0, double z1)
{
      for (int i = 0; i < n_layers; i++)
            if (layers[i].y0 <= z0 && layers[i].y1 >= z1)
                  return i;
      return 0;
}

void KURS::Make_obj_elems()
{
      int shift_y = num_obj_z * num_y * (num_x - 1),
            shift_z = shift_y + num_obj_z * num_x * (num_y - 1), l = 0;

      num_elems = (num_obj_z - 1) * (num_x - 1) * (num_y - 1);
      obj_elems.resize(num_elems);

      for (int i = 0; i < num_obj_z - 1; i++)
            for (int j = 0; j < num_x - 1; j++)
                  for (int k = 0; k < num_y - 1; k++)
                  {
                        obj_elems[l].num.resize(12);
                        obj_elems[l].num[0] = i * num_y * (num_x - 1) + j * num_y +
k;
                        obj_elems[l].num[1] = i * num_y * (num_x - 1) + j * num_y + k
+ 1;
                        obj_elems[l].num[2] = (i + 1) * num_y * (num_x - 1) + j *
num_y + k;
                        obj_elems[l].num[3] = (i + 1) * num_y * (num_x - 1) + j *
num_y + k + 1;
                        obj_elems[l].num[4] = shift_y + i * num_x * (num_y - 1) + j +
k * num_x;
                        obj_elems[l].num[5] = shift_y + i * num_x * (num_y - 1) + j +
1 + k * num_x;
                        obj_elems[l].num[6] = shift_y + (i + 1) * num_x * (num_y - 1)
+ j + k * num_x;
                        obj_elems[l].num[7] = shift_y + (i + 1) * num_x * (num_y - 1)
+ j + 1 + k * num_x;
                        obj_elems[l].num[8] = shift_z + i * num_y * num_x + k * num_x
+ j;
                        obj_elems[l].num[9] = shift_z + i * num_y * num_x + k * num_x
+ 1 + j;
                        obj_elems[l].num[10] = shift_z + i * num_y * num_x + (k + 1)
* num_x + j;
                        obj_elems[l].num[11] = shift_z + i * num_y * num_x + (k + 1)
* num_x + 1 + j;

                        if (edges[obj_elems[l].num[0]].end.x <= x_obj_right &&
```

```
                                edges[obj_elems[l].num[0]].beg.x >= x_obj_left &&
                                edges[obj_elems[l].num[4]].end.y <= y_obj_right &&
                                edges[obj_elems[l].num[4]].beg.y >= y_obj_left &&
                                edges[obj_elems[l].num[8]].end.z <= z_obj_top &&
                                edges[obj_elems[l].num[8]].beg.z >= z_obj_bottom)
                                obj_elems[l].mater = Nmat; //следующий за слоями 2d
сетки
                        else
                        {
                                unsigned int num_area = Get_Num_Layer_3D(edges[obj_el-
ems[l].num[8]].beg.z, edges[obj_elems[l].num[8]].end.z);
                                obj_elems[l].mater = layers[num_area].n_mat;
                        }
                        l++;
                }
}

void KURS::Make_centers()
{
        //формируем точки – середины элементов, в которых будем искать значения
        centers.resize(num_elems);
        for (int i = 0; i < num_elems; i++)
        {
                element el = obj_elems[i];
                hx = edges[el.num[0]].end.x - edges[el.num[0]].beg.x;
                hy = edges[el.num[4]].end.y - edges[el.num[4]].beg.y;
                hz_o = edges[el.num[8]].end.z - edges[el.num[8]].beg.z;
                centers[i].x = edges[el.num[0]].beg.x + hx / 2.;
                centers[i].y = edges[el.num[4]].beg.y + hy / 2.;
                centers[i].z = edges[el.num[8]].beg.z + hz_o / 2.;
        }
}

void KURS::Generate_3D_Net(string net_file, bool go_to_z_up)
{
        ifstream fin(net_file + ".txt");
        fin >> x_obj_left >> x_obj_right >> h_obj_x >> d_obj_x >>
                y_obj_left >> y_obj_right >> h_obj_y >> d_obj_y >>
                z_obj_bottom >> z_obj_top >> h_obj_z >> d_obj_z >>
                lay_obj >>
                x_left >> x_right >> d_x >>
                y_left >> y_right >> d_y >>
                z_sr_left >> z_sr_right >> d_sr_z; //считываем границы области, шаги по
x,y,z, коэффициент растяжения, слой снизу, в котором лежит объект

        Make_grid(go_to_z_up);
        Make_edges();
        Make_obj_elems();
        Make_centers();

        // параллельно x
        for (int k = 0; k < num_y; k++)
                for (int j = 0; j < num_x - 1; j++)
                {
                        KR1_3D.push_back((num_obj_z - 1) * num_y * (num_x - 1) + j * num_y
+ k);
                        KR1_3D.push_back(j * num_y + k);
                }
        for (int i = 1; i < num_obj_z - 1; i++)
                for (int j = 0; j < num_x - 1; j++)
                {
                        KR1_3D.push_back(i * num_y * (num_x - 1) + j * num_y);
                        KR1_3D.push_back(i * num_y * (num_x - 1) + (j + 1) * num_y - 1);
                }
```

```
	// параллельно y
	int shift_y = num_obj_z * num_y * (num_x - 1),
		shift_z = shift_y + num_obj_z * num_x * (num_y - 1), l = 0;

	for (int k = 0; k < num_y - 1; k++)
		for (int j = 0; j < num_x; j++)
		{
			KR1_3D.push_back(shift_y + (num_obj_z - 1) * num_x * (num_y - 1) +
k * num_x + j);
			KR1_3D.push_back(shift_y + k * num_x + j);
		}
	for (int i = 1; i < num_obj_z - 1; i++)
		for (int k = 0; k < num_y - 1; k++)
		{
			KR1_3D.push_back(shift_y + i * num_x * (num_y - 1) + k * num_x);
			KR1_3D.push_back(shift_y + i * num_x * (num_y - 1) + (k + 1) *
num_x - 1);
		}

	// параллельно z
	for (int i = 0; i < num_obj_z - 1; i++)
		for (int j = 0; j < num_x; j++)
		{
			KR1_3D.push_back(shift_z + i * num_y * num_x + j);
			KR1_3D.push_back(shift_z + i * num_y * num_x + num_x * (num_y - 1)
+ j);
		}
	for (int i = 0; i < num_obj_z - 1; i++)
		for (int k = 1; k < num_y - 1; k++)
		{
			KR1_3D.push_back(shift_z + i * num_y * num_x + k * num_x);
			KR1_3D.push_back(shift_z + i * num_y * num_x + (k + 1) * num_x -
1);
		}
}

double KURS::Get_solution_3D(double t, double x, double y, double z, int i, bool B)
{

	element el = obj_elems[i];
	hx = edges[el.num[0]].end.x - edges[el.num[0]].beg.x;
	hy = edges[el.num[4]].end.y - edges[el.num[4]].beg.y;
	hz_o = edges[el.num[8]].end.z - edges[el.num[8]].beg.z;

	double result_x = 0.0, result_y = 0.0, result_z = 0.0, modul;

	//Линейные одномерные функции
	double X1 = (edges[el.num[0]].end.x - x) / hx;
	double X2 = (x - edges[el.num[0]].beg.x) / hx;
	double Y1 = (edges[el.num[1]].beg.y - y) / hy;
	double Y2 = (y - edges[el.num[0]].beg.y) / hy;
	double Z1 = (edges[el.num[3]].beg.z - z) / hz_o;
	double Z2 = (z - edges[el.num[0]].beg.z) / hz_o;

	if (B == false)
	{
		//Трилинейные базисные функции
		double psi[12];
		psi[0] = Y1 * Z1;
		psi[1] = Y2 * Z1;
		psi[2] = Y1 * Z2;
		psi[3] = Y2 * Z2;

		psi[4] = X1 * Z1;
		psi[5] = X2 * Z1;
```

```
		psi[6] = X1 * Z2;
		psi[7] = X2 * Z2;

		psi[8] = X1 * Y1;
		psi[9] = X2 * Y1;
		psi[10] = X1 * Y2;
		psi[11] = X2 * Y2;

		// Линейная комбинация базисных функций на веса
		for (int i = 0; i < 4; i++)
			result_x += x0_3D[t][el.num[i]] * psi[i];

		for (int i = 4; i < 8; i++)
			result_y += x0_3D[t][el.num[i]] * psi[i];

		for (int i = 8; i < 12; i++)
			result_z += x0_3D[t][el.num[i]] * psi[i];

		modul = sqrt(result_x * result_x + result_y * result_y + result_z * re-
sult_z);
	}
	else
	{
		double psi[8];
		psi[0] = -Z1 / hy; //dx/dy
		psi[1] = Z1 / hy;
		psi[2] = -Z2 / hy;
		psi[3] = Z2 / hy;

		psi[4] = -Z1 / hx; //dy/dx
		psi[5] = Z1 / hx;
		psi[6] = -Z2 / hx;
		psi[7] = Z2 / hx;

		for (int i = 0; i < 4; i++)
			result_x += x0_3D[t][el.num[i]] * psi[i];

		for (int i = 4; i < 8; i++)
			result_y += x0_3D[t][el.num[i]] * psi[i];

		modul = result_y - result_x;
	}
	return modul;
}

void KURS::Locals_3D(int el_id, int time_layer, bool two_layers) // получение локаль-
ной матрицы A
{
	element el = obj_elems[el_id];
	hx = edges[el.num[0]].end.x - edges[el.num[0]].beg.x;
	hy = edges[el.num[4]].end.y - edges[el.num[4]].beg.y;
	hz_o = edges[el.num[8]].end.z - edges[el.num[8]].beg.z;
	lambda = materials[el.mater].lambda;
	sigma = materials[el.mater].sigma;
	Get_G_3D();
	Get_M_3D();
	//Get_b_3D(time_layer, el_id, two_layers);
}

void KURS::Get_G_3D() // получение локальной G
{
	double a1 = (lambda * hx * hy) / (6 * hz_o),
		a2 = (lambda * hx * hz_o) / (6 * hy),
		a3 = (lambda * hz_o * hy) / (6 * hx),
		a4 = -(lambda * hz_o) / 6.,
```

```
		a5 = (lambda * hy) / 6.,
		a6 = -(lambda * hx) / 6.;

	for (int i = 0; i < 4; i++)
		for (int j = 0; j < 4; j++)
			G_loc[i][j] = a1 * G1[i][j] + a2 * G2[i][j];

	for (int i = 4; i < 8; i++)
		for (int j = 0; j < 4; j++)
			G_loc[j][i] = G_loc[i][j] = a4 * G2[i - 4][j];

	for (int i = 8; i < 12; i++)
		for (int j = 0; j < 4; j++)
			G_loc[i][j] = a5 * G3T[i - 8][j];

	for (int i = 4; i < 8; i++)
		for (int j = 4; j < 8; j++)
			G_loc[i][j] = a1 * G1[i - 4][j - 4] + a3 * G2[i - 4][j - 4];

	for (int i = 8; i < 12; i++)
		for (int j = 4; j < 8; j++)
			G_loc[j][i] = G_loc[i][j] = a6 * G1[i - 8][j - 4];

	for (int i = 0; i < 4; i++)
		for (int j = 8; j < 12; j++)
			G_loc[i][j] = a5 * G3[i][j - 8];

	for (int i = 8; i < 12; i++)
		for (int j = 8; j < 12; j++)
			G_loc[i][j] = a2 * G1[i - 8][j - 8] + a3 * G2[i - 8][j - 8];
}

void KURS::Get_M_3D() // получение локальной M
{
	double koef = hx * hy * hz_o / 36.;
	for (int i = 0; i < 4; i++)
		for (int j = 0; j < 4; j++)
			M_loc[i][j] = M_loc[j][i] = koef * M[i][j];

	for (int i = 4; i < 8; i++)
		for (int j = 4; j < 8; j++)
			M_loc[i][j] = M_loc[j][i] = koef * M[i - 4][j - 4];

	for (int i = 8; i < 12; i++)
		for (int j = 8; j < 12; j++)
			M_loc[i][j] = M_loc[j][i] = koef * M[i - 8][j - 8];
}
void KURS::Get_b_3D(int time_layer, int el_id, bool two_layers) // получение локаль-
ного b
{
	F.clear();
	F.resize(12);
	element el = obj_elems[el_id];
	if (el.mater == Nmat && time_layer != 0) //для области объекта
	{
		double x, y, r, z;
		double koef_sigma = sigma - materials[layers[lay_obj].n_mat].sigma;
		for (int i = 0; i < 8; i++)
		{
			x = edges[el.num[i]].middle.x;
			y = edges[el.num[i]].middle.y;
			r = sqrt(x * x + y * y);
			z = edges[el.num[i]].middle.z;
			if (two_layers == true)
			{
```

```
                        if (i < 4)
                              F[i] = koef_sigma * Get_solution(time_layer, r, z, -y,
false, true);
                        else
                              F[i] = koef_sigma * Get_solution(time_layer, r, z, x,
false, true);
                  }
                  else
                  {
                        if (i < 4)
                              F[i] = koef_sigma * Get_solution(time_layer, r, z, -y,
false, false);
                        else
                              F[i] = koef_sigma * Get_solution(time_layer, r, z, x,
false, false);
                  }
            }
            for (int i = 8; i < 11; i++)
                  F[i] = 0.;
      }
      multiply(M_loc, F, b_loc);
}

void KURS::Assemble_Locals_3D(int el_id) // внесение локальных в глобальную СЛАУ
{
      vector<int> L = obj_elems[el_id].num;
      int k = obj_elems[el_id].num.size();
      for (int i = 0; i < k; i++)
            di[L[i]] += A_loc[i][i];

      for (int i = 0; i < rasm; i++)
            for (int j = 0; j < i; j++)
                  for (int k = ia[L[i]]; k < ia[L[i] + 1]; k++)
                        if (ja[k] == L[j])
                        {
                              al[k] += A_loc[i][j];
                              au[k] += A_loc[j][i];
                              k++;
                              break;
                        }
}

void KURS::Make_profile() // Изменение строчно-столбцового формата в профильный
{
      double dif, sum = 0, ias = 0, k, j;
      ia_profile.resize(ia.size());
      ia_profile[0] = ia_profile[1] = 0;
      for (int i = 2; i < ia.size(); i++)
      {
            dif = ia[i] - ia[i - 1];
            for (j = sum; j < sum + dif; j++)
                  if (dif > 1)
                  {
                        if ((ja[j + 1] - ja[j]) == 1)
                              al_profile.push_back(al[j]);
                        else
                        {
                              al_profile.push_back(al[j]);
                              for (k = 0; k < ja[j + 1] - ja[j] - 1; k++)
                              {
                                    al_profile.push_back(0);
                                    ias++;
                              }
                        }
                  }
```

```
                else
                        al_profile.push_back(al[j]);
                ia_profile[i] = ia[i] + ias;
                sum += dif;
        }
        au_profile = al_profile;
}

void KURS::Without_selection()
{
        double storona = sqrt(5 * 5 * 3.141592653589793);
        double len = storona / 25;
        kvadro.resize(100);
        for (int i = 0; i < 25; i++)
        {
                kvadro[i].middle.x = -storona / 2 + len * (i + 0.5);
                kvadro[i].middle.y = storona / 2;
                kvadro[i].middle.z = 0;
                kvadro[i].l = -len;
        }
        for (int i = 0; i < 25; i++)
        {
                kvadro[i + 25].middle.x = -storona / 2 + len * (i + 0.5);
                kvadro[i + 25].middle.y = -storona / 2;
                kvadro[i + 25].middle.z = 0;
                kvadro[i + 25].l = len;
        }
        for (int i = 0; i < 25; i++)
        {
                kvadro[i + 50].middle.x = storona / 2;
                kvadro[i + 50].middle.y = -storona / 2 + len * (i + 0.5);
                kvadro[i + 50].middle.z = 0;
                kvadro[i + 50].l = len;
        }
        for (int i = 0; i < 25; i++)
        {
                kvadro[i + 75].middle.x = -storona / 2;
                kvadro[i + 75].middle.y = -storona / 2 + len * (i + 0.5);
                kvadro[i + 75].middle.z = 0;
                kvadro[i + 75].l = -len;
        }
        bool finded;
        int eletm;
        for (int j = 0; j < kvadro.size(); j++)
        {
                finded = false;
                eletm = 0;
                for (int i = 0; i < num_elems && !finded; i++)
                {
                        if (kvadro[j].middle.x >= edges[obj_elems[i].num[0]].beg.x &&
kvadro[j].middle.x <= edges[obj_elems[i].num[0]].end.x &&
                                kvadro[j].middle.y >= edges[obj_elems[i].num[4]].beg.y &&
kvadro[j].middle.y <= edges[obj_elems[i].num[4]].end.y &&
                                kvadro[j].middle.z >= edges[obj_elems[i].num[8]].beg.z &&
kvadro[j].middle.z <= edges[obj_elems[i].num[8]].end.z)
                        {
                                finded = true;
                                eletm = i;
                        }
                }

                hx = edges[obj_elems[eletm].num[0]].end.x - edges[obj_el-
ems[eletm].num[0]].beg.x;
                hy = edges[obj_elems[eletm].num[4]].end.y - edges[obj_el-
ems[eletm].num[4]].beg.y;
```

```
			hz_o = edges[obj_elems[eletm].num[8]].end.z - edges[obj_el-
ems[eletm].num[8]].beg.z;

			double X1 = (edges[obj_elems[eletm].num[0]].end.x - kvadro[j].middle.x) /
hx;
			double X2 = (kvadro[j].middle.x - edges[obj_elems[eletm].num[0]].beg.x) /
hx;
			double Y1 = (edges[obj_elems[eletm].num[1]].beg.y - kvadro[j].middle.y) /
hy;
			double Y2 = (kvadro[j].middle.y - edges[obj_elems[eletm].num[0]].beg.y) /
hy;
			double Z1 = (edges[obj_elems[eletm].num[3]].beg.z - kvadro[j].middle.z) /
hz_o;
			double Z2 = (kvadro[j].middle.z - edges[obj_elems[eletm].num[0]].beg.z) /
hz_o;

			// Находим значение триилинейных базисных функций
			double psi[12];
			psi[0] = Y1 * Z1;
			psi[1] = Y2 * Z1;
			psi[2] = Y1 * Z2;
			psi[3] = Y2 * Z2;

			psi[4] = X1 * Z1;
			psi[5] = X2 * Z1;
			psi[6] = X1 * Z2;
			psi[7] = X2 * Z2;

			if (j < kvadro.size() / 2)
				for (int k = 0; k < 4; k++)
					b[obj_elems[eletm].num[k]] += psi[k] * kvadro[j].l;
			else
				for (int k = 4; k < 8; k++)
					b[obj_elems[eletm].num[k]] += psi[k] * kvadro[j].l;
		}
}
void KURS::Find_in_object()
{
		N = num_edges;
		x0_3D.resize(time_number);
		for (int i = 0; i < time_number; i++)
			x0_3D[i].resize(N);
		temp.resize(N);
		int center_y = 41;
		/*int center_y = 82;*/

		//-------Инициализация решения на 0-м слое как решение стационарной задачи------
----------
		Clear_and_resize();//очищаем вектора для каждого временного слоя
		ofstream ff("chekelems.txt");

		for (int i = 0; i < num_elems; i++)
		{
			Locals_3D(i, 0, true);
			for (int i = 0; i < rasm; i++)
				for (int j = 0; j < rasm; j++)
					A_loc[i][j] = G_loc[i][j];

			Assemble_Locals_3D(i); //сборка левой части

			for (int j = 0; j < rasm; j++)		//сборка вектора правой части
				b[obj_elems[i].num[j]] += b_loc[j];
		}

		//Without_selection();
```

```
	Get_KR1_3D();
	/*di_profile.resize(N);
	di_profile = di;
	z.resize(N);
	z = b;
	Make_profile();
	LUDec();
	Direct_prof();
	Reverse_prof();
	x0_3D[0] = z;
	y.resize(N);
	Ax(x0_3D[0], y);
	for (int i = 0; i < N; i++)
		y[i] = y[i] - b[i];
	double nev = Norm(y) / Norm(b);
	cout << endl << nev;*/


	ofstream out("result_object.txt");
	out << "t = " << time[0] << endl;

	double otv;
	for (int i = 0; i < num_elems; i++)
		if (centers[i].y == centers[center_y].y) //срез по y
		{
			otv = Get_solution_3D(0, centers[i].x, centers[i].y, centers[i].z,
i, false);
			if (centers[i].x > -30 && centers[i].x < 60 && centers[i].z < 20 &&
centers[i].z > -60)
				out << centers[i].x << "\t" << centers[i].z << "\t" << otv <<
endl;
		}

	ofstream fout("in_reciever.txt");
	double r;
	bool finded = false;
	int reciever_elem = 0;
	//-------------------------------------------------------------------------
------------

	//----------------------Инициализация решения на 1-м слое-------------------
------------
	for (int t = 1; t < 2; t++)
	{
		Clear_and_resize();//очищаем вектора для каждого временного слоя

		//считаем коэффициенты по времени для двухслойной неявной схемы
		t0 = time[t] - time[t - 1]; //t0
		nu0 = 1 / t0;

		for (int i = 0; i < num_elems; i++)
		{
			Locals_3D(i, t, true);
			for (int i = 0; i < rasm; i++)
				for (int j = 0; j < rasm; j++)
					A_loc[i][j] = G_loc[i][j] + sigma * M_loc[i][j] * nu0;

			Assemble_Locals_3D(i); //сборка левой части

			//сборка вектора правой части
			for (int j = 0; j < rasm; j++)
				temp[j] = x0_3D[t - 1][obj_elems[i].num[j]];
			multiply(M_loc, temp, M2);//M*x2
			for (int j = 0; j < rasm; j++)
				b[obj_elems[i].num[j]] += b_loc[j] + M2[j] * nu0 * sigma;
```

```
		}

		Get_KR1_3D();
		LOS_LU_3D(t);

		out << "t = " << time[t] << endl;
		for (int i = 0; i < num_elems; i++)
			if (centers[i].y == centers[center_y].y) //срез по y
			{
				otv = Get_solution_3D(t, centers[i].x, centers[i].y, cen-
ters[i].z, i, false);
				if (centers[i].x > -30 && centers[i].x < 60 && centers[i].z <
20 && centers[i].z > -60)
					out << centers[i].x << "\t" << centers[i].z << "\t" <<
otv << endl;
			}

		//ищем значения в приемнике
		finded = false;
		reciever_elem = 0;

		for (int i = 0; i < num_elems && !finded; i++)
		{
			if (reciever_x >= edges[obj_elems[i].num[0]].beg.x && reciever_x <=
edges[obj_elems[i].num[0]].end.x &&
				reciever_y >= edges[obj_elems[i].num[4]].beg.y && reciever_y
<= edges[obj_elems[i].num[4]].end.y &&
				reciever_z >= edges[obj_elems[i].num[8]].beg.z && reciever_z
<= edges[obj_elems[i].num[8]].end.z)
			{
				finded = true;
				reciever_elem = i;
			}
		}

		r = sqrt(reciever_x * reciever_x + reciever_y * reciever_y);
		double B2D = Get_solution(t, r, reciever_z, -reciever_y, true, false);
		double B3D = Get_solution_3D(t, reciever_x, reciever_y, reciever_z,
reciever_elem, true);
		fout << setprecision(16) << time[t] << "\t" << B3D << endl;
	//ищем B общего поля в приемнике
	}
	//-------------------------------------------------------------------------
-----------

	//-------------------Инициализация решения на 3-м и посл. слоях-----------
-----------

	for (int t = 2; t < time_number; t++)
	{
		Clear_and_resize(); //очищаем вектора для каждого временного слоя

		//считаем коэффициенты по времени
		t0 = time[t] - time[t - 1]; //t0
		t1 = time[t] - time[t - 2]; //t
		t2 = time[t - 1] - time[t - 2]; //t1
		nu0 = (t1 + t0) / t1 / t0;
		nu1 = t1 / t2 / t0;
		nu2 = t0 / t2 / t1;

		for (int i = 0; i < num_elems; i++)
		{
			Locals_3D(i, t, false);
			for (int i = 0; i < rasm; i++)
				for (int j = 0; j < rasm; j++)
```

```
                                    A_loc[i][j] = G_loc[i][j] + sigma * M_loc[i][j] * nu0;

                        Assemble_Locals_3D(i); //сборка левой части

                        //сборка вектора правой части F = b_loc(j)-nu2*M*x2+nu1*M*x1
                        for (int j = 0; j < rasm; j++)
                              temp[j] = x0_3D[t - 2][obj_elems[i].num[j]];
                        multiply(M_loc, temp, M2);//M*x2
                        for (int j = 0; j < rasm; j++)
                              temp[j] = x0_3D[t - 1][obj_elems[i].num[j]];
                        multiply(M_loc, temp, M1);//M*x1
                        for (int j = 0; j < rasm; j++)
                              b[obj_elems[i].num[j]] += b_loc[j] - M2[j] * nu2 * sigma +
M1[j] * nu1 * sigma;
                  }

                  Get_KR1_3D();
                  LOS_LU_3D(t);

                  for (int i = 0; i < num_elems; i++)
                  {
                        if (centers[i].y == centers[center_y].y) //срез по y
                        {
                              otv = Get_solution_3D(t, centers[i].x, centers[i].y, cen-
ters[i].z, i, false);
                              if (centers[i].x > -30 && centers[i].x < 60 && centers[i].z <
20 && centers[i].z > -60)
                                    out << centers[i].x << "\t" << centers[i].z << "\t" <<
otv << endl;
                        }
                  }
                  double B2D = Get_solution(t, r, reciever_z, -reciever_y, true, false);
                  double B3D = Get_solution_3D(t, reciever_x, reciever_y, reciever_z,
reciever_elem, true);
                  fout << setprecision(16) << time[t] << "\t" << B3D << endl;
            //ищем В общего поля в приемнике
            }
            fout.close();
            out.close();
}

void KURS::Get_KR1_3D() // учет первых краевых
{
            for (int j = 0; j < KR1_3D.size(); j++)
            {
                  int node_id = KR1_3D[j];
                  di[node_id] = 1E+15;
                  b[node_id] = 0;
            }
}
//-----------------------------------------------------------------------

//--------------блок функций и процедур для решателя--------------------

void KURS::Ax(vector<double>& x, vector<double>& y)
{
            for (int i = 0; i < N; i++)
            {
                  y[i] = di[i] * x[i];
                  for (int j = ia[i]; j < ia[i + 1]; j++)
                  {
                        int k = ja[j];
                        y[i] += al[j] * x[k];
                        y[k] += au[j] * x[i];
                  }
```

```
	}
}

double KURS::Norm(vector<double>& x)
{
	double norm = 0;
	for (int i = 0; i < N; i++)
		norm += x[i] * x[i];
	return(sqrt(norm));
}

double KURS::mult(const vector<double>& a, const vector<double>& b)
{
	double res = 0;
	for (int i = 0; i < a.size(); i++)
		res += a[i] * b[i];
	return res;
}

void KURS::LOS_LU(int t)
{
	r.resize(N);
	z.resize(N);
	p.resize(N);
	Ar.resize(N);
	y.resize(N);
	L.resize(ia[N]);
	D.resize(N);
	U.resize(ia[N]);
	normB = Norm(b);
	cout.precision(15);
	FactLU(L, U, D);
	double p_p = 0, p_r = 0, r_r = 0, Ar_p = 0;
	double a = 0, B = 0, eps2 = 1e-10;
	Ax(x0[t], y);
	for (int i = 0; i < N; i++)
		y[i] = b[i] - y[i];
	Direct(L, D, r, y);
	Reverse(U, z, r);
	Ax(z, y);
	Direct(L, D, p, y);
	r_r = mult(r, r);
	normR = sqrt(r_r) / normB;
	for (iter = 1; iter < maxIter + 1 && normR >= eps; iter++)
	{
		p_p = mult(p, p);
		p_r = mult(p, r);
		a = p_r / p_p;
		for (int i = 0; i < N; i++)
		{
			x0[t][i] = x0[t][i] + z[i] * a;
			r[i] = r[i] - p[i] * a;
		}
		Reverse(U, y, r);
		Ax(y, Ar);
		Direct(L, D, Ar, Ar);
		Ar_p = mult(Ar, p);
		B = -(Ar_p / p_p);
		for (int i = 0; i < N; i++)
		{
			z[i] = y[i] + z[i] * B;
			p[i] = Ar[i] + p[i] * B;
		}
		if (r_r - (r_r - a * a * p_p) < eps2)
			r_r = mult(r, r);
```

```
			else
				r_r = r_r - a * a * p_p;
			normR = sqrt(r_r) / normB;
			cout << iter << ". " << normR << endl;
		}
}

void KURS::LOS_LU_3D(int t)
{
		cout << t << endl;
		r.resize(N);
		z.resize(N);
		p.resize(N);
		Ar.resize(N);
		y.resize(N);
		L.resize(ia[N]);
		D.resize(N);
		U.resize(ia[N]);
		normB = Norm(b);
		cout.precision(15);
		FactLU(L, U, D);
		double p_p = 0, p_r = 0, r_r = 0, Ar_p = 0;
		double a = 0, B = 0, eps2 = 1e-10;
		Ax(x0_3D[t], y);
		for (int i = 0; i < N; i++)
			y[i] = b[i] - y[i];
		Direct(L, D, r, y);
		Reverse(U, z, r);
		Ax(z, y);
		Direct(L, D, p, y);
		r_r = mult(r, r);
		normR = sqrt(r_r) / normB;
		normr0 = normR;
		for (iter = 1; iter < maxIter + 1 && normR / normr0 >= eps; iter++)
		{
			p_p = mult(p, p);
			p_r = mult(p, r);
			a = p_r / p_p;
			for (int i = 0; i < N; i++)
			{
				x0_3D[t][i] = x0_3D[t][i] + z[i] * a;
				r[i] = r[i] - p[i] * a;
			}
			Reverse(U, y, r);
			Ax(y, Ar);
			Direct(L, D, Ar, Ar);
			Ar_p = mult(Ar, p);
			B = -(Ar_p / p_p);
			for (int i = 0; i < N; i++)
			{
				z[i] = y[i] + z[i] * B;
				p[i] = Ar[i] + p[i] * B;
			}
			if (r_r - (r_r - a * a * p_p) < eps2)
				r_r = mult(r, r);
			else
				r_r = r_r - a * a * p_p;
			normR = sqrt(r_r) / normB;
			cout << iter << ". " << normR / normr0 << endl;
		}
}
void KURS::LUDec() //  LU разложение
{
		for (int i = 0; i < N; i++)
		{
```

```
			int i0 = ia_profile[i];
			int i1 = ia_profile[i + 1];
			int j = i - (ia_profile[i + 1] - ia_profile[i]);
			double bdi = 0;
			for (int k = ia_profile[i]; k < ia_profile[i + 1]; k++, j++)
			{
				int ki = ia_profile[i];
				int kj = ia_profile[j];
				int dif = k - ia_profile[i] - ia_profile[j + 1] + ia_profile[j];
				if (dif < 0)
					kj += abs(dif);
				else
					ki += dif;
				double bal = 0;
				double bau = 0;
				for (ki; ki < k; ki++, kj++)
				{
					bal += al_profile[ki] * au_profile[kj];
					bau += au_profile[ki] * al_profile[kj];
				}
				al_profile[k] = al_profile[k] - bal;
				au_profile[k] = au_profile[k] - bau;
				au_profile[k] = au_profile[k] / di_profile[j];
				bdi += al_profile[k] * au_profile[k];
			}
			di_profile[i] -= bdi;
		}
	}

	void KURS::Direct_prof() // прямой ход Ly=F
	{
		for (int i = 0; i < N; i++)
		{
			int j = i - (ia_profile[i + 1] - ia_profile[i]);
			double sum = 0;
			for (int k = ia_profile[i]; k < ia_profile[i + 1]; k++, j++)
			{
				sum += z[j] * al_profile[k];
			}
			z[i] = (z[i] - sum) / di_profile[i];
		}
	}

	void KURS::Reverse_prof() // обратный ход Ux=y
	{
		for (int i = N - 1; i >= 0; i--)
		{
			int j = i - (ia_profile[i + 1] - ia_profile[i]);
			for (int k = ia_profile[i]; k < ia_profile[i + 1]; k++, j++)
			{
				z[j] -= z[i] * au_profile[k];
			}
		}
	}


	void KURS::FactLU(vector<double>& L, vector<double>& U, vector<double>& D)
	{
		L = al;
		U = au;
		D = di;
		double l, u, d;
		for (int k = 0; k < N; k++)
		{
			d = 0;
```

```
			int i0 = ia[k], i1 = ia[k + 1];
			int i = i0;
			for (; i0 < i1; i0++)
			{
				l = 0;
				u = 0;
				int j0 = i, j1 = i0;
				for (; j0 < j1; j0++)
				{
					int t0 = ia[ja[i0]], t1 = ia[ja[i0] + 1];
					for (; t0 < t1; t0++)
					{
						if (ja[j0] == ja[t0])
						{
							l += L[j0] * U[t0];
							u += L[t0] * U[j0];
						}
					}
				}
				L[i0] -= l;
				U[i0] -= u;
				U[i0] /= D[ja[i0]];
				d += L[i0] * U[i0];
			}
			D[k] -= d;
		}
}

void KURS::Direct(vector<double>& L, vector<double>& D, vector<double>& y, vector<dou-
ble>& b)
{
		y = b;
		for (int i = 0; i < N; i++)
		{
			double sum = 0;
			int k0 = ia[i], k1 = ia[i + 1];
			int j;
			for (; k0 < k1; k0++)
			{
				j = ja[k0];
				sum += y[j] * L[k0];
			}
			double buf = y[i] - sum;
			y[i] = buf / D[i];
		}
}

void KURS::Reverse(vector<double>& U, vector<double>& x, vector<double>& y)
{
		x = y;
		for (int i = N - 1; i >= 0; i--)
		{
			int k0 = ia[i], k1 = ia[i + 1];
			int j;
			for (; k0 < k1; k0++)
			{
				j = ja[k0];
				x[j] -= x[i] * U[k0];
			}
		}
}

int main()
{
		KURS A;
		A.Input();
```

```
        A.rasm = 4; //для двумерной
        A.Generate_Portrait();
        A.Find_field();

        A.rasm = 12; //для трехмерной
        A.Generate_Portrait_3D();
        A.Find_in_object();
        return 0;
}
```