

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Вычислительных технологий
(полное название кафедры)

Петр Сергеевич Жигалов

(фамилия, имя, отчество студента – автора работы)

Анализ систем источник-приёмник в задачах морской геоэлектрики

(полное название темы магистерской диссертации)

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
по направлению высшего образования

01.04.02 – Прикладная математика и информатика

(код и наименование направления подготовки магистра)

факультет прикладной математики и информатики

(факультет)

Тема диссертации утверждена приказом по НГТУ № 4931/2 от «15» октября 2014 г.

Руководитель

Шурина Э.П.

(фамилия, имя, отчество)

д.т.н., профессор

(ученая степень, ученое звание)

Новосибирск 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Вычислительных технологий
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Шокин Ю.И.
(фамилия, имя, отчество)

.....
(подпись, дата)

ЗАДАНИЕ
на магистерскую диссертацию
студенту Жигалову Петру Сергеевичу
(фамилия, имя, отчество)
факультета прикладной математики и информатики
Направление подготовки 01.04.02 – Прикладная математика и информатика
(код и наименование направления подготовки магистра)
Магистерская программа Математическое моделирование
(наименование программы)
детерминированных и стохастических процессов
Тема Анализ систем источник-приёмник в задачах морской геоэлектрики
(полное название темы)

Цели работы решение трёхмерной прямой задачи морской
геоэлектрики векторным методом конечных элементов

Руководитель
Шурина Э.П.
(фамилия, имя, отчество)
д.т.н., профессор
(ученая степень, ученое звание)

.....
(подпись, дата)

Студент
Жигалов П.С.
(фамилия, имя, отчество)
ФПМИ, ПММ-42
(факультет, группа)

.....
(подпись, дата)

Аннотация

Отчёт 90 с., 33 рис., 9 табл., 49 источников, 2 прил.

**СИСТЕМА УРАВНЕНИЙ МАКСВЕЛА, УРАВНЕНИЕ ГЕЛЬМГОЛЬЦА,
ВЕКТОРНЫЙ МКЭ, РМЛ-СЛОЙ, МОРСКАЯ ГЕОЭЛЕКТРИКА, НЕФТЯНЫЕ
МЕСТОРОЖДЕНИЯ**

Объектом исследования является поведение электрического поля в недрах земли под слоем морской воды.

Цель работы – решение трёхмерной прямой задачи морской геоэлектрики векторным методом конечных элементов.

В процессе работы проводилось численное моделирование электрического поля векторным методом конечных элементов, проводились исследования влияния слоя воздуха, целесообразности применения РМЛ-слоя для ограничения области моделирования, сравнивалось поведение электромагнитного поля при различном расположении источника поля и объектов с различными электрофизическими свойствами друг относительно друга.

В результате исследования было получено представление электрического поля в различных средах, также были даны рекомендации по возможностям ограничения области моделирования.

Оглавление

Введение	6
1. Математическая модель	9
1.1. Уравнения Максвелла и Гельмгольца	9
1.2. Вариационная постановка	11
1.3. Вариационная постановка с учётом PML-слоя	14
1.4. Дискретная вариационная постановка	15
1.5. Тетраэдральные конечные элементы	17
1.6. Треугольные конечные элементы	19
1.7. Двухуровневый решатель	20
2. Построение и решение СЛАУ	23
2.1. Структура глобальной матрицы СЛАУ	23
2.2. Учёт краевых условий	24
2.3. Учёт токовой петли	25
3. Вычислительные эксперименты	26
3.1. Верификация программного комплекса	26
3.1.1 Расчётная область	26
3.1.2 Тестирование на гладких функциях	26
3.2. Исследование влияния слоя воздуха	27
3.2.1 Описание расчётной области	28
3.2.2 Конечноэлементная сетка	28
3.2.3 Результаты исследования	29
3.3. Исследование эффективности применения PML-слоя	31
3.3.1 Описание расчётной области	32
3.3.2 Варьирование коэффициентов растяжения	33
3.3.3 Варьирование толщины PML-слоя	33
3.3.4 Варьирование размера области, на границе которой вводится PML-слой	34

3.3.5	Проверка выполнения условий на контактных границах	35
3.3.6	Графическое представление результатов	36
3.3.7	Анализ целесообразности применения PML-слоя	37
3.4.	Задача, приближенная к реальной	38
3.4.1	Описание расчётной области	38
3.4.2	Конечноэлементная сетка	39
3.4.3	Результаты вычислительного эксперимента	40
4.	Описание программного комплекса	46
4.1.	Формат конфигурационных файлов	46
4.2.	Аргументы командной строки и переменные окружения	47
	Заключение	49
	Список литературы	50
	Приложение А. Примеры конфигурационных файлов	55
	Приложение Б. Интерфейсы основных классов программного комплекса	60

Введение

В современном мире сложилась ситуация, что экономика многих стран, в число которых входят Россия, Швеция, Канада, Объединённые Арабские Эмираты, зависит от цены на нефть. Цены на углеводороды могут расти или падать, но конкуренция за обладание ими всегда велика и доходит порой до вооружённых конфликтов. Особенно актуальными в последнее время становятся задачи геологоразведки в недрах земли, которые находятся под толщей морской воды, ведь, по оценкам специалистов, на территории только Северного Ледовитого океана может находиться до четверти мировых запасов нефти и газа [1].

Обычно задача морской геоэлектрики выглядит так, как показано на рисунке 1: источник перемещается кораблём со специальным оборудованием. Приёмники располагаются на морском дне, обычно вдоль линии или по равномерной сетке.

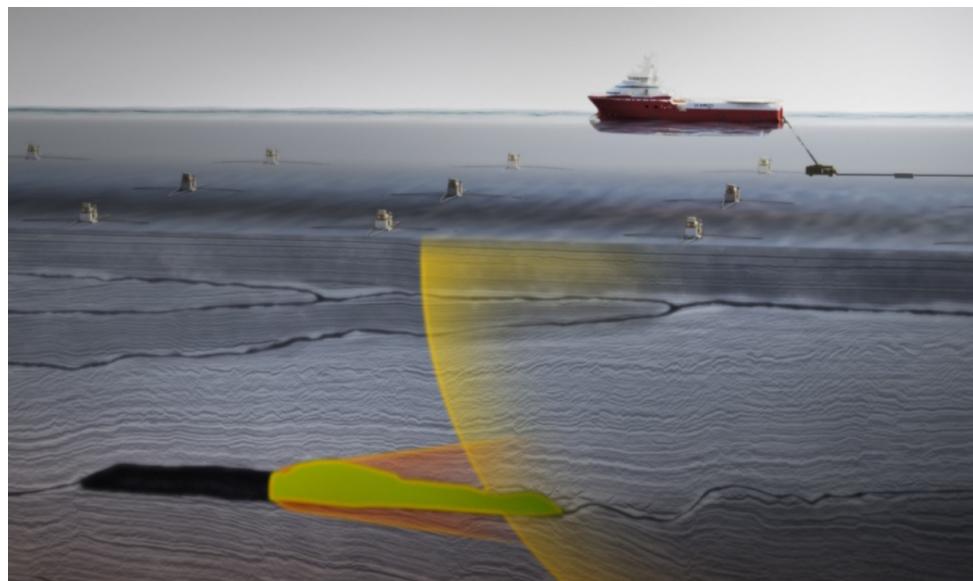


Рисунок 1 — задача морской геоэлектрики

К отличительным особенностям задач морской геоэлектрики относится низкая частота источника электромагнитного поля (0.25-100 Гц) [2] и, как следствие, большой размер области моделирования. Кроме того, морское дно имеет сложный рельеф, а электропроводность морской воды может изменяться в зависимости от глубины [1]. Это вызвано различной солёностью и температурой разных слоёв морской воды, эти свойства, кроме того, могут меняться от внеш-

них факторов, таких как сезон, погодные условия или интенсивность таяния льдов.

Геометрические размеры локального источника возбуждения электромагнитного поля составляют несколько сотен метров, тогда как размеры области моделирования составляют 6000 м и более. Это приводит к необходимости применения специальных методов для сокращения расчётной области. Для этого нередко в область моделирования не включается воздух, вместо этого на границе раздела сред воздух-вода задаются условия непротекания. Однако такой подход не позволяет правильно учесть физические процессы, протекающие в воздухе [3, 4]. Другим подходом является выделение из области некоторой подобласти меньшего размера и задание на её границах специальных поглощающих условий. К таким условиям относятся Absorbing Boundary Conditions (ABC) [5], предложенные G. Mur в 1981 году, а также Perfectly Matched Layer (PML) [6, 7], который предложил J.P. Berenger в 1994 году. PML-слой учитывается в вариационной формулировке как подобласть со специальными коэффициентами.

В настоящее время для решения задач морской геоэлектрики наиболее широко используется векторный метод конечных элементов (ВМКЭ). Этот метод подробно освещён в работах [8, 9].

Целью работы является решение трёхмерной прямой задачи морской геоэлектрики векторным методом конечных элементов. Для достижения поставленной цели были сформулированы следующие задачи:

1. Исследование влияния слоя воздуха при различной глубине источника электромагнитного возмущения.
2. Исследование целесообразности применения PML-слоя для ограничения области моделирования в задачах морской геоэлектрики на низких частотах.
3. Исследование поведения электромагнитного поля при различном расположении источника поля и искомого объекта друг относительно друга.

Основные промежуточные результаты работы были представлены на следующих конференциях:

- Городская научно-практическая конференция аспирантов и магистрантов «Progress Through Innovation» [10] (Новосибирск, 2015);
- 53 Международная научная студенческая конференция МНСК-2015 [11] (Новосибирск, 2015);
- X Международная научно-техническая конференция «Аналитические и численные методы моделирования естественно-научных и социальных проблем» [12] (Пенза, 2015);
- Всероссийская научно-техническая конференция «Наука. Технологии. Инновации» [4] (Новосибирск, 2015);
- Российская научно-техническая конференция «Обработка информации и математическое моделирование» [13] (Новосибирск, 2015).

1. Математическая модель

1.1. Уравнения Максвелла и Гельмгольца

Электромагнитное поле описывается системой уравнений Максвелла [14]:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \text{ — закон Фарадея,} \quad (1)$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \sigma \mathbf{E} + \mathbf{J} \text{ — закон Ампера,} \quad (2)$$

$$\nabla \cdot \mathbf{B} = 0 \text{ — закон Гаусса для магнитной индукции,}$$

$$\nabla \cdot \mathbf{D} = \rho \text{ — закон Гаусса для электрической индукции,}$$

где \mathbf{E} — напряжённость электрического поля (В/м), \mathbf{H} — напряжённость магнитного поля (А/м), $\mathbf{B} = \mu \mathbf{H}$ — магнитная индукция (Тл), $\mathbf{D} = \varepsilon \mathbf{E}$ — электрическая индукция (Кл/м²), ρ — плотность электрических зарядов (Кл/м³), σ — электрическая проводимость (См/м), $\varepsilon = \varepsilon_r \varepsilon_0$ — диэлектрическая проницаемость (Ф/м), ε_r — относительная диэлектрическая проницаемость, $\varepsilon_0 = 8.85 \cdot 10^{-12}$ Ф/м — диэлектрическая проницаемость вакуума, $\mu = \mu_r \mu_0$ — магнитная проницаемость (Гн/м), μ_r — относительная магнитная проницаемость, $\mu_0 = 4\pi \cdot 10^{-7}$ Гн/м — магнитная проницаемость вакуума, \mathbf{J} — плотность стороннего электрического тока (А/м²).

На границе $\Gamma = \Omega^j \cap \Omega^k$ между материалами j и k с различными электрофизическими свойствами должны быть выполнены следующие условия:

$$[\![\mathbf{E} \times \mathbf{n}]\!]_{\Gamma} = 0 \text{ — тангенциальная компонента } \mathbf{E} \text{ непрерывна,} \quad (3)$$

$$[\![\mathbf{B} \cdot \mathbf{n}]\!]_{\Gamma} = 0 \text{ — нормальная компонента } \mathbf{B} \text{ непрерывна,}$$

$$[\![\mathbf{H} \times \mathbf{n}]\!]_{\Gamma} = \mathbf{J}_{\Gamma} \text{ — тангенциальная компонента } \mathbf{H} \text{ разрывна,}$$

$$[\![\mathbf{D} \cdot \mathbf{n}]\!]_{\Gamma} = \rho_{\Gamma} \text{ — нормальная компонента } \mathbf{D} \text{ разрывна.} \quad (4)$$

При моделировании электрического поля в частотной области будем предполагаться, что \mathbf{E} и \mathbf{J} зависят от времени гармонически, то есть:

$$\mathbf{E}(t) = \mathbf{E}e^{i\omega t}, \quad \mathbf{J}(t) = \mathbf{J}e^{i\omega t}.$$

Используя такое представление, получим из (1) и (2):

$$\nabla \times \mathbf{E} = -i\omega \mathbf{B}, \quad (5)$$

$$\nabla \times \mathbf{H} = i\omega \mathbf{D} + \sigma \mathbf{E} + \mathbf{J}. \quad (6)$$

Выполним следующие преобразования над (5):

$$\nabla \times \mathbf{E} = -i\omega \mu \mathbf{H},$$

$$\mu^{-1} \nabla \times \mathbf{E} = -i\omega \mathbf{H},$$

$$\nabla \times (\mu^{-1} \nabla \times \mathbf{E}) = -i\omega \nabla \times \mathbf{H}. \quad (7)$$

Подставим в (7) (6):

$$\nabla \times (\mu^{-1} \nabla \times \mathbf{E}) = -i\omega(i\omega \varepsilon \mathbf{E} + \sigma \mathbf{E} + \mathbf{J}),$$

$$\nabla \times (\mu^{-1} \nabla \times \mathbf{E}) = \omega^2 \varepsilon \mathbf{E} - i\omega \sigma \mathbf{E} - i\omega \mathbf{J},$$

$$\nabla \times (\mu^{-1} \nabla \times \mathbf{E}) + k^2 \mathbf{E} = -i\omega \mathbf{J}, \quad (8)$$

где $k^2 = i\omega \sigma - \omega^2 \varepsilon$. Уравнение (8) называется уравнением Гельмгольца.

Краевые условия для уравнения (8) можно записать следующим образом:

$$\mathbf{E} \times \mathbf{n}|_{S_1} = \mathbf{E}^g, \quad (9)$$

$$\sigma \mathbf{E} \cdot \mathbf{n}|_{S_2} = 0. \quad (10)$$

В случае удалённых границ (9) принимает вид условия «большого бака»:

$$\mathbf{E} \times \mathbf{n}|_{S_1} = 0. \quad (11)$$

Источником электромагнитного возмущения будет выступать замкнутая токовая петля.

Подействуем оператором $\nabla \cdot$ на уравнение (2):

$$\nabla \cdot (\nabla \times \mathbf{H}) = \nabla \cdot \left(\frac{\partial \mathbf{D}}{\partial t} + \sigma \mathbf{E} + \mathbf{J} \right).$$

Так как $\nabla \cdot (\nabla \times \mathbf{H}) = 0$, $\nabla \cdot \frac{\partial \mathbf{D}}{\partial t} = \nabla \cdot \frac{\partial \varepsilon \mathbf{E}}{\partial t} = \nabla \cdot (i\omega \varepsilon \mathbf{E})$ и, так как для замкнутой петли с током выполняется $\nabla \cdot \mathbf{J} = 0$, получим закон сохранения заряда:

$$\nabla \cdot (\sigma + i\omega \varepsilon) \mathbf{E} = 0. \quad (12)$$

1.2. Вариационная постановка

Введём следующие пространства [8, 9]:

$$\mathbb{H}(\text{rot}, \Omega) = \{ \mathbf{v} \in [\mathbb{L}^2(\Omega)]^3 : \nabla \times \mathbf{v} \in [\mathbb{L}^2(\Omega)]^3 \},$$

$$\mathbb{H}_0(\text{rot}, \Omega) = \{ \mathbf{v} \in \mathbb{H}(\text{rot}, \Omega) : \mathbf{v} \times \mathbf{n}|_{\partial\Omega} = 0 \}.$$

Скалярное произведение в этих пространствах имеет вид:

$$(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \mathbf{u} \cdot \bar{\mathbf{v}} d\Omega.$$

Скалярно умножим (8) на некоторую пробную функцию $\mathbf{v} \in \mathbb{H}_0(\text{rot}, \Omega)$:

$$(\nabla \times (\mu^{-1} \nabla \times \mathbf{E}), \mathbf{v}) + (k^2 \mathbf{E}, \mathbf{v}) = -(i\omega \mathbf{J}, \mathbf{v}),$$

$$\int_{\Omega} \nabla \times (\mu^{-1} \nabla \times \mathbf{E}) \cdot \bar{\mathbf{v}} d\Omega + \int_{\Omega} k^2 \mathbf{E} \cdot \bar{\mathbf{v}} d\Omega = - \int_{\Omega} i\omega \mathbf{J} \cdot \bar{\mathbf{v}} d\Omega.$$

Воспользовавшись первой векторной формулой Грина (13):

$$\int_D \nabla \times \mathbf{u} \cdot \bar{\mathbf{v}} dV = \int_D \mathbf{u} \cdot (\nabla \times \bar{\mathbf{v}}) dV + \int_{\partial D} (\mathbf{n} \times \mathbf{u}) \cdot \bar{\mathbf{v}} dS, \quad (13)$$

получим:

$$\begin{aligned} & \int_{\Omega} \mu^{-1} \nabla \times \mathbf{E} \cdot \nabla \times \bar{\mathbf{v}} d\Omega + \int_{\partial\Omega} \mathbf{n} \times (\mu^{-1} \nabla \times \mathbf{E}) \cdot \bar{\mathbf{v}} dS + \\ & + \int_{\Omega} k^2 \mathbf{E} \cdot \bar{\mathbf{v}} d\Omega = - \int_{\Omega} i\omega \mathbf{J} \cdot \bar{\mathbf{v}} d\Omega. \end{aligned}$$

Применим тождества $(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c} = -(\mathbf{a} \times \mathbf{c}) \cdot \mathbf{b}$ и $\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$:

$$\begin{aligned} & \int_{\Omega} \mu^{-1} \nabla \times \mathbf{E} \cdot \nabla \times \bar{\mathbf{v}} d\Omega + \int_{\Omega} k^2 \mathbf{E} \cdot \bar{\mathbf{v}} d\Omega = \\ & = - \int_{\Omega} i\omega \mathbf{J} \cdot \bar{\mathbf{v}} d\Omega - \int_{\partial\Omega} \bar{\mathbf{v}} \times \mathbf{n} \cdot (\mu^{-1} \nabla \times \mathbf{E}) dS. \end{aligned} \quad (14)$$

Так как $\mathbf{v} \in \mathbb{H}_0(\text{rot}, \Omega)$, то из свойств пространства $\mathbb{H}_0(\text{rot}, \Omega)$ второй интеграл в правой части равен нулю, тогда уравнение (14) примет вид:

$$\int_{\Omega} \mu^{-1} \nabla \times \mathbf{E} \cdot \nabla \times \bar{\mathbf{v}} d\Omega + \int_{\Omega} k^2 \mathbf{E} \cdot \bar{\mathbf{v}} d\Omega = - \int_{\Omega} i\omega \mathbf{J} \cdot \bar{\mathbf{v}} d\Omega. \quad (15)$$

В результате векторная вариационная постановка имеет вид: **Найти** $\mathbf{E} \in \mathbb{H}_0(\text{rot}, \Omega)$, **такое что** $\forall \mathbf{v} \in \mathbb{H}_0(\text{rot}, \Omega)$ **будет выполнено** (15).

Введём ещё два пространства [9]

$$\mathbb{H}(\text{grad}, \Omega) = \{\varphi \in \mathbb{L}^2(\Omega) : \nabla \varphi \in [\mathbb{L}^2(\Omega)]^3\},$$

$$\mathbb{H}_0(\text{grad}, \Omega) = \{\varphi \in \mathbb{H}(\text{grad}, \Omega) : \varphi|_{\partial\Omega} = 0\}.$$

В соответствии с комплексом Де Рама (De Rham) [15]

$$\mathbb{H}(\text{grad}, \Omega) \xrightarrow{\nabla} \mathbb{H}(\text{rot}, \Omega) \xrightarrow{\nabla \times} \mathbb{H}(\text{div}, \Omega) \xrightarrow{\nabla \cdot} \mathbb{L}^2(\Omega), \quad (16)$$

будет иметь место вложение $\nabla\varphi \in \mathbb{H}_0(\text{rot}, \Omega)$, $\forall \varphi \in \mathbb{H}_0(\text{grad}, \Omega)$. Возьмём $\mathbf{v} = \nabla\varphi$, тогда (15) примет вид:

$$\int_{\Omega} \mu^{-1} \nabla \times \mathbf{E} \cdot \nabla \times (\nabla\varphi) d\Omega + \int_{\Omega} k^2 \mathbf{E} \cdot (\nabla\varphi) d\Omega = - \int_{\Omega} i\omega \mathbf{J} \cdot (\nabla\varphi) d\Omega.$$

Используя свойство дивергенции $\nabla \cdot (\varphi \mathbf{F}) = \nabla\varphi \cdot \mathbf{F} + \varphi \nabla \cdot \mathbf{F}$ и применив формулу Остроградского-Гаусса (17)

$$\int_D \nabla \cdot \mathbf{F} dV = \int_{\partial D} \mathbf{F} \cdot \mathbf{n} dS, \quad (17)$$

получим:

$$\int_{\Omega} \mu^{-1} \nabla \times \mathbf{E} \cdot \nabla \times (\nabla\varphi) d\Omega + \int_{\Omega} k^2 \mathbf{E} \cdot (\nabla\varphi) d\Omega = - \int_{\Omega} i\omega \bar{\varphi} \nabla \cdot \mathbf{J} d\Omega - \int_{\partial\Omega} i\omega \bar{\varphi} \mathbf{J} \cdot \mathbf{n} dS.$$

Поскольку $\nabla \times (\nabla\varphi) = 0$, $\nabla \cdot \mathbf{J} = 0$ и $\varphi|_{\partial\Omega} = 0$, в левой части останется только один интеграл:

$$\int_{\Omega} k^2 \mathbf{E} \cdot (\nabla\varphi) d\Omega = 0.$$

После преобразований получим:

$$\int_{\Omega} \bar{\varphi} \nabla \cdot (k^2 \mathbf{E}) d\Omega = 0,$$

следовательно, решение задачи (15) будет в слабом смысле удовлетворять закону сохранения заряда (12) [14].

1.3. Вариационная постановка с учётом PML-слоя

Для ограничения расчётной области введём PML-слой Ω^{PML} , который является подобластью основной расчётной области Ω со специальными коэффициентами, построенными таким образом, чтобы обеспечить полное поглощение электрического поля внутри слоя и не допустить его отражения от внутренних границ и прохождения через внешние границы слоя (рисунок 2).

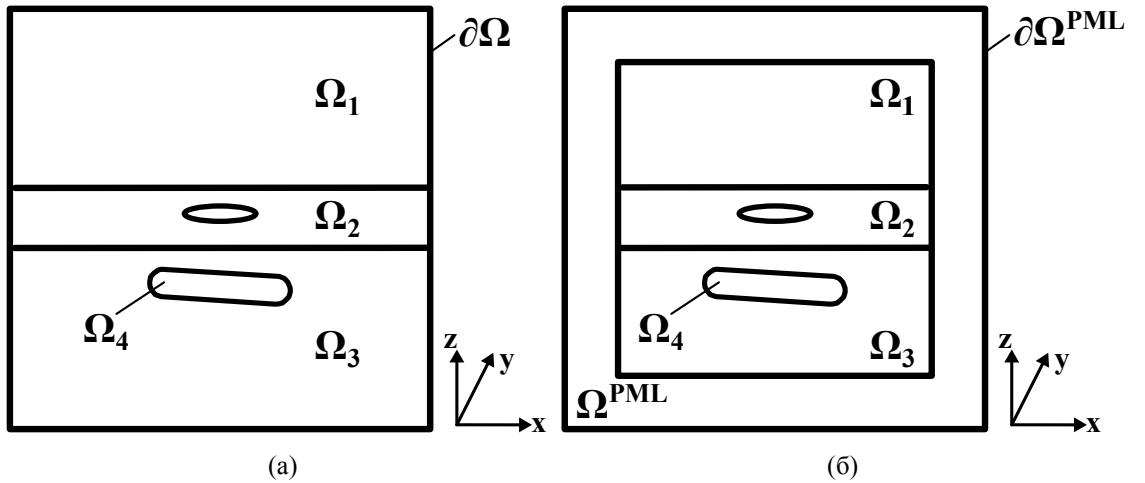


Рисунок 2 — расчётные области: (а) без PML-слоя и (б) с PML-слоем

PML-слой определяется модифицированными координатами $\tilde{x}, \tilde{y}, \tilde{z}$, полученными следующей заменой координат [7]:

$$\tilde{x} = \int_0^x s_x(t) dt, \quad \tilde{y} = \int_0^y s_y(t) dt, \quad \tilde{z} = \int_0^z s_z(t) dt,$$

где $s_j(\tau) = 1$ вне PML-слоя, а внутри него может быть задано в виде:

$$s_j(\tau) = 1 + \chi \left(\frac{d(\tau)}{\delta} \right)^m, \quad m \geq 1, \quad (18)$$

где $d(\tau)$ – расстояние в j -м направлении от внутренней границы PML-слоя, δ – толщина PML-слоя, χ – некоторое комплексное число, причём $\text{Re}(\chi) \geq 0$, $\text{Im}(\chi) \geq 0$. Оператор ∇ в новых координатах будет иметь вид:

$$\tilde{\nabla} = \left[\frac{1}{s_x} \frac{\partial}{\partial x}, \frac{1}{s_y} \frac{\partial}{\partial y}, \frac{1}{s_z} \frac{\partial}{\partial z} \right].$$

После такой замены, внутри PML-слоя уравнение Гельмгольца (8) будет иметь вид (19)

$$\tilde{\nabla} \times (\mu^{-1} \tilde{\nabla} \times \tilde{\mathbf{E}}) + k^2 \tilde{\mathbf{E}} = 0, \quad (19)$$

что приведёт к преобразованию уравнения (15) к виду (20):

$$\int_{\Omega^{PML}} \mu^{-1} \tilde{\nabla} \times \tilde{\mathbf{E}} \cdot \tilde{\nabla} \times \bar{\mathbf{v}} d\Omega^{PML} + \int_{\Omega^{PML}} k^2 \tilde{\mathbf{E}} \cdot \bar{\mathbf{v}} d\Omega^{PML} = 0. \quad (20)$$

В результате, если обозначить $\widehat{\Omega} = \Omega \setminus \Omega^{PML}$, то векторная вариационная постановка с учётом PML-слоя примет вид: **Найти** $\mathbf{E} \in \mathbb{H}_0(\text{rot}, \widehat{\Omega})$ и $\tilde{\mathbf{E}} \in \mathbb{H}_0(\text{rot}, \Omega^{PML})$, **такие что** $\forall \mathbf{v} \in \mathbb{H}_0(\text{rot}, \widehat{\Omega})$ и $\forall \tilde{\mathbf{v}} \in \mathbb{H}_0(\text{rot}, \Omega^{PML})$ **будем выполнено:**

$$\begin{cases} \int_{\widehat{\Omega}} \mu^{-1} \nabla \times \mathbf{E} \cdot \nabla \times \bar{\mathbf{v}} d\widehat{\Omega} + \int_{\widehat{\Omega}} k^2 \mathbf{E} \cdot \bar{\mathbf{v}} d\widehat{\Omega} = - \int_{\widehat{\Omega}} i\omega \mathbf{J} \cdot \bar{\mathbf{v}} d\widehat{\Omega} \\ \int_{\Omega^{PML}} \mu^{-1} \tilde{\nabla} \times \tilde{\mathbf{E}} \cdot \tilde{\nabla} \times \tilde{\mathbf{v}} d\Omega^{PML} + \int_{\Omega^{PML}} k^2 \tilde{\mathbf{E}} \cdot \tilde{\mathbf{v}} d\Omega^{PML} = 0. \end{cases}$$

1.4. Дискретная вариационная постановка

Разобьём область Ω на m непересекающихся элементов:

$$\Omega = \bigcup_{k=1}^m \Omega_k, \quad \forall i \neq j, \quad \Omega_i \cap \Omega_j = \emptyset.$$

Введём конечномерные подпространства:

$$\mathbb{H}_0^h(\text{rot}, \Omega) \subset \mathbb{H}_0(\text{rot}, \Omega), \quad \mathbb{H}_0^h(\text{grad}, \Omega) \subset \mathbb{H}_0(\text{grad}, \Omega).$$

Для дискретных подпространств $\mathbb{H}_0^h(\text{rot}, \Omega)$ и $\mathbb{H}_0^h(\text{grad}, \Omega)$ комплекс Де Рама (16) также будет верен, следовательно, закон сохранения заряда (12) будет также выполнен в слабом смысле [14].

Пространство $\mathbb{H}_0^h(\text{rot}, \Omega)$ является прямой суммой подпространств [14, 16]

$$\mathbb{H}_0^h(\text{rot}, \Omega) = \mathbb{N}_0^h(\text{rot}, \Omega) \oplus (\mathbb{N}_0^h(\text{rot}, \Omega))^{\perp}, \quad (21)$$

где $\mathbb{N}_0^h(\text{rot}, \Omega)$ – ядро rot-оператора, $(\mathbb{N}_0^h(\text{rot}, \Omega))^{\perp}$ – его ортогональное дополнение. Для выполнения условий непрерывности (3)-(4) необходимо использовать полный базис (базис II типа) [14, 17–19, 22], состоящий из двух типов базисных функций. Первый тип – роторные базисные функции из пространства $(\mathbb{N}_0^h(\text{rot}, \Omega))^{\perp}$, которые обеспечивают непрерывность тангенциальных компонент электрического поля \mathbf{E} (3). Второй – градиентные базисные функции из пространства $\mathbb{N}_0^h(\text{rot}, \Omega)$, которые обеспечивают скачок нормальной компоненты электрического поля \mathbf{E} (4) и выполнение закона сохранения заряда (12).

Представим векторнозначную функцию \mathbf{E}^h в виде разложения по базису $\psi_j \in \mathbb{H}_0^h(\text{rot}, \Omega)$:

$$\mathbf{E}^h = \sum_{j=1}^n q_j \psi_j.$$

В качестве тестовой функции выберем базисную функцию $\psi_i \in \mathbb{H}_0^h(\text{rot}, \Omega)$, тогда конечноэлементная аппроксимация вариационного уравнения (15) примет вид:

$$\begin{aligned} \sum_{j=1}^n \left(\int_{\Omega} \mu^{-1} \nabla \times \psi_j \cdot \nabla \times \psi_i d\Omega + \int_{\Omega} k^2 \psi_j \cdot \psi_i d\Omega \right) q_j &= \\ &= - \int_{\Omega} i\omega \mathbf{J} \cdot \psi_i d\Omega. \end{aligned} \quad (22)$$

В матрично-векторной форме (22) можно представить в виде следующей системы линейных алгебраических уравнений (СЛАУ):

$$(\mathbf{G} + \mathbf{M})\mathbf{q} = \mathbf{f}, \quad (23)$$

$$\mathbf{G}_{i,j} = \int_{\Omega_k} \mu^{-1} \nabla \times \mathbf{w}_i \cdot \nabla \times \mathbf{w}_j d\Omega_k, \quad \mathbf{M}_{i,j} = \int_{\Omega_k} k^2 \mathbf{w}_i \cdot \mathbf{w}_j d\Omega_k.$$

Матрица СЛАУ будет иметь симметричную разреженную структуру, поэтому её удобно хранить в формате CSLR (Compressed Sparse (Lower triangle) Row) или CSR (Compressed Sparse Row) [20].

1.5. Тетраэдральные конечные элементы

В качестве конечных элементов для представления расчётной области, будем пользоваться тетраэдрами. На тетраэдральном конечном элементе определим \mathcal{L} -координаты, называемые также барицентрическими координатами [21]. Введём нумерацию вершин и рёбер, показанную на рисунке 3:

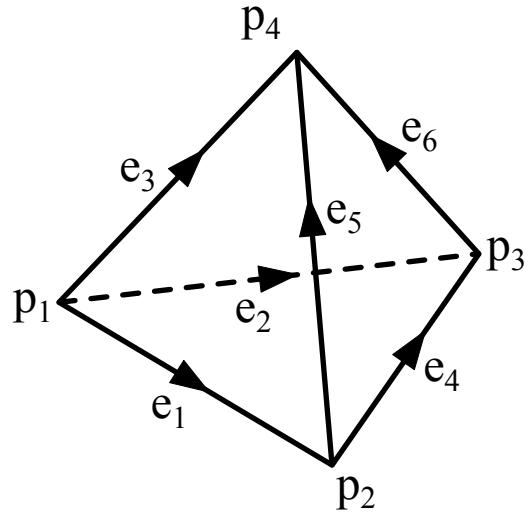


Рисунок 3 — тетраэдральный конечный элемент

Под \mathcal{L} -координатами понимают функции следующего вида:

$$\mathcal{L}_i(x, y, z) = \alpha_{i,1}x + \alpha_{i,2}y + \alpha_{i,3}z + \alpha_{i,4}, \quad i = \overline{1..4}.$$

Коэффициенты $\alpha_{i,j}$ могут быть определены по формуле (24):

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} \end{bmatrix} = \begin{bmatrix} p_{1x} & p_{2x} & p_{3x} & p_{4x} \\ p_{1y} & p_{2y} & p_{3y} & p_{4y} \\ p_{1z} & p_{2z} & p_{3z} & p_{4z} \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1}. \quad (24)$$

Задав \mathcal{L} -координаты, можно определить на тетраэдре базисные функции. В отличие от узлового метода конечных элементов, в векторном методе конечных

элементов базисные функции ассоциированы не с узлами, а с рёбрами (edge), гранями (face) и объёмами (volume) [22, 23]. Так как будут использованы полные (II типа) базисы первого и второго порядков, то ограничимся рассмотрением только базисных функций, ассоциированных с рёбрами и гранями.

Иерархический векторный базис Вебба второго порядка второго типа на тетраэдрах имеет вид [24]:

$$\begin{aligned}
 \mathbf{w}_i^{1,\text{I}} &= \mathcal{L}_k \nabla \mathcal{L}_l - \mathcal{L}_l \nabla \mathcal{L}_k; \quad i=1, \dots, 6; \quad k, l = 1, \dots, 4; \quad k < l, \\
 \mathbf{w}_i^{1,\text{II}} &= \mathcal{L}_k \nabla \mathcal{L}_l + \mathcal{L}_l \nabla \mathcal{L}_k; \quad i=7, \dots, 12; \quad k, l = 1, \dots, 4; \quad k < l, \\
 \mathbf{w}_i^{2,\text{I}} &= \mathcal{L}_k \mathcal{L}_l \nabla \mathcal{L}_j + \mathcal{L}_j \mathcal{L}_l \nabla \mathcal{L}_k - 2\mathcal{L}_j \mathcal{L}_k \nabla \mathcal{L}_l; \quad i=13, \dots, 16; \quad j, k, l = 1, \dots, 4; \quad j < k < l, \\
 \mathbf{w}_i^{2,\text{I}} &= \mathcal{L}_k \mathcal{L}_l \nabla \mathcal{L}_j - 2\mathcal{L}_j \mathcal{L}_l \nabla \mathcal{L}_k + \mathcal{L}_j \mathcal{L}_k \nabla \mathcal{L}_l; \quad i=17, \dots, 20; \quad j, k, l = 1, \dots, 4; \quad j < k < l, \\
 \mathbf{w}_i^{2,\text{II}} &= \nabla(\mathcal{L}_j \mathcal{L}_k \mathcal{L}_l); \quad i=21, \dots, 24; \quad j, k, l = 1, \dots, 4; \quad j < k < l, \\
 \mathbf{w}_i^{2,\text{II}} &= \nabla(\mathcal{L}_j \mathcal{L}_k (\mathcal{L}_j - \mathcal{L}_k)); \quad i=25, \dots, 30; \quad j, k = 1, \dots, 4; \quad j < k,
 \end{aligned} \tag{25}$$

где $\mathbf{w}_1^{1,\text{I}}, \dots, \mathbf{w}_6^{1,\text{I}}$ – базисные функции первого порядка первого типа, ассоциированные с рёбрами, $\mathbf{w}_7^{1,\text{II}}, \dots, \mathbf{w}_{12}^{1,\text{II}}$ – базисные функции первого порядка второго типа, ассоциированные с рёбрами, $\mathbf{w}_{13}^{2,\text{I}}, \dots, \mathbf{w}_{20}^{2,\text{I}}$ – базисные функции второго порядка первого типа, ассоциированные с гранями, $\mathbf{w}_{21}^{2,\text{II}}, \dots, \mathbf{w}_{24}^{2,\text{II}}$ – базисные функции второго порядка второго типа, ассоциированные с гранями, $\mathbf{w}_{25}^{2,\text{II}}, \dots, \mathbf{w}_{30}^{2,\text{II}}$ – базисные функции второго порядка второго типа, ассоциированные с рёбрами. Так как этот базис иерархический, то для получения базиса меньшего порядка следует ограничиться меньшим количеством функций. Так, для базиса первого порядка второго типа следует использовать функции $\mathbf{w}_1^{1,\text{I}}, \dots, \mathbf{w}_{12}^{1,\text{II}}$.

Для вычисления интегралов в (22) воспользуемся кубатурной формулой численного интегрирования (формулой Гаусса) [25]:

$$\int_{\Omega_k} f(x, y, z) d\Omega_k = \sum_{i=1}^m f(x_i, y_i, z_i) w_i,$$

где (x_i, y_i, z_i) – точки Гаусса, m – число точек Гаусса, w_i – соответствующие веса. При работе с базисными функциями второго порядка нужно использовать формулы, которые бы обеспечивали восьмой порядок интегрирования [26]. Для

базисных функций первого порядка будет достаточно и меньших порядков интегрирования [25, 27].

1.6. Треугольные конечные элементы

Границы области Ω являются двумерными и представляют собой треугольники. Для учёта краевых условий (9) требуется построить разложение \mathbf{E}^g по базису соответствующей границы в смысле МНК, для этого нужно решать СЛАУ вида

$$\mathbf{M}^{S_1} \tilde{\mathbf{q}} = \mathbf{b}^{S_1}, \quad (26)$$

где $\mathbf{M}_{i,j}^{S_1} = \int_{S_1} \tilde{\mathbf{w}}_i \cdot \tilde{\mathbf{w}}_j dS_1$, $\mathbf{b}_i^{S_1} = \int_{S_1} \mathbf{E}^g \cdot \tilde{\mathbf{w}}_i dS_1$, $\tilde{\mathbf{w}}_i$ и $\tilde{\mathbf{w}}_j$ – базисные функции на треугольниках.

Определим \mathcal{L} -координаты на треугольниках таким же образом, как и на тетраэдрах. Введём нумерацию вершин и рёбер согласно рисунку 4:

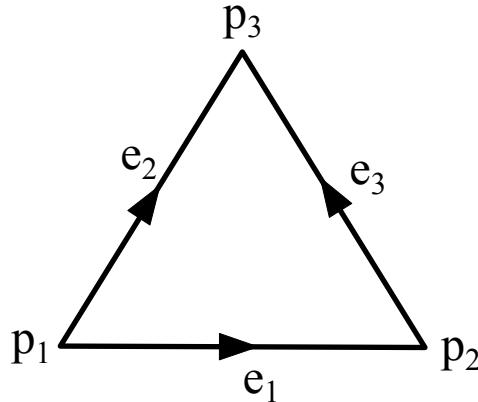


Рисунок 4 — треугольный конечный элемент

Тогда \mathcal{L} -координаты примут вид:

$$\mathcal{L}_i(x, y) = \alpha_{i,1}x + \alpha_{i,2}y + \alpha_{i,3}, \quad i = \overline{1..3}.$$

Коэффициенты $\alpha_{i,j}$ могут быть определены по формуле (27):

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{bmatrix} = \begin{bmatrix} p_{1x} & p_{2x} & p_{3x} \\ p_{1y} & p_{2y} & p_{3y} \\ 1 & 1 & 1 \end{bmatrix}^{-1}. \quad (27)$$

Иерархический векторный базис Вебба второго порядка второго типа на треугольниках имеет вид:

$$\begin{aligned}
\tilde{\mathbf{w}}_i^{1,\text{I}} &= \mathcal{L}_k \nabla \mathcal{L}_l - \mathcal{L}_l \nabla \mathcal{L}_k; \quad i=1, \dots, 3; \quad k, l=1, \dots, 3; \quad k < l, \\
\tilde{\mathbf{w}}_i^{1,\text{II}} &= \mathcal{L}_k \nabla \mathcal{L}_l + \mathcal{L}_l \nabla \mathcal{L}_k; \quad i=4, \dots, 6; \quad k, l=1, \dots, 3; \quad k < l, \\
\tilde{\mathbf{w}}_7^{2,\text{I}} &= \mathcal{L}_2 \mathcal{L}_3 \nabla \mathcal{L}_1 + \mathcal{L}_1 \mathcal{L}_3 \nabla \mathcal{L}_2 - 2 \mathcal{L}_1 \mathcal{L}_2 \nabla \mathcal{L}_3, \\
\tilde{\mathbf{w}}_8^{2,\text{I}} &= \mathcal{L}_2 \mathcal{L}_3 \nabla \mathcal{L}_1 - 2 \mathcal{L}_1 \mathcal{L}_3 \nabla \mathcal{L}_2 + \mathcal{L}_1 \mathcal{L}_2 \nabla \mathcal{L}_3, \\
\tilde{\mathbf{w}}_9^{2,\text{II}} &= \nabla(\mathcal{L}_1 \mathcal{L}_2 \mathcal{L}_3), \\
\tilde{\mathbf{w}}_i^{2,\text{II}} &= \nabla(\mathcal{L}_j \mathcal{L}_k (\mathcal{L}_j - \mathcal{L}_k)); \quad i=10, \dots, 12; \quad j, k=1, \dots, 3; \quad j < k,
\end{aligned}$$

где $\tilde{\mathbf{w}}_1^{1,\text{I}}, \dots, \tilde{\mathbf{w}}_3^{1,\text{I}}$ – базисные функции первого порядка первого типа, $\tilde{\mathbf{w}}_4^{1,\text{II}}, \dots, \tilde{\mathbf{w}}_6^{1,\text{II}}$ – базисные функции первого порядка второго типа, $\tilde{\mathbf{w}}_7^{2,\text{I}}, \tilde{\mathbf{w}}_8^{2,\text{I}}$ – базисные функции второго порядка первого типа, $\tilde{\mathbf{w}}_9^{2,\text{II}}, \dots, \tilde{\mathbf{w}}_{12}^{2,\text{II}}$ – базисные функции второго порядка второго типа. Для базиса первого порядка второго типа следует использовать функции $\tilde{\mathbf{w}}_1^{1,\text{I}}, \dots, \tilde{\mathbf{w}}_6^{1,\text{II}}$.

Для вычисления интегралов в (26) воспользуемся формулой Гаусса [25]:

$$\int_{\Omega_k} f(x, y) d\Omega_k = \sum_{i=1}^m f(x_i, y_i) w_i,$$

где (x_i, y_i) – точки Гаусса, m – число точек Гаусса, w_i – соответствующие веса. Так же, как и для тетраэдров, для работы с базисом второго порядка нужно использовать формулы, обеспечивающие восьмой порядок интегрирования [26]. Для базиса первого порядка достаточно и меньших порядков интегрирования [25, 28].

1.7. Двухуровневый решатель

Так как пространство $\mathbb{H}_0^h(\text{rot}, \Omega)$ представимо в виде прямой суммы подпространств (21), можно построить специальный двухуровневый решатель, который учитывает ядро rot-оператора. Принцип построения подробно изложен в [23, 29].

Рассмотрим иерархический узловой базис пространства $\mathbb{H}_0^h(\text{grad}, \Omega)$ на тетраэдре:

$$\begin{aligned}\phi_i &= \mathcal{L}_i; \quad i=1,\dots,4, \\ \phi_i &= \mathcal{L}_k \mathcal{L}_l; \quad i=5,\dots,10; \quad k,l=1,\dots,4; \quad k < l, \\ \phi_i &= \mathcal{L}_j \mathcal{L}_k \mathcal{L}_l; \quad i=11,\dots,14; \quad j,k,l=1,\dots,4; \quad j < k < l, \\ \phi_i &= \mathcal{L}_j \mathcal{L}_k (\mathcal{L}_j - \mathcal{L}_k); \quad i=15,\dots,20; \quad j,k=1,\dots,4; \quad j < k.\end{aligned}$$

Градиенты этих базисных функций принадлежат пространству $\mathbb{H}_0^h(\text{rot}, \Omega)$:

$$\begin{aligned}\nabla \phi_1 &= \nabla \mathcal{L}_1 = -\mathbf{w}_1^{1,I} - \mathbf{w}_2^{1,I} - \mathbf{w}_3^{1,I}, \\ \nabla \phi_2 &= \nabla \mathcal{L}_2 = \mathbf{w}_1^{1,I} - \mathbf{w}_4^{1,I} + \mathbf{w}_5^{1,I}, \\ \nabla \phi_3 &= \nabla \mathcal{L}_3 = \mathbf{w}_2^{1,I} + \mathbf{w}_4^{1,I} - \mathbf{w}_6^{1,I}, \\ \nabla \phi_4 &= \nabla \mathcal{L}_4 = \mathbf{w}_3^{1,I} - \mathbf{w}_5^{1,I} + \mathbf{w}_6^{1,I}, \\ \nabla \phi_i &= \nabla (\mathcal{L}_k \mathcal{L}_l) = \mathbf{w}_{i+2}^{1,II}; \quad i=5,\dots,10; \quad k,l=1,\dots,4; \quad k < l, \\ \nabla \phi_i &= \nabla (\mathcal{L}_j \mathcal{L}_k \mathcal{L}_l) = \mathbf{w}_{i+10}^{2,II}; \quad i=11,\dots,14; \quad j,k,l=1,\dots,4; \quad j < k < l, \\ \nabla \phi_i &= \nabla (\mathcal{L}_j \mathcal{L}_k (\mathcal{L}_j - \mathcal{L}_k)) = \mathbf{w}_{i+10}^{2,II}; \quad i=15,\dots,20; \quad j,k=1,\dots,4; \quad j < k.\end{aligned}\tag{28}$$

Таким образом мы можем построить матрицу перехода \mathbf{P} от базиса \mathbf{w} (25) к базису $\nabla \phi$ (28). В силу иерархичности базисов, матрица \mathbf{P} будет иметь блочную структуру. Матрицы перехода для базисов различных порядков имеют вид:

$$\begin{aligned}\mathbf{P}^{1,I} &= \begin{bmatrix} -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \quad \mathbf{P}^{1,II} = \begin{bmatrix} \mathbf{P}^{1,I} & \mathbb{O}_{4 \times 6} \\ \mathbb{O}_{6 \times 6} & \mathbf{I}_{6 \times 6} \end{bmatrix}, \\ \mathbf{P}^{2,I} &= \begin{bmatrix} \mathbf{P}^{1,II} & \mathbb{O}_{10 \times 8} \end{bmatrix}, \quad \mathbf{P}^{2,II} = \begin{bmatrix} \mathbf{P}^{2,I} & \mathbb{O}_{10 \times 10} \\ \mathbb{O}_{10 \times 20} & \mathbf{I}_{10 \times 10} \end{bmatrix},\end{aligned}$$

где $\mathbf{I}_{n \times n}$ – единичная матрица размерностью $n \times n$, $\mathbb{O}_{n \times m}$ – нулевая матрица размерностью $n \times m$.

Общий алгоритм многоуровневых методов имеет следующий вид:

1. Выбрать некоторое начальное приближение;
2. Уточнить решение на полном пространстве (подавить высокочастотную составляющую ошибки);
3. Спроектировать приближение с шага 2 на грубое подпространство;
4. Уточнить решение на грубом подпространстве (подавить низкочастотную составляющую ошибки);
5. Интерполировать решение на полное подпространство;
6. Повторить шаги 2 - 5 пока не будет достигнута нужная точность.

Псевдокод алгоритма двухуровневого метода:

Входные данные: $\mathbf{A}, \mathbf{f}, \mathbf{q}_0, \mathbf{P}, \gamma, \varepsilon_1, \varepsilon_2$

Результат: \mathbf{q}

$$\mathbf{r}_0 = \mathbf{f} - \mathbf{A}\mathbf{q}_0$$

цикл $i = 1, 2, \dots$ **пока** $\|\mathbf{r}_i\| > \gamma \|\mathbf{r}_0\|$ **выполнять**

$$\tilde{\mathbf{z}} = \text{solve}(\mathbf{P}\mathbf{A}\mathbf{P}^T, \mathbf{P}\mathbf{r}_{i-1}, \varepsilon_1)$$

$$\tilde{\mathbf{q}}_i = \mathbf{q}_{i-1} + \mathbf{P}^T \tilde{\mathbf{z}}$$

$$\tilde{\mathbf{r}}_i = \mathbf{f} - \mathbf{A}\tilde{\mathbf{q}}_i$$

$$\mathbf{z} = \text{solve}(\mathbf{A}, \tilde{\mathbf{r}}_i, \varepsilon_2)$$

$$\mathbf{q}_i = \tilde{\mathbf{q}}_i + \mathbf{z}$$

$$\mathbf{r}_i = \mathbf{f} - \mathbf{A}\mathbf{q}_i$$

конец

$$\mathbf{q} = \mathbf{q}_i$$

где \mathbf{A} и \mathbf{f} – матрица и правая часть СЛАУ (23); \mathbf{q}_0 – начальное приближение; \mathbf{P} – матрица перехода от базиса \mathbf{w} (25) к базису $\nabla\phi$ (28); γ – точность решения СЛАУ; ε_1 и ε_2 – точности решения промежуточных СЛАУ на подпространстве ядра и полном пространстве соответственно; $\text{solve}(\mathbf{A}, \mathbf{b}, \varepsilon)$ – процедура решения СЛАУ с матрицей \mathbf{A} и правой частью \mathbf{b} с точностью ε .

В качестве $\text{solve}(\mathbf{A}, \mathbf{b}, \varepsilon)$ можно воспользоваться методами на подпространствах Крылова [20, 30–32], например, GMRES или COCG. Величины ε_1 и ε_2 следует выбирать небольшими, от 0.9 до 0.01 [23].

2. Построение и решение СЛАУ

2.1. Структура глобальной матрицы СЛАУ

Рассмотрим структуру глобальной матрицы СЛАУ на примере двух тетраэдральных конечных элементов с базисом первого полного порядка, имеющих общую грань. Глобальная нумерация вершин и рёбер этих тетраэдров приведена на рисунке 5.

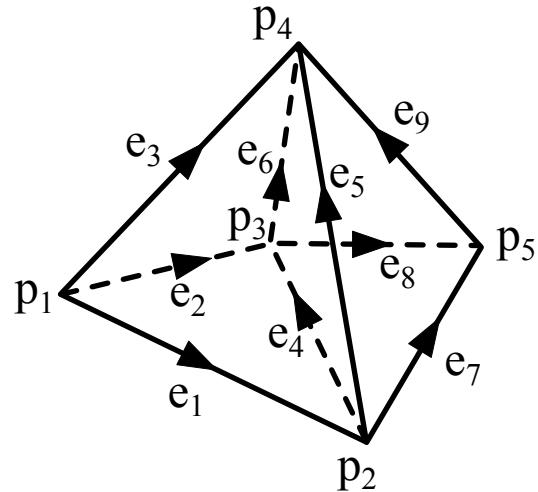


Рисунок 5 — два тетраэдральных конечных элемента

Локальные матрицы тетраэдра имеют блочный вид, матрица массы является плотной (29), а в матрице жёсткости ненулевой только один блок (30):

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^{I,I} & \mathbf{M}^{I,II} \\ \mathbf{M}^{II,I} & \mathbf{M}^{II,II} \end{bmatrix}, \quad (29)$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}^{I,I} & \mathbb{O} \\ \mathbb{O} & \mathbb{O} \end{bmatrix}, \quad (30)$$

где $\mathbf{M}^{I,I}$, $\mathbf{G}^{I,I}$ – блоки от интегралов, содержащих только роторные ($\mathbf{w}_1^{1,I}, \dots, \mathbf{w}_6^{1,I}$); $\mathbf{M}^{II,II}$ – только градиентные ($\mathbf{w}_7^{1,II}, \dots, \mathbf{w}_{12}^{1,II}$); $\mathbf{M}^{I,II}$, $\mathbf{M}^{II,I}$ – и роторные, и градиентные базисные функции.

Глобальная матрица также имеет блочную структуру, для двух тетраэдров она схематично приведена на рисунке 6. Под G_1 и G_2 понимаются элементы матриц жёсткости (30) первого и второго тетраэдров соответственно, под M_1 и

M_2 – элементы матриц массы (29) первого и второго тетраэдров соответственно.

$G_1 + M_1$				M_1	M_1	M_1	M_1	M_1	M_1								
$G_1 + M_1$				M_1	M_1	M_1	M_1	M_1	M_1								
$G_1 + M_1$				M_1	M_1	M_1	M_1	M_1	M_1								
$G_1 + M_1$	$G_2 + M_2$	$G_2 + M_2$	$G_2 + M_2$	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2					
$G_1 + M_1$	$G_1 + M_1$	$G_1 + M_1$	$G_2 + M_2$	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2					
$G_1 + M_1$	$G_1 + M_1$	$G_1 + M_1$	$G_2 + M_2$	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2					
$G_1 + M_1$	$G_1 + M_1$	$G_1 + M_1$	$G_2 + M_2$	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2					
			$G_2 + M_2$					M_2	M_2	M_2	M_2	M_2					
			$G_2 + M_2$					M_2	M_2	M_2	M_2	M_2					
			$G_2 + M_2$					M_2	M_2	M_2	M_2	M_2					
M_1	M_1	M_1	M_1	M_1	M_1				M_1	M_1	M_1	M_1	M_1	M_1			
M_1	M_1	M_1	M_1	M_1	M_1				M_1	M_1	M_1	M_1	M_1	M_1			
M_1	M_1	M_1	M_1	M_1	M_1				M_1	M_1	M_1	M_1	M_1	M_1			
M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2
M_1	M_1	M_1	M_1	M_2	M_2	M_2	M_2	M_2	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2
M_1	M_1	M_1	M_2	M_2	M_2	M_2	M_2	M_2	M_1	M_1	M_1	M_1	M_1	M_1	M_2	M_2	M_2
			M_2	M_2	M_2	M_2	M_2	M_2					M_2	M_2	M_2	M_2	M_2
			M_2	M_2	M_2	M_2	M_2	M_2					M_2	M_2	M_2	M_2	M_2
			M_2	M_2	M_2	M_2	M_2	M_2					M_2	M_2	M_2	M_2	M_2

Рисунок 6 — структура СЛАУ

2.2. Учёт краевых условий

Неоднородные краевые условия первого рода вида (9) учитываются путём обнуления внедиагональных элементов строк глобальной матрицы, соответствующих базисным функциям, ассоциированным с рёбрами и гранями сетки на границе, причём диагональные элементы приравниваются к 1, а элементы вектора правой части – к значениям, взятым из решения (26) [20].

Однородные краевые условия первого рода вида (11) учитываются аналогично, с той лишь разницей, что соответствующие элементы вектора правой части обнуляются вместе с элементами строки.

Однородные электрические краевые условия второго рода вида (10) учитываются естественным образом и не вносят вклад в матрицу или правую часть СЛАУ, поэтому не требуют какой-либо специальной процедуры учёта.

2.3. Учёт токовой петли

Петля с током представляется в виде бесконечно тонкого замкнутого контура, толщиной которого в данной задаче можно пренебречь. Для учёта такого источника необходимо при построении сетки задать необходимую конфигурацию рёбер, которая будет с достаточной точностью аппроксимировать геометрию контура. Далее необходимо разложить ток в петле по базисным функциям и учесть в виде добавки в правую часть, соблюдая правильную ориентацию рёбер в пространстве.

3. Вычислительные эксперименты

3.1. Верификация программного комплекса

Верификация полученной конечноэлементной аппроксимации будет проводиться на тестовой задаче, имеющей аналитическое решение.

3.1.1. Расчётная область

Расчётная область представляет собой куб со следующими параметрами: $x \in [0, 1]$, $y \in [0, 1]$, $z \in [0, 1]$. Куб разбивается на регулярную тетраэдральную сетку согласно рисунку 7:

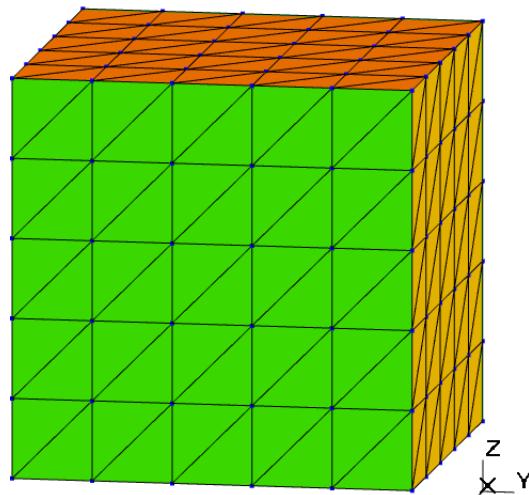


Рисунок 7 — конечноэлементная сетка для верификации

Всего в сетке 750 тетраэдров, 300 треугольников по границе, 216 узлов, 1115 рёбер и 1650 граней.

Физические параметры среды заданы следующим образом: $\varepsilon = \varepsilon_0$, $\mu = \mu_0$, $\sigma = 10$ См/м. Частота источника поля $\nu = \frac{100}{2\pi}$ Гц. На всех внешних гранях расчётной области заданы краевые условия первого рода (9).

3.1.2. Тестирование на гладких функциях

В качестве аналитического решения уравнения (8) выберем функцию

$$\mathbf{E} = \begin{pmatrix} e^{-(0.5-y)^2-(0.5-z)^2} \\ e^{-(0.5-x)^2-(0.5-z)^2} \\ e^{-(0.5-x)^2-(0.5-y)^2} \end{pmatrix}. \quad (31)$$

Тестирование будем проводить на базисных функциях первого и второго порядка второго типа. Погрешности в норме пространства \mathbb{L}^2 полученных решений приведены в таблице 1.

Таблица 1 — относительные погрешности в норме \mathbb{L}^2

Порядок базисных ф-й	$\frac{\ \mathbf{E} - \mathbf{E}^h\ _{\mathbb{L}^2}}{\ \mathbf{E}\ _{\mathbb{L}^2}}$	$\frac{\ \mathbf{E}_x - \mathbf{E}_x^h\ _{\mathbb{L}^2}}{\ \mathbf{E}_x\ _{\mathbb{L}^2}}$	$\frac{\ \mathbf{E}_y - \mathbf{E}_y^h\ _{\mathbb{L}^2}}{\ \mathbf{E}_y\ _{\mathbb{L}^2}}$	$\frac{\ \mathbf{E}_z - \mathbf{E}_z^h\ _{\mathbb{L}^2}}{\ \mathbf{E}_z\ _{\mathbb{L}^2}}$
1	6.608e-3	7.869e-3	5.877e-3	5.877e-3
2	1.775e-4	1.895e-4	1.712e-4	1.712e-4

Измельчим сетку расчётной области, изображённую на рисунке 7, в 2 и 4 раза. Погрешности в норме пространства \mathbb{L}^2 полученных решений и порядок аппроксимации – в таблице 2.

Таблица 2 — относительные погрешности в норме \mathbb{L}^2

Порядок базиса	$\frac{\ \mathbf{E} - \mathbf{E}^h\ _{\mathbb{L}^2}}{\ \mathbf{E}\ _{\mathbb{L}^2}}$	$\frac{\ \mathbf{E} - \mathbf{E}^{h/2}\ _{\mathbb{L}^2}}{\ \mathbf{E}\ _{\mathbb{L}^2}}$	$\frac{\ \mathbf{E} - \mathbf{E}^{h/4}\ _{\mathbb{L}^2}}{\ \mathbf{E}\ _{\mathbb{L}^2}}$	$\log_2 \frac{\ \mathbf{E} - \mathbf{E}^h\ _{\mathbb{L}^2}}{\ \mathbf{E} - \mathbf{E}^{h/2}\ _{\mathbb{L}^2}}$	$\log_2 \frac{\ \mathbf{E} - \mathbf{E}^{h/2}\ _{\mathbb{L}^2}}{\ \mathbf{E} - \mathbf{E}^{h/4}\ _{\mathbb{L}^2}}$
1	6.608e-3	1.637e-3	5.051e-4	2.013	1.696
2	1.775e-4	2.164e-5	3.582e-6	3.036	2.595

Из результатов видно, метод достаточно хорошо аппроксимировал гладкую функцию. Порядок аппроксимации получился второй для базисных функций первого порядка второго типа и третий для базисных функций второго порядка второго типа, что совпадает с теоретическими значениями.

3.2. Исследование влияния слоя воздуха

Проведём исследование влияния слоя воздуха в модельной задаче морской геоэлектрики при различной глубине погружения в воду источника электромагнитного возмущения.

В этом исследовании будем пользоваться базисными функциями второго полного порядка.

3.2.1. Описание расчётной области

Расчётная область показана на рисунке 8, где Ω_1 – воздух ($\sigma = 10^{-6}$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_2 – морская вода ($\sigma = 3.3$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_3 – грунт ($\sigma = 0.2$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_4 – углеводороды ($\sigma = 10^{-2}$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); L_1 , L_2 и L_3 – размеры области моделирования по осям x , y и z соответственно; $L_1 = L_2 = L_3 = 6000$ м; $h_1 = 600$ м – толщина Ω_2 ; $l_1 = 400$ м, $h_3 = 75$ м, $h_2 = 135$ м – длина, толщина и глубина объекта Ω_4 соответственно.

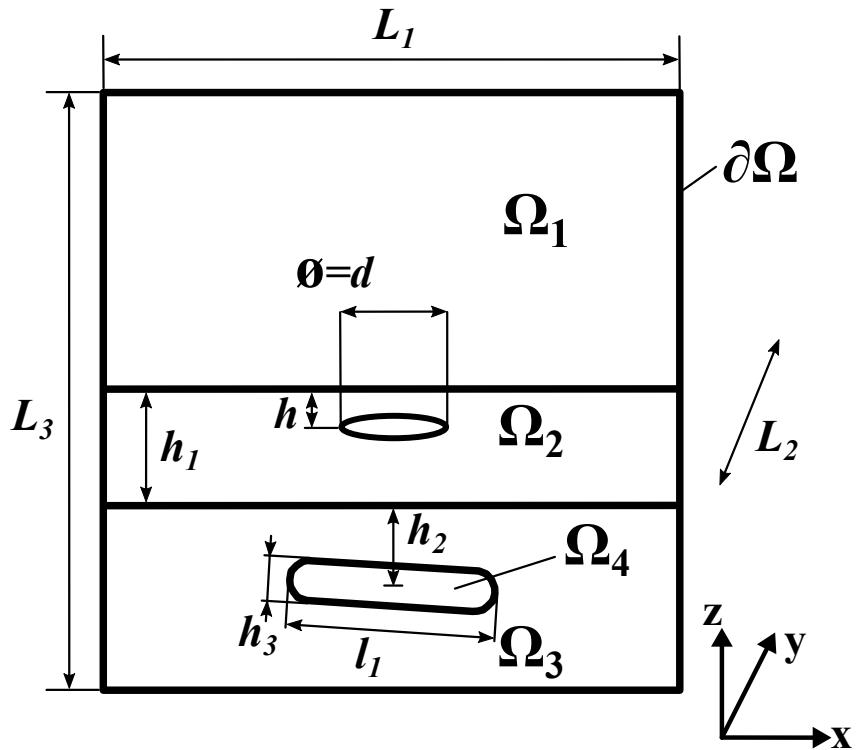


Рисунок 8 — расчётная область

Объект Ω_4 представляет собой скруглённый прямоугольный параллелепипед с двумя равными сторонами, наклонённый под углом 5° . Источником электрического поля является токовая петля диаметром $d = 100$ м с током частотой 1 Гц, глубина h которой варьируется в ходе исследования.

3.2.2. Конечноэлементная сетка

Фрагмент $x \in [-600, 0]$, $y \in [-600, 600]$ $z \in [-1000, 600]$ одной из конечно-элементных сеток, использованных для проведения исследования, представлен на рисунке 9.

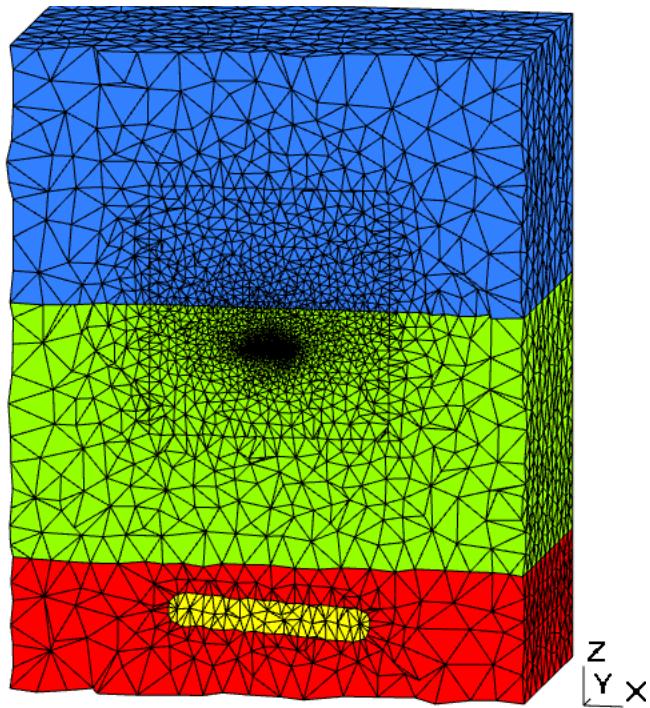


Рисунок 9 — фрагмент конечноэлементной сетки

3.2.3. Результаты исследования

Разности решений в норме \mathbb{L}^2 в объеме $[-600, 600] \times [-600, 600] \times [-1000, 0]$ между областью, в которой присутствует слой воздуха, и областью, в которой заданы условия непротекания (10), для некоторых значений глубины петли h показаны в таблице 3. В форме графика эти данные приведены на рисунке 10.

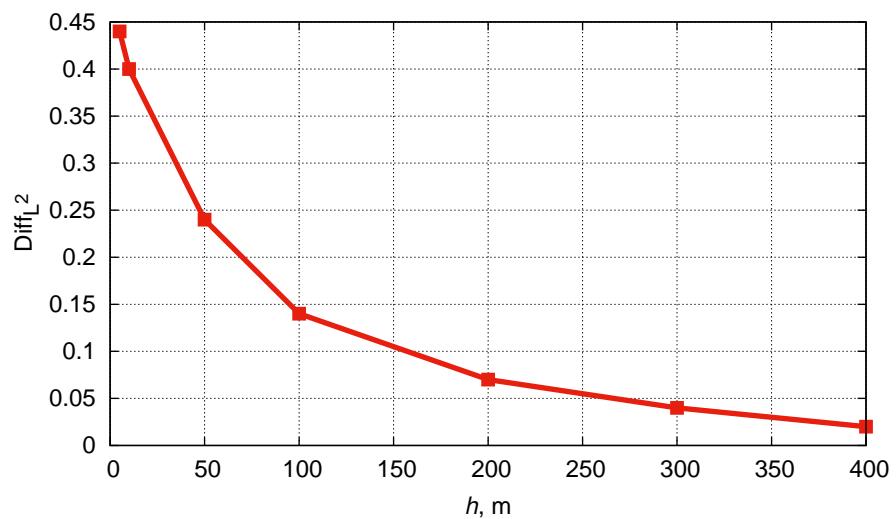


Рисунок 10 — график изменения относительной разности решений при изменении глубины

Таблица 3 — относительные разности решений

Глубина петли	5	10	50	100	200	300	400
$\frac{\ \mathbf{E}^{air} - \mathbf{E}^{noair}\ _{\mathbb{L}^2}}{\ \mathbf{E}^{air}\ _{\mathbb{L}^2}}$	0.44	0.40	0.24	0.14	0.07	0.04	0.02

Графики вещественной компоненты E_y по линии $y = 0, z = -610$ для различных глубин петли представлены на рисунках 11 и 12.

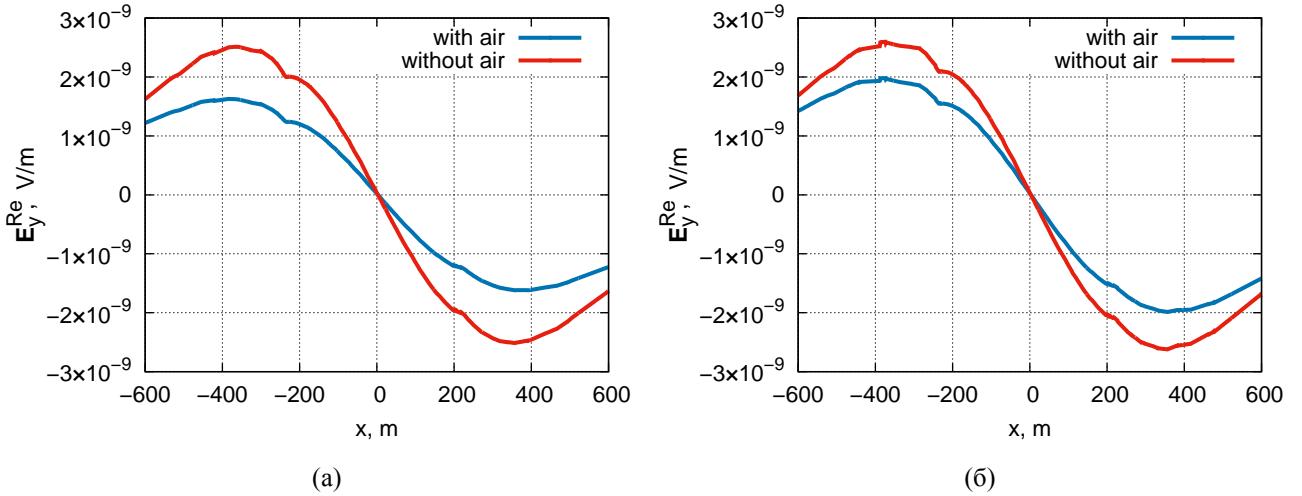


Рисунок 11 — $\text{Re}(\mathbf{E}_y)$ по линии $y = 0, z = -610$, глубина (а) 5 м и (б) 50 м

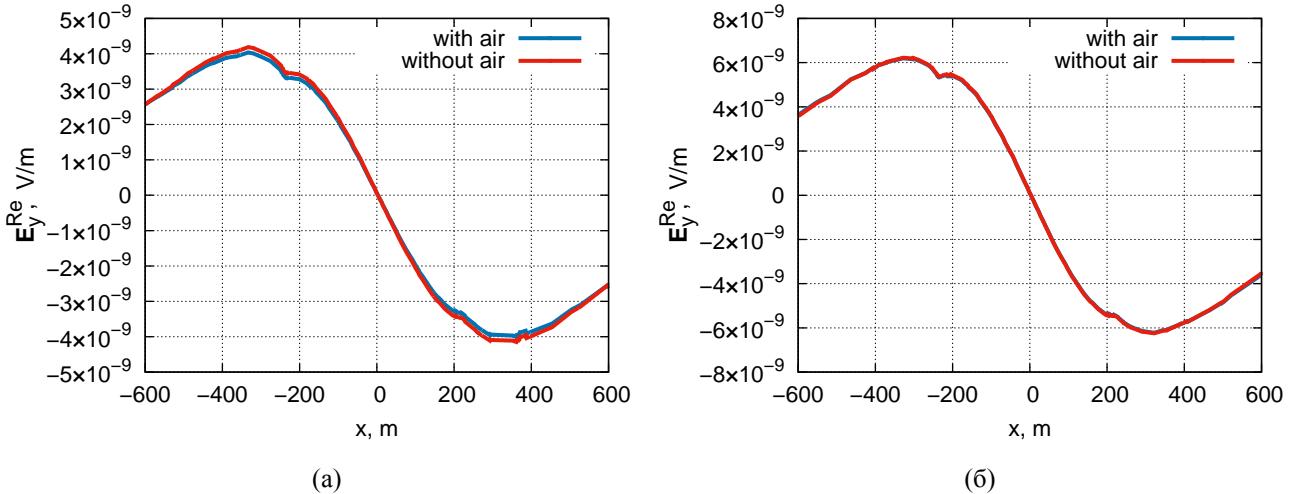


Рисунок 12 — $\text{Re}(\mathbf{E}_y)$ по линии $y = 0, z = -610$, глубина (а) 200 м и (б) 300 м

Из результатов следует, что слой воздуха оказывает значительное влияние на получаемое решение при расположении источника электромагнитного возмущения на малой глубине (меньше трёхсот метров для рассмотренной конфигурации).

3.3. Исследование эффективности применения PML-слоя

Цель вычислительных экспериментов: определение эффективности применения PML-слоя. Геометрические характеристики PML-слоя показаны на рисунке 13.

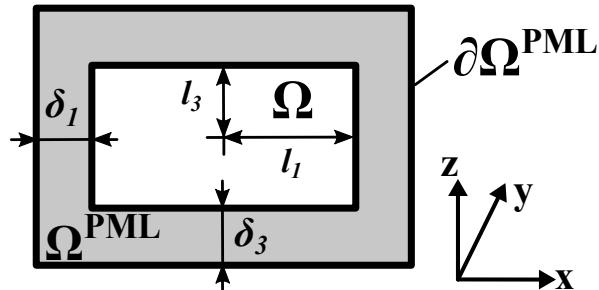


Рисунок 13 — геометрические характеристики PML-слоя

Вычислительные эксперименты проводились следующим образом: последовательно варьировался каждый из параметров PML-слоя: толщина слоя в k -м направлении δ_k , где $k = \frac{x}{1}, \frac{y}{2}, \frac{z}{3}$, расстояние от центра области до внутренних границ слоя l_k , коэффициент комплексного растяжения координат χ (18), оставшиеся параметры фиксировались, что позволило определить параметры, оказывающие наибольшее влияние на характеристики PML-слоя.

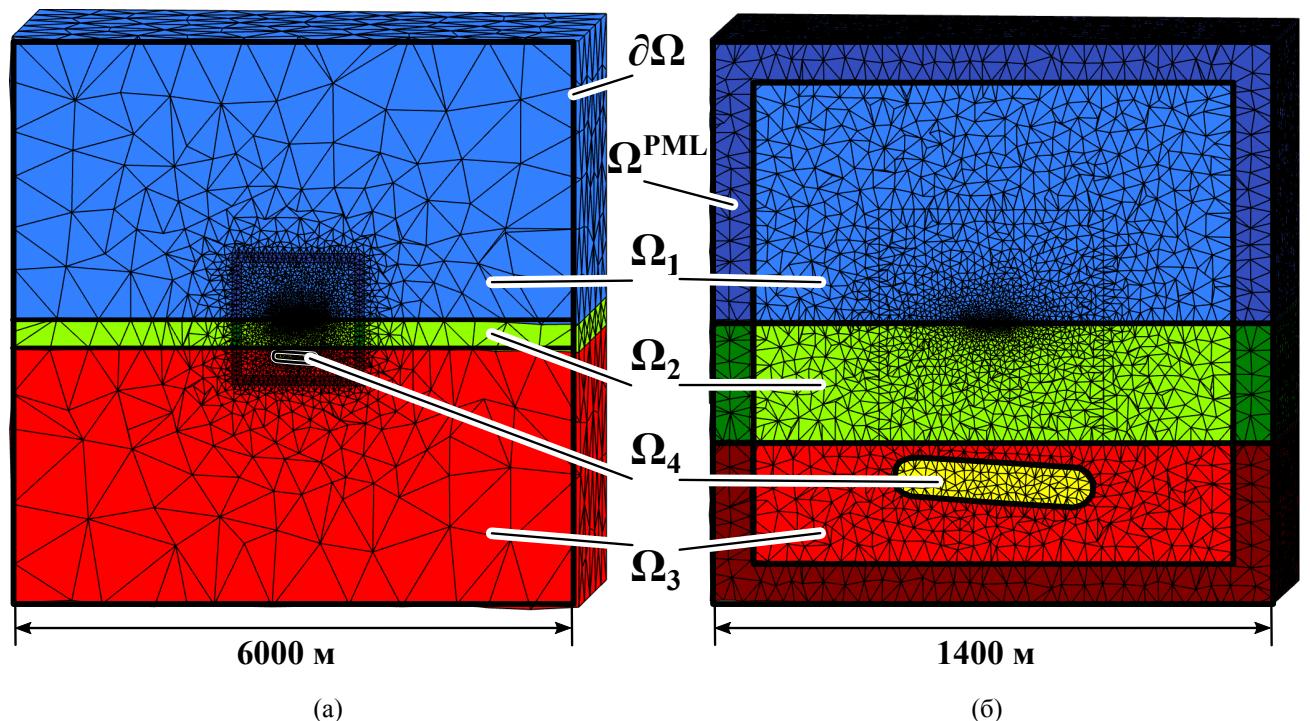


Рисунок 14 — конечноэлементные сетки: (а) «большой бак» и (б) PML-слой

Фрагменты тетраэдральных конечноэлементных сеток с «большим баком» и с PML-слоем приведены на рисунках 14а и 14б.

В этом исследовании будем пользоваться базисными функциями первого полного порядка.

3.3.1. Описание расчётной области

Расчётная область показана на рисунке 15, где Ω_1 – воздух ($\sigma = 10^{-6}$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_2 – морская вода ($\sigma = 3.3$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_3 – грунт ($\sigma = 0.2$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_4 – углеводороды ($\sigma = 10^{-2}$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); L_1 , L_2 и L_3 – размеры области моделирования по осям x , y и z соответственно; $L_1 = L_2 = L_3 = 6000$ м; $h_1 = 300$ м – толщина Ω_2 ; $l_1 = 400$ м, $h_2 = 100$ м – длина, толщина и глубина объекта Ω_4 соответственно.

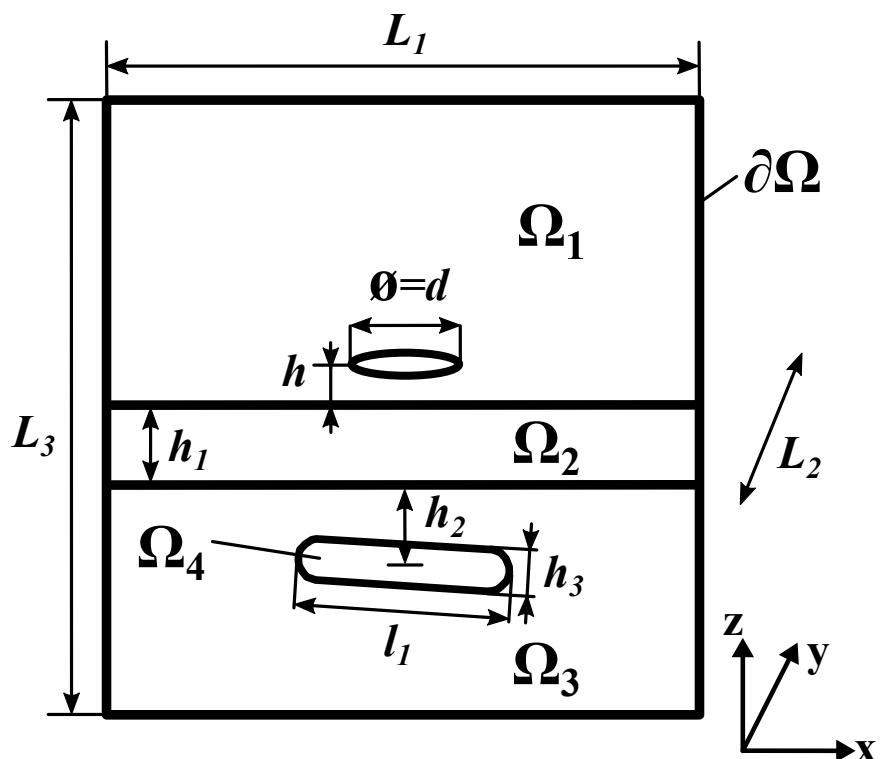


Рисунок 15 — расчётная область

Объект Ω_4 представляет собой скруглённый прямоугольный параллелепипед с двумя равными сторонами, наклонённый под углом 5° . В качестве источника электрического поля выступает петля диаметром $d = 100$ м с током частотой 1 Гц, расположенная в воздухе на расстоянии $h = 5$ м от границы раздела сред воздух-вода. Также рассматривается случай, когда петля расположена

в воде ($h = -5$ м).

Выделим внутри области с условием «большого бака» и области, на границе которой задан PML-слой, подобласть Ω' размером $1000 \times 1000 \times 1000$ м³. Для этой подобласти в данном исследовании будем оценивать разность в норме \mathbb{L}^2 между действительными компонентами \mathbf{E}_y векторов решений $\mathbf{E}_y^{\text{бак}}$ и $\mathbf{E}_y^{\text{PML}}$, полученных с применением «большого бака» и PML-слоя соответственно.

3.3.2. Варьирование коэффициентов растяжения

Зафиксируем $\delta_k = 100$ м, $l_k = 600$ м, $m = 3$, $h = 5$ м и будем варьировать коэффициент комплексного растяжения координат χ . Размер получаемой СЛАУ для «большого бака» – 653814, с PML-слоем – 616180. Результаты приведены в таблице 4. Для случая $h = -5$ м размер получаемой СЛАУ для «большого бака» – 652396, с PML-слоем – 614504. Результаты приведены в таблице 5.

Таблица 4 — варьирование коэффициентов растяжения при $h = 5$ м

$\text{Re}(\chi)$ в Ω_1	$\text{Im}(\chi)$ в Ω_1	$\text{Re}(\chi)$ в Ω_2	$\text{Im}(\chi)$ в Ω_2	$\text{Re}(\chi)$ в Ω_3	$\text{Im}(\chi)$ в Ω_3	$\frac{\ \text{Re}(\mathbf{E}_y^{\text{бак}}) - \text{Re}(\mathbf{E}_y^{\text{PML}})\ }{\ \text{Re}(\mathbf{E}_y^{\text{бак}})\ }$	Время, бак	Время, PML
3	0	1	5	3	1	0.106636	650	592
3	1	0	6	2	1	0.0925		599
4	0	1	5	3	1	0.0947		731
4	1	0	6	2	1	0.0910		591

Таблица 5 — варьирование коэффициентов растяжения при $h = -5$ м

$\text{Re}(\chi)$ в Ω_1	$\text{Im}(\chi)$ в Ω_1	$\text{Re}(\chi)$ в Ω_2	$\text{Im}(\chi)$ в Ω_2	$\text{Re}(\chi)$ в Ω_3	$\text{Im}(\chi)$ в Ω_3	$\frac{\ \text{Re}(\mathbf{E}_y^{\text{бак}}) - \text{Re}(\mathbf{E}_y^{\text{PML}})\ }{\ \text{Re}(\mathbf{E}_y^{\text{бак}})\ }$	Время, бак	Время, PML
4	0	1	5	3	1	0.0929047	309	344
4	0	1	6	3	1	0.0870		294
4	0	1	6	3	2	0.0809		253
4	1	1	6	3	2	0.0658		306

3.3.3. Варьирование толщины PML-слоя

Зафиксируем $\chi_{\Omega_1} = (4, 1)$, $\chi_{\Omega_2} = (0, 6)$, $\chi_{\Omega_3} = (2, 1)$, $l_k = 600$ м, $m = 3$, $h = 5$ м и будем варьировать толщину PML-слоя δ_k . Результаты приведены в таблице 6. Для случая $h = -5$ м выберем $\chi_{\Omega_1} = (4, 0)$, $\chi_{\Omega_2} = (1, 6)$, $\chi_{\Omega_3} = (3, 2)$. Результаты приведены в таблице 7.

Таблица 6 — варьирование толщины PML-слоя при $h = 5$ м

δ_k	$\frac{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}}) - \operatorname{Re}(\mathbf{E}_y^{\text{PML}})\ }{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}})\ }$	Время, бак	Время, PML	Размер СЛАУ, бак	Размер СЛАУ, PML
80	0.1199	673	1289	659858	618128
100	0.0910	650	591	653814	616180
120	0.0784	609	1142	654354	617324

Таблица 7 — варьирование толщины PML-слоя при $h = -5$ м

δ_k	$\frac{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}}) - \operatorname{Re}(\mathbf{E}_y^{\text{PML}})\ }{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}})\ }$	Время, бак	Время, PML	Размер СЛАУ, бак	Размер СЛАУ, PML
80	0.1201	359	297	652312	614822
100	0.0809	309	253	652396	614504
120	0.0623	250	859	652422	615394

3.3.4. Варьирование размера области, на границе которой вводится PML-слой

Зафиксируем $\chi_{\Omega_1} = (4, 1)$, $\chi_{\Omega_2} = (0, 6)$, $\chi_{\Omega_3} = (2, 1)$, $\delta_k = 100$ м, $m = 3$, $h = 5$ м и будем варьировать l_k — размер области, на границе которой вводится PML-слой. Результаты приведены в таблице 8. Для случая $h = -5$ м выберем $\chi_{\Omega_1} = (4, 0)$, $\chi_{\Omega_2} = (1, 6)$, $\chi_{\Omega_3} = (3, 2)$. Результаты приведены в таблице 9.

Таблица 8 — варьирование размера области, на границе которой вводится PML-слой, при $h = 5$ м

l_k	$\frac{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}}) - \operatorname{Re}(\mathbf{E}_y^{\text{PML}})\ }{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}})\ }$	Время, бак	Время, PML	Размер СЛАУ, бак	Размер СЛАУ, PML
500	0.187456	628	587	659130	621390
600	0.0909998	650	591	652396	614504
800	0.0440642	718	658	794310	744856

Таблица 9 — варьирование размера области, на границе которой вводится PML-слой, при $h = -5$ м

l_k	$\frac{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}}) - \operatorname{Re}(\mathbf{E}_y^{\text{PML}})\ }{\ \operatorname{Re}(\mathbf{E}_y^{\text{бак}})\ }$	Время, бак	Время, PML	Размер СЛАУ, бак	Размер СЛАУ, PML
500	0.1751746	317	238	659814	621390
600	0.0809429	309	253	652396	614504
800	0.0348019	357	329	793272	743780

3.3.5. Проверка выполнения условий на контактных границах

Проверим, что в случае независимого варьирования коэффициентов комплексного растяжения координат χ , на границе двух соседних PML-слоёв с различными характеристиками не нарушаются условия на контактных границах (3)-(4). Для этого рассмотрим напряжённость электрического поля \mathbf{E} вдоль линии $x = 650$, $y = 0$, $z = [-0.005, 0.005]$ (середина PML-слоя по x -направлению):

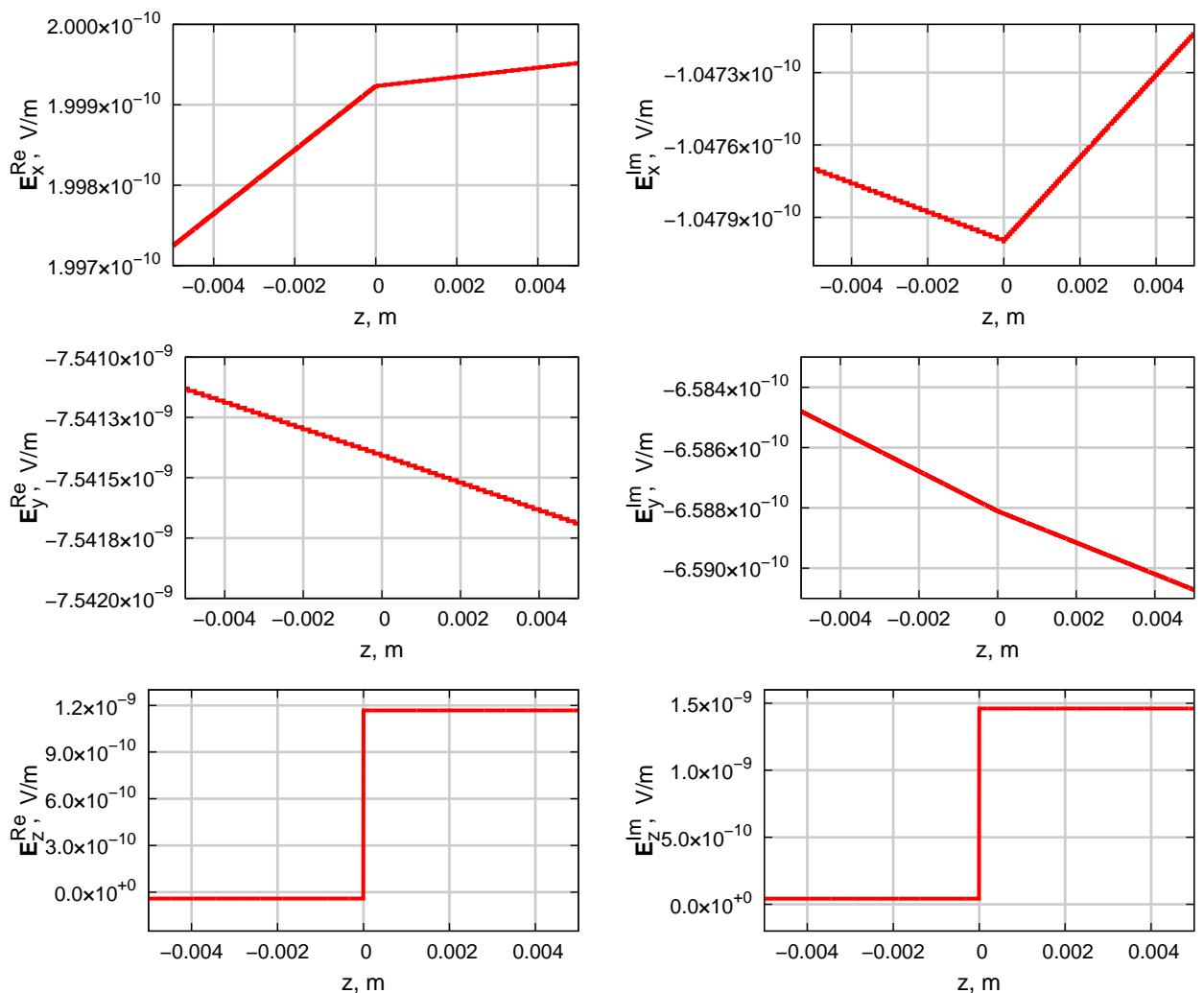


Рисунок 16 — графики компонент электрического поля на контактных границах

Как видно из графиков на рисунке 16, разрывна только нормальная компонента \mathbf{E}_z , следовательно, условия (3)-(4) выполнены.

3.3.6. Графическое представление результатов

Рассмотрим результаты, полученные при параметрах $h = 5$ м, $\chi_{\Omega_1} = (4, 0)$, $\chi_{\Omega_2} = (1, 6)$, $\chi_{\Omega_2} = (3, 2)$, $m = 3$, $l_k = 600$ м и $\delta_k = 100$ м. На рисунках 17 и 18 показаны картины электрического поля в сечении плоскостью $z = -10$ м. На рисунках 17а и 18а представлено решение с PML-слоем; 17б и 18б – решение с «большим баком».

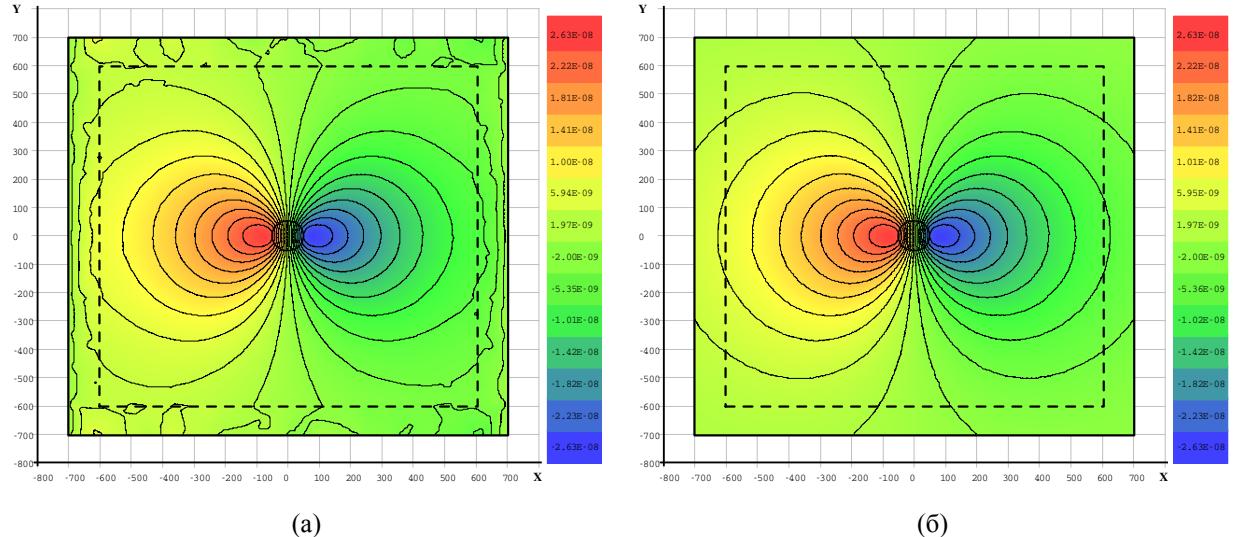


Рисунок 17 — $\text{Re}(\mathbf{E}_y)$

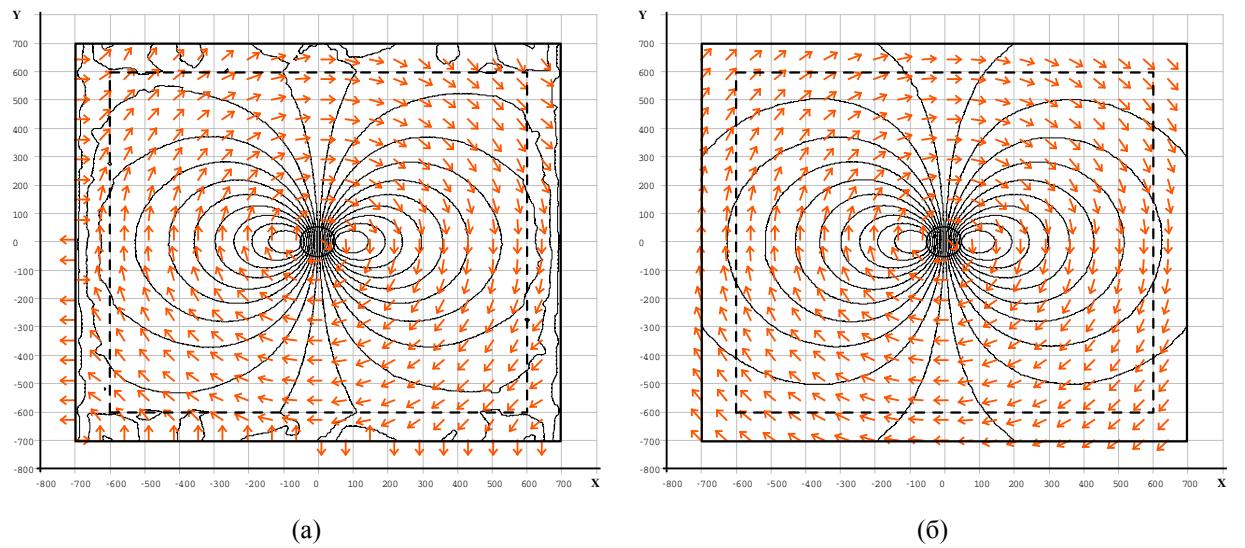


Рисунок 18 — изолинии $\text{Re}(\mathbf{E}_y)$ и векторы $(\text{Re}(\mathbf{E}_x), \text{Re}(\mathbf{E}_y))^T$

На рисунке 19 показаны картины электрического поля в сечении плоскостью $y = 0$. На рисунках 19а и 19б представлено решение с PML-слоем и решение с «большим баком» соответственно.

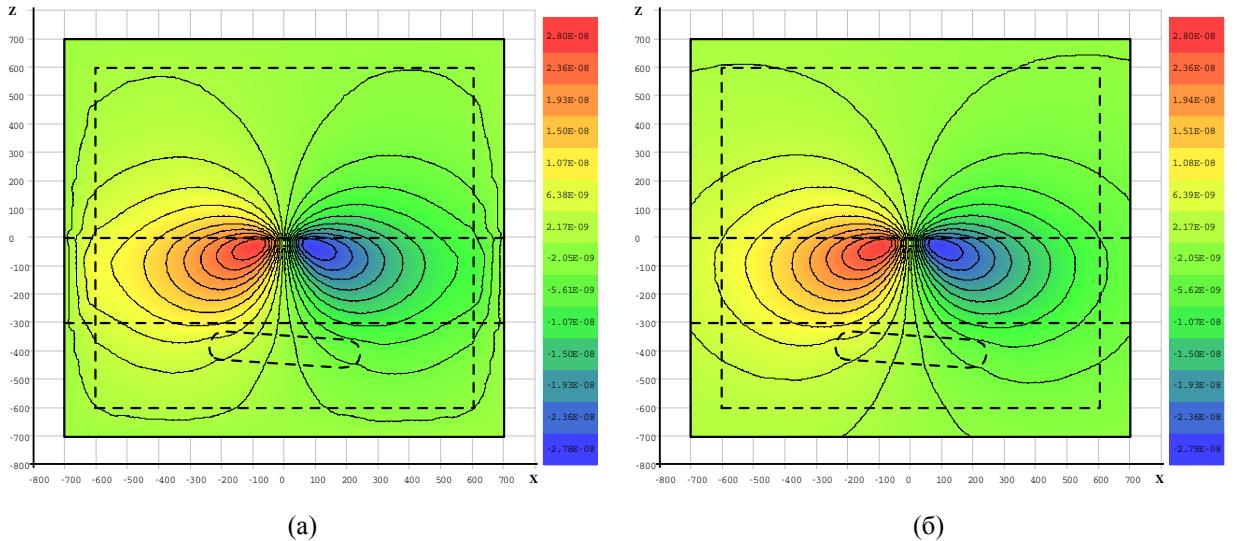


Рисунок 19 — $\text{Re}(\mathbf{E}_y)$

3.3.7. Анализ целесообразности применения PML-слоя

Наибольшее влияние на точность получаемого решения при введении PML-слоя оказывают коэффициент комплексного растяжения координат χ и l – размер области, на границе которой вводится PML-слой. При увеличении размера внутренней области ожидаемо увеличивается размер СЛАУ, поэтому этот параметр при решении реальных задач следует не варьировать, а выбрать как некоторое ограничение. Таким образом, в первую очередь стоит проводить поиск оптимального значения именно для коэффициента растяжения χ . Выбор толщины PML-слоя влияет на точность решения незначительно, поэтому достаточно лишь убедиться, что она выбрана в подходящих для задачи пределах – не слишком большой и не слишком маленькой.

В рассмотренном виде применение PML-слоя для сокращения области моделирования задач низкочастотной морской геоэлектрики незначительно уменьшает размерность СЛАУ и, в большинстве случаев, не приводит к существенному сокращению времени решения задачи. Одной из причин этого может быть то, что основное растяжение приходится на вещественные компоненты координат. Это приводит к значительной «вытянутости» тетраэдров внутри PML-слоя и, как следствие, сильному ухудшению свойств матрицы СЛАУ и увеличению времени решения. Параллелепипедальные конечные элементы

лишены подобного недостатка, поэтому для них можно проводить комплексное растяжение в гораздо больших диапазонах. Однако, такие элементы не подходят для аппроксимации сколь-либо сложных областей. Для использования в одной сетке и тетраэдральных, и параллелепипедальных конечных элементов можно применять специальные переходные элементы [33, 34], либо воспользоваться неконформными методами [35–39].

3.4. Задача, приближенная к реальной

В этом исследовании рассмотрим поведение электрического поля при различном расположении источника этого поля относительно объекта, скрытого в грунте. Различное положение источника обеспечивается за счёт «протаскивания» его на некотором расстоянии от морского дна.

В этом исследовании будем пользоваться базисными функциями второго полного порядка.

3.4.1. Описание расчётной области

Расчётная область показана на рисунке 20, где Ω_1 – воздух ($\sigma = 10^{-6}$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_2 – морская вода ($\sigma = 3.3$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_3 – грунт ($\sigma = 0.2$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); Ω_4 – углеводороды ($\sigma = 10^{-2}$ См/м, $\mu = \mu_0$, $\varepsilon = \varepsilon_0$); L_1 , L_2 и L_3 – размеры области моделирования по осям x , y и z соответственно; $L_1 = L_2 = L_3 = 6000$ м; $h_1 = 600$ м – толщина Ω_2 ; $l_1 = 400$ м, $h_3 = 75$ м, $h_2 = 120$ м – длина, толщина и глубина объекта Ω_4 соответственно.

В качестве источника электрического поля выступает петля диаметром $d = 100$ м с током частотой 1 Гц, расположенная в воде на расстоянии $h = 590$ м от границы раздела сред воздух-вода (в 10 метрах над поверхностью грунта). Объект Ω_4 представляет собой скруглённый прямоугольный параллелепипед с двумя равными сторонами, наклонённый под углом 5° .

В этой области рассмотрим электрическое поле при различных смещениях объекта относительно источника ($l_2 = -300, -200, -100$ и 0 м), а также сравним с электрическим полем в случае проводящего объекта ($\sigma = 10^2$ См/м).

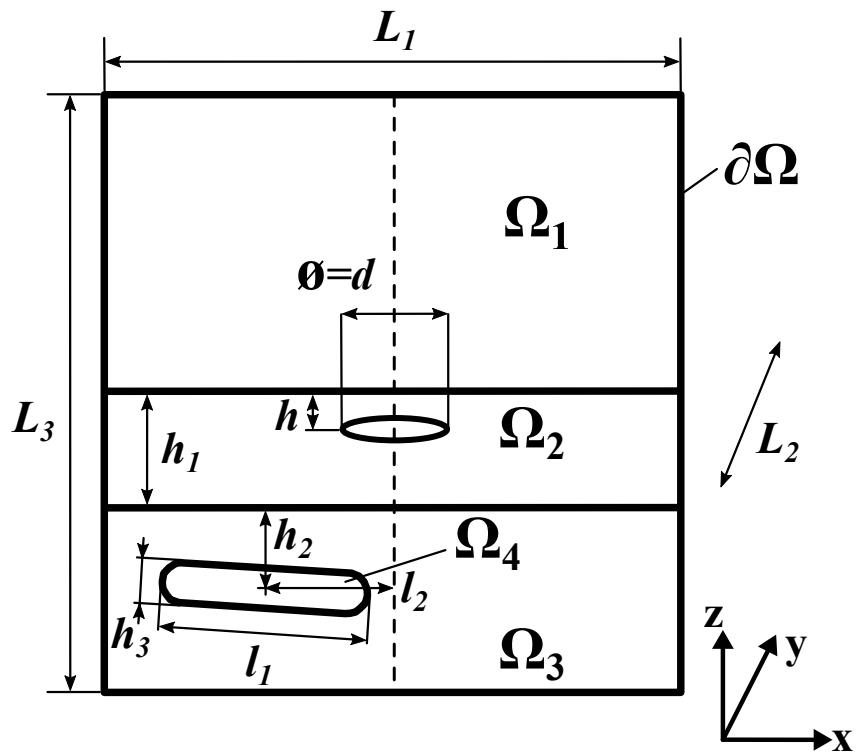


Рисунок 20 — расчётная область

3.4.2. Конечноэлементная сетка

Фрагмент $x \in [-600, 0], y \in [-600, 600] z \in [-1000, 600]$ одной из конечно-элементных сеток, использованных для проведения исследования, представлен на рисунке 21.

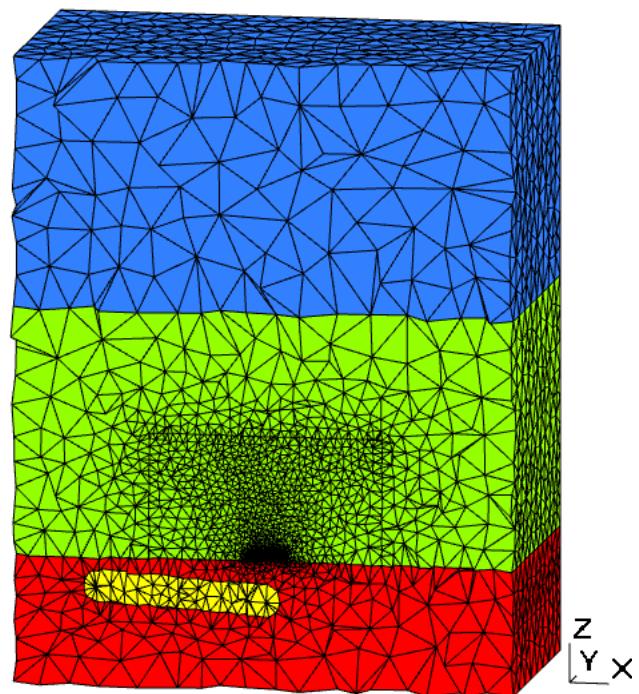


Рисунок 21 — фрагмент конечноэлементной сетки

3.4.3. Результаты вычислительного эксперимента

Рассмотрим результаты для проводящего и непроводящего объекта при различных его смещениях относительно источника ($l_2 = -300, -200, -100$ и 0 м). На рисунках 22-23 показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = 0$ в сечении $y = 0$, на рисунках 22а и 23а – для объекта с $\sigma = 10^{-2}$ См/м, на рисунках 22б и 23б – для объекта с $\sigma = 10^2$ См/м.

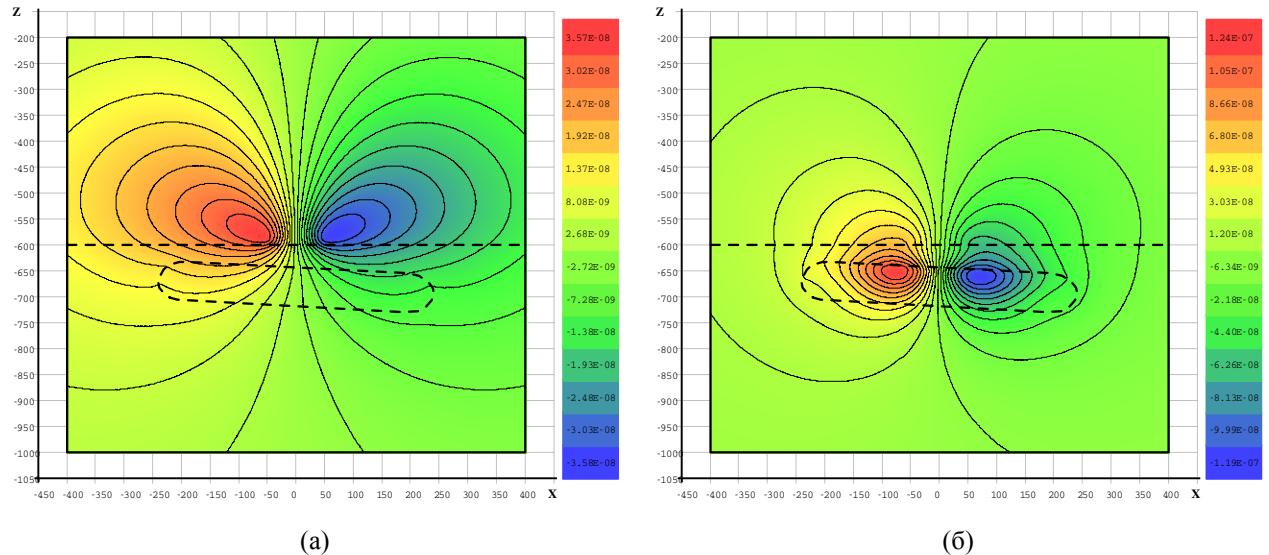


Рисунок 22 — $\text{Re}(\mathbf{E}_y)$ при $l_2 = 0$

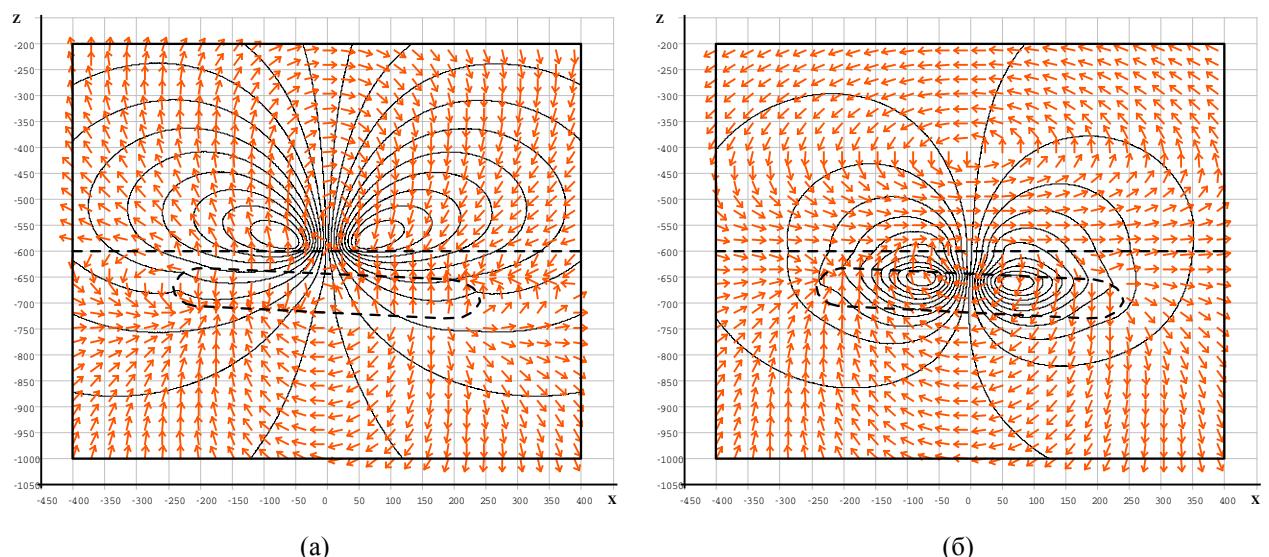


Рисунок 23 — векторы $(\text{Re}(\mathbf{E}_x), \text{Re}(\mathbf{E}_z))^T$, изолинии $\text{Re}(\mathbf{E}_y)$ при $l_2 = 0$

На рисунках 24а и 24б показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = 0$ в сечении $z = -601$ м для объекта с

$\sigma = 10^{-2}$ См/м и для объекта с $\sigma = 10^2$ См/м соответственно.

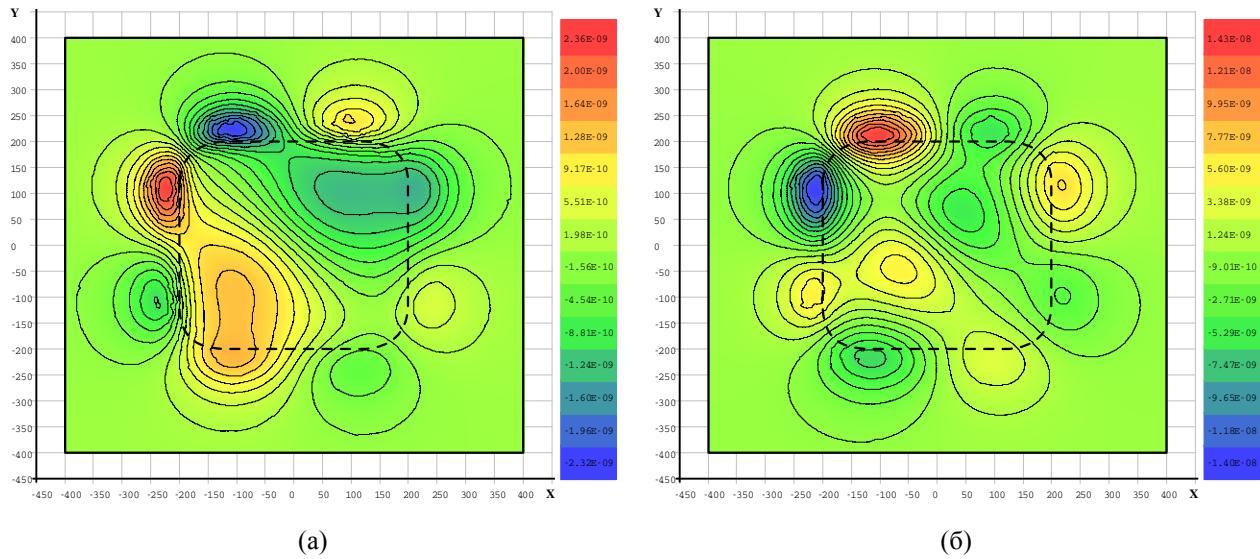


Рисунок 24 — $\text{Re}(\mathbf{E}_z)$ при $l_2 = 0$

На рисунках 25-26 показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = -100$ в сечении $y = 0$, на рисунках 25а и 26а – для объекта с $\sigma = 10^{-2}$ См/м, на рисунках 25б и 26б – для объекта с $\sigma = 10^2$ См/м.

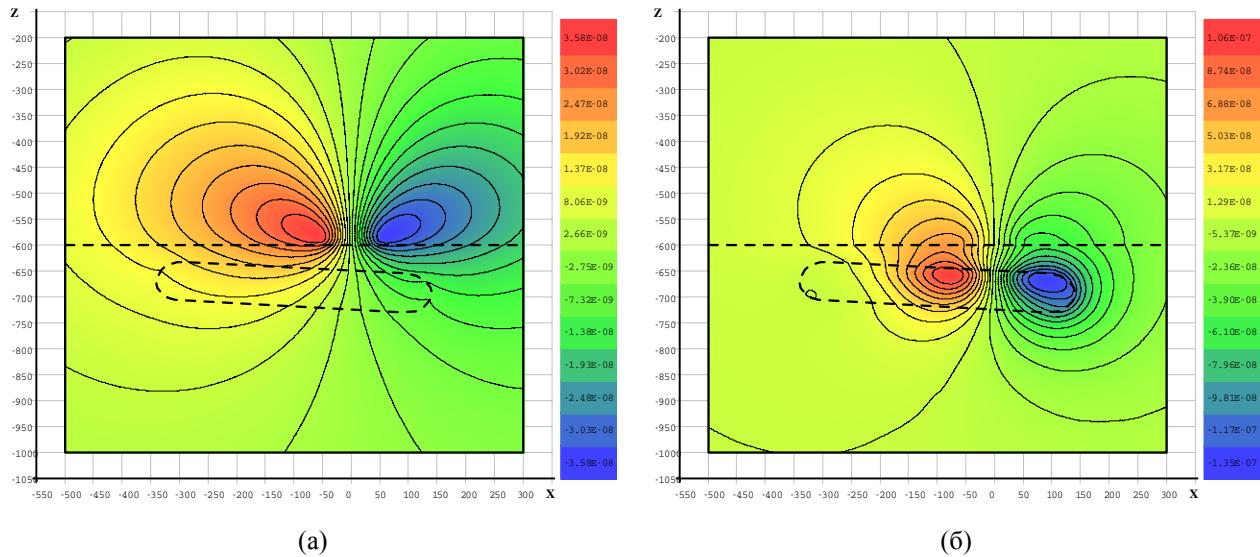


Рисунок 25 — $\text{Re}(\mathbf{E}_y)$ при $l_2 = -100$

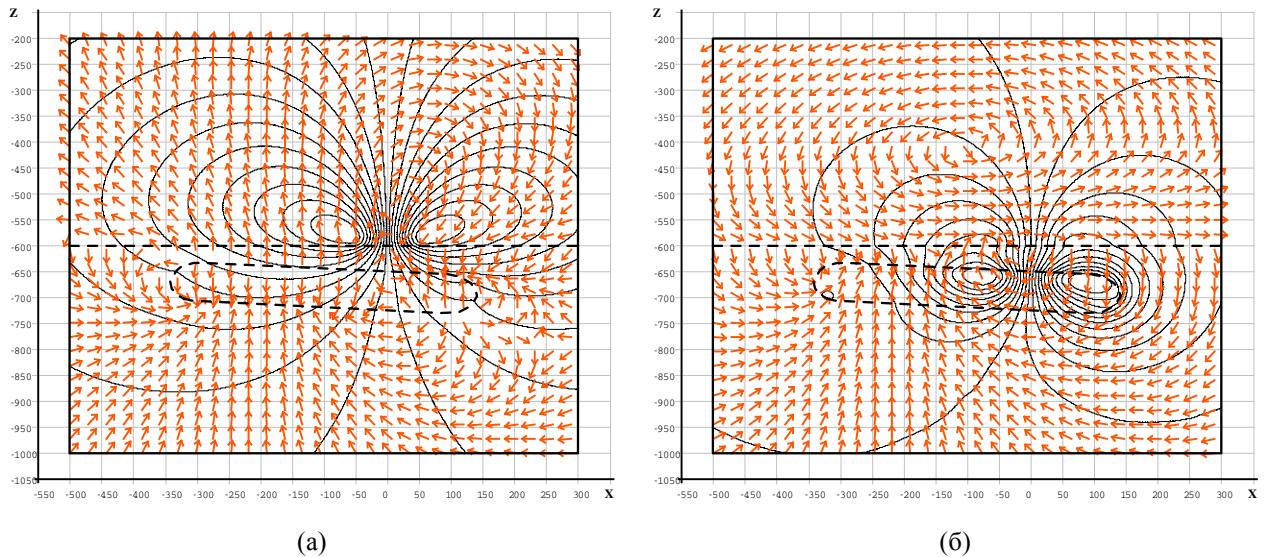


Рисунок 26 — векторы $(\text{Re}(\mathbf{E}_x), \text{Re} \mathbf{E}_z)^T$, изолинии $\text{Re}(\mathbf{E}_y)$ при $l_2 = -100$

На рисунках 27а и 27б показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = -100$ в сечении $z = -601$ м для объекта с $\sigma = 10^{-2}$ См/м и для объекта с $\sigma = 10^2$ См/м соответственно.

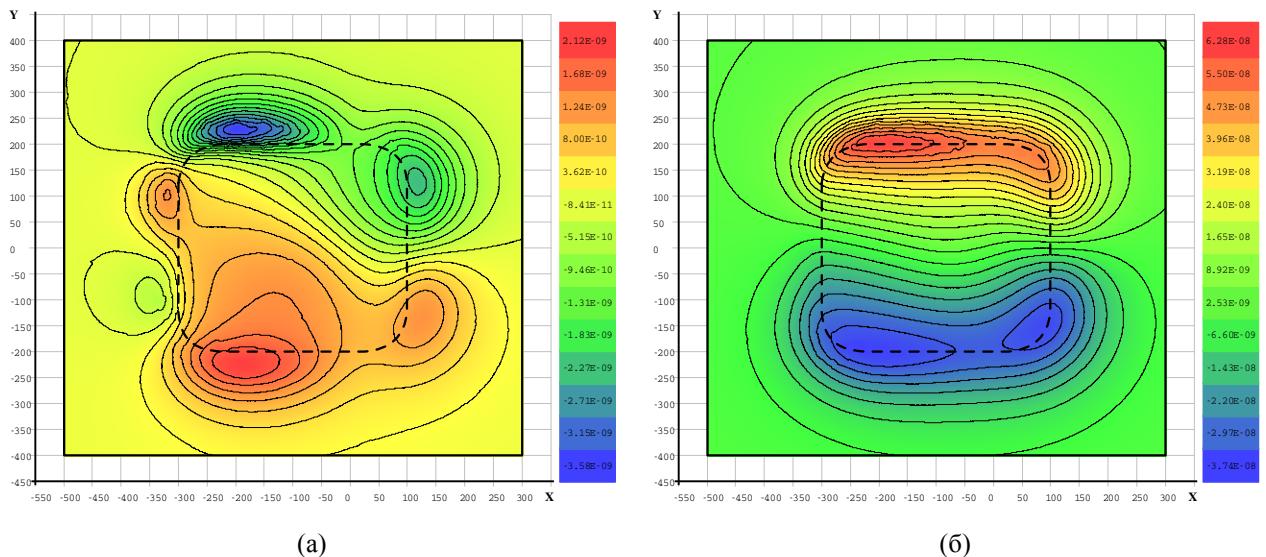


Рисунок 27 — $\text{Re}(\mathbf{E}_z)$ при $l_2 = -100$

На рисунках 28-29 показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = -200$ в сечении $y = 0$, на рисунках 28а и 29а — для объекта с $\sigma = 10^{-2}$ См/м, на рисунках 28б и 29б — для объекта с $\sigma = 10^2$ См/м.

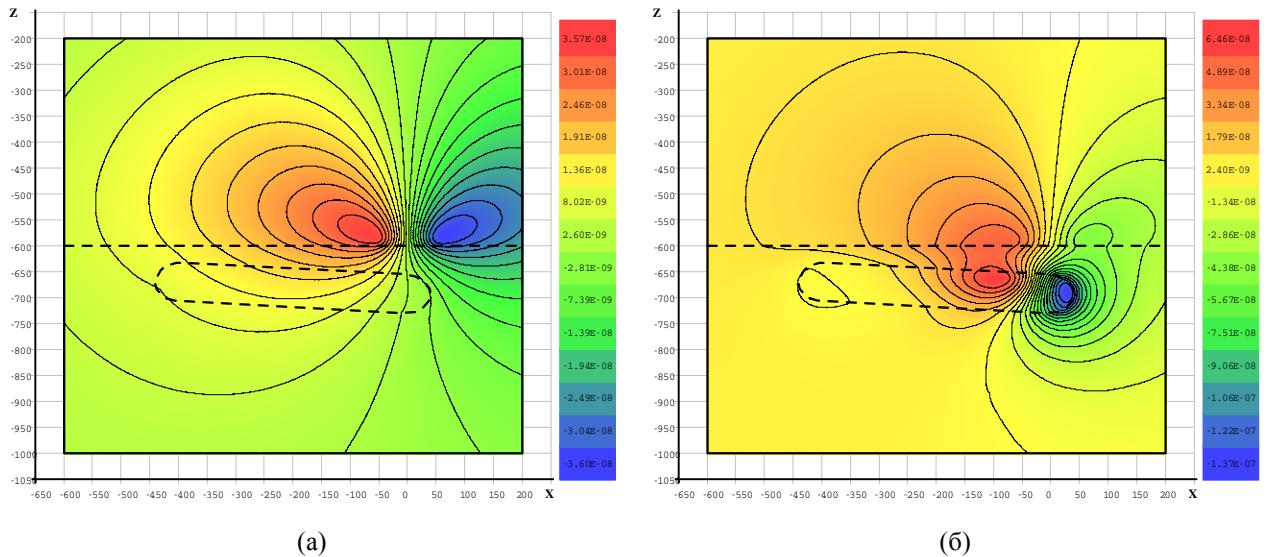


Рисунок 28 — $\text{Re}(\mathbf{E}_y)$ при $l_2 = -200$

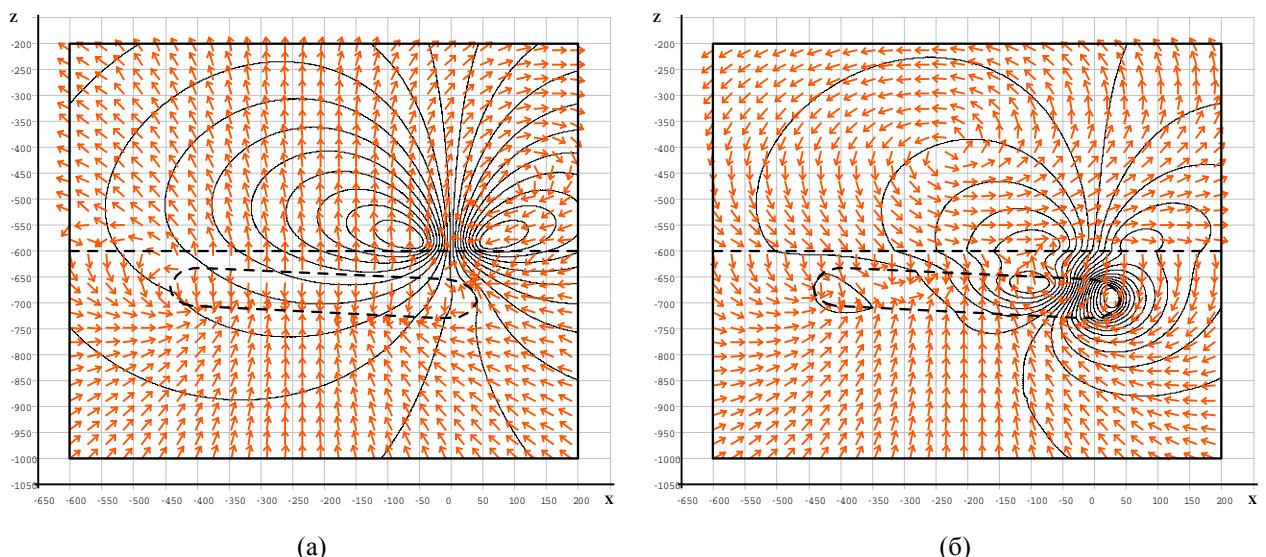


Рисунок 29 — векторы $(\text{Re}(\mathbf{E}_x), \text{Re} \mathbf{E}_z)^T$, изолинии $\text{Re}(\mathbf{E}_y)$ при $l_2 = -200$

На рисунках 30а и 30б показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = -200$ в сечении $z = -601$ м для объекта с $\sigma = 10^{-2}$ См/м и для объекта с $\sigma = 10^2$ См/м соответственно.

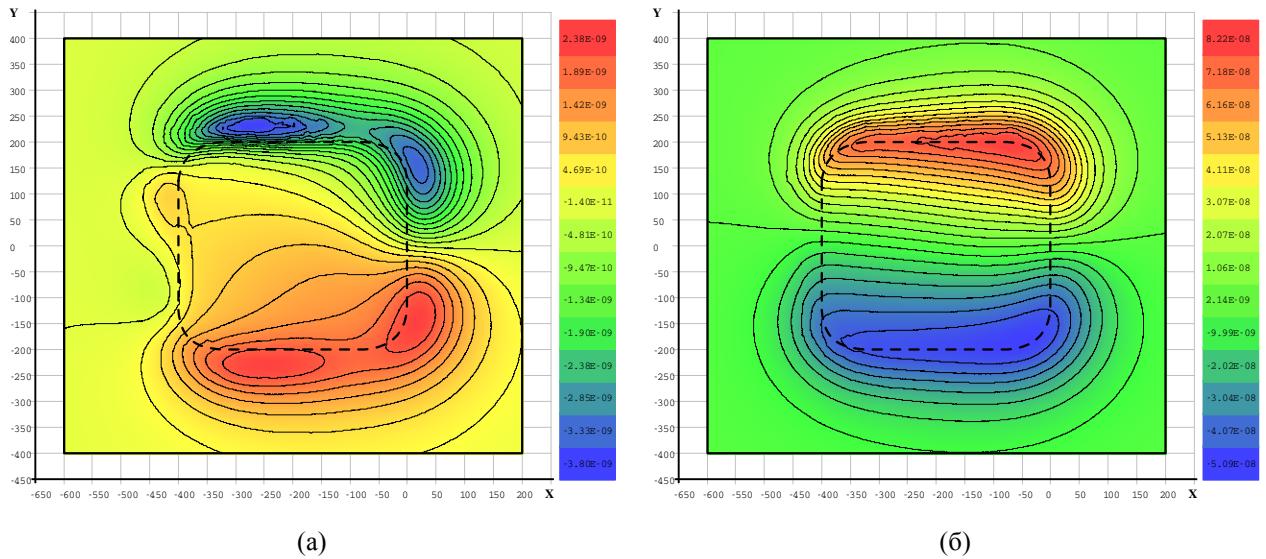


Рисунок 30 — $\text{Re}(\mathbf{E}_z)$ при $l_2 = -200$

На рисунках 31-32 показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = -300$ в сечении $y = 0$, на рисунках 31а и 32а – для объекта с $\sigma = 10^{-2}$ См/м, на рисунках 31б и 32б – для объекта с $\sigma = 10^2$ См/м.

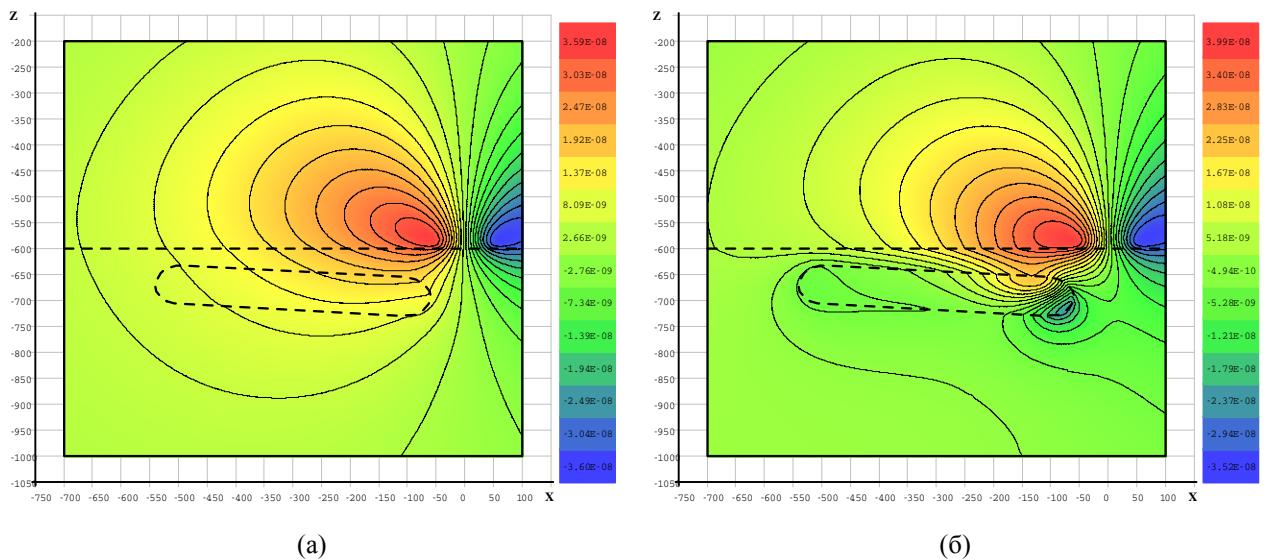


Рисунок 31 — $\text{Re}(\mathbf{E}_y)$ при $l_2 = -300$

На рисунках 33а и 33б показано графическое представление напряжённости электрического поля \mathbf{E} при $l_2 = -300$ в сечении $z = -601$ м для объекта с $\sigma = 10^{-2}$ См/м и для объекта с $\sigma = 10^2$ См/м соответственно.

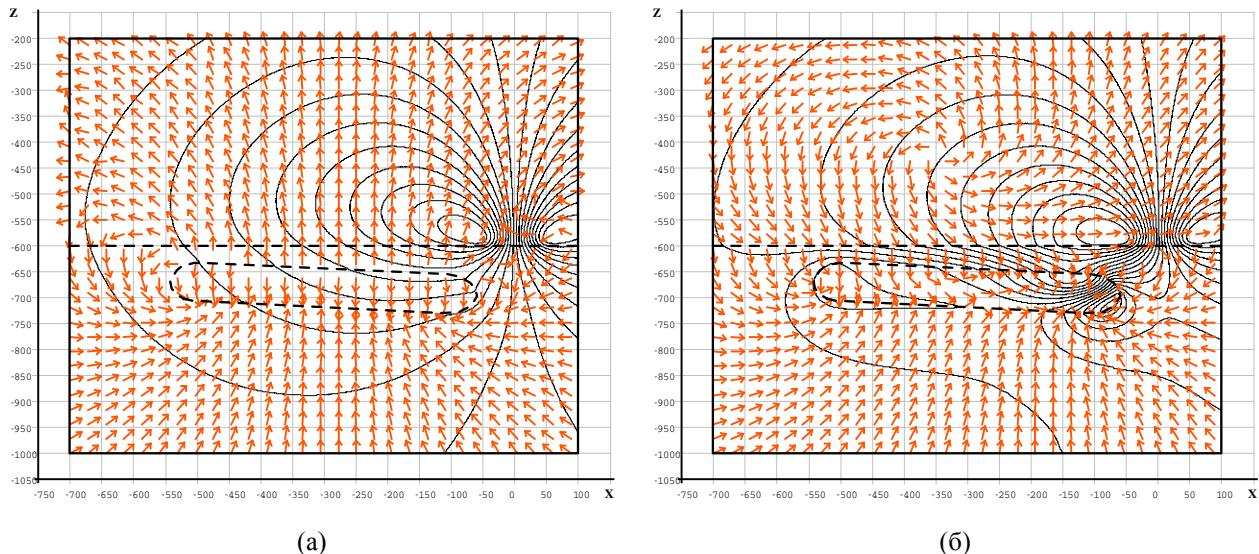


Рисунок 32 — векторы $(\operatorname{Re}(\mathbf{E}_x), \operatorname{Re}(\mathbf{E}_z))^T$, изолинии $\operatorname{Re}(\mathbf{E}_y)$ при $l_2 = -300$

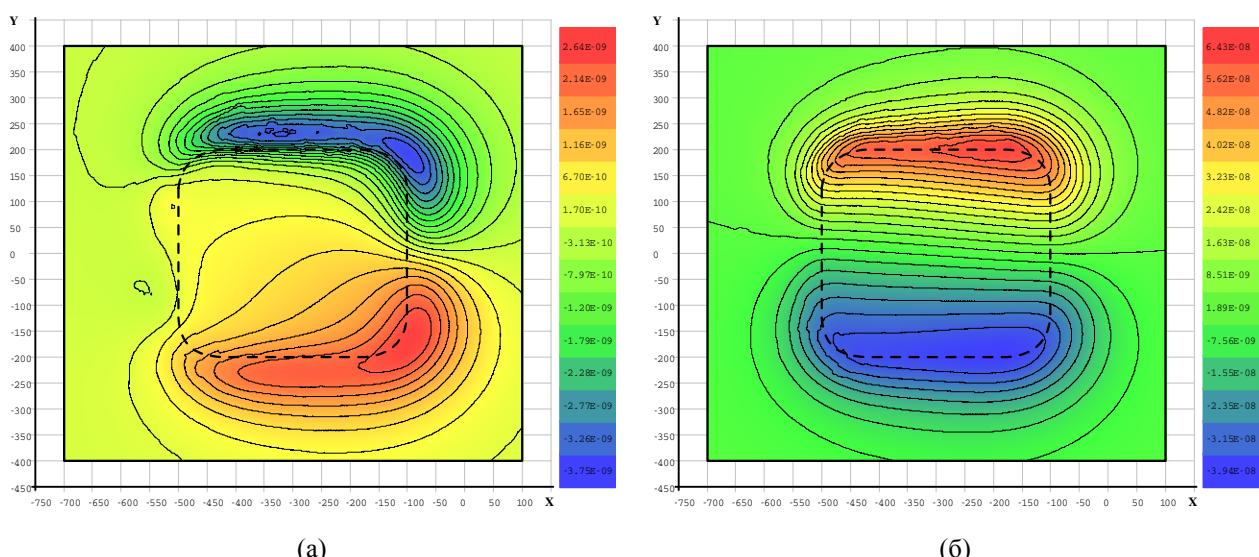


Рисунок 33 — $\operatorname{Re}(\mathbf{E}_z)$ при $l_2 = -300$

Результаты моделирования показывают, что проводящий объект хорошо «виден» и на некотором расстоянии от морского дна, а непроводящий – только вблизи поверхности или при некотором заглублении приёмника в грунт, причём наибольший отклик на источник электромагнитного возмущения для непроводящего объекта наблюдается в том случае, когда этот источник расположен с некоторым смещением от центра симметрии объекта.

4. Описание программного комплекса

Разработанный программный комплекс ориентирован на решение задач, представимых в виде уравнения Гельмгольца, векторным методом конечных элементов. Комплекс написан на языке C++ [40] и может использовать стороннюю библиотеку Intel® MKL [41] и распараллеливание с помощью OpenMP [42], однако это не является обязательным – достаточно любого компилятора C++, поддерживающего стандарт C++03 и новее. Программный комплекс не имеет жёсткой привязки к архитектуре процессора и операционной системе.

Этапы пре- и постпроцессинга выполняются сторонними программами. Построение конечноэлементной сетки выполняется с помощью пакета Gmsh [43]. Регулярные тетраэдralьные сетки могут быть построены с помощью пакета TetMeshGen [44]. Результаты работы программного комплекса выводятся в формате пакета Tecplot [45]. В двумерном и одномерных случаях этот формат, помимо пакета Tecplot, может быть обработан в пакетах FEM Draw [46] и gnuplot [47] соответственно.

Входными данными для программного комплекса являются аргументы командной строки, в которых задаются режим работы и расположение основных конфигурационных файлов; сами конфигурационные файлы, в которых задаются настройки программного комплекса, описания физических областей и PML-слоя; файл сетки в формате MSH ASCII file format [48]. Выходными данными для программного комплекса являются файлы в формате пакета Tecplot, а также файл с результатами решения СЛАУ. Параметры выходных данных задаются в основном конфигурационном файле.

4.1. Формат конфигурационных файлов

Конфигурационные файлы представляют собой файлы настроек в формате `ini` [49]. Данный формат имеет простую структуру, поэтому легко обрабатывается программно и имеет понятный для чтения и редактирования человеком вид. Рассмотрим пример синтаксиса файлов подобного типа:

```

; Commentary
[Section]
parameter1 = value1
parameter2 = value2

[Section.Subsection]
parameter3 = value3

```

В рассматриваемом примере `Commentary` – комментарий; `Section` – название секции, некоторая заранее определённая строка; `Subsection` – название подсекции, используется как уникальный идентификатор подсекции, отличающий её от других подсекций той же секции; `parameterX` – название параметра, строка из множества допустимых параметров для данной секции; `valueX` – значение соответствующего параметра, тип значения определяется соответствующим параметром.

Подробные примеры конфигурационных файлов с пояснениями приведены в приложении А.

4.2. Аргументы командной строки и переменные окружения

В зависимости от режима работы программного комплекса список аргументов командной строки может оказаться различным.

Режим решения. Наиболее часто используемый режим, в процессе работы которого производится чтение исходных данных и решение задачи, после чего результат решения обрабатывается постпроцессором. Опции командной строки, доступные в таком режиме:

```
programname [-nosolve|-continue] [-nopost] [config.ini]
```

где `programname` – имя программы; `-nosolve` – ключ, по которому не будет проводиться процедура решения СЛАУ, вместо этого веса решения будут восстановлены из файла, указанного в основном конфигурационном файле; `-continue` – ключ, по которому начальные веса будут взяты из файла, указанного в основном конфигурационном файле; `-nopost` – ключ, отключающий постпроцессор; `config.ini` – имя основного конфигурационного файла (по умолчанию – «`config.ini`»).

Режим сравнения. Режим, в котором последовательное решение двух задач, а затем производится сравнение полученных решений в заданной подоб-

ласти Ω' в норме $\mathbb{L}^2(\Omega')$. Опции командной строки, доступные в таком режиме:

```
programname -diff [-nosolve|-continue] [-nopost] master.ini  
[-nosolve|-continue] [-nopost] slave.ini diff.ini
```

где `master.ini` – имя конфигурационного файла для первичной задачи, сетка которой будет использоваться для численного интегрирования; `slave.ini` – имя конфигурационного файла для вторичной задачи; `diff.ini` – имя конфигурационного файла сравнения решений; остальные параметры идентичны таковым в режиме решения.

Режим сравнения на идентичных сетках. Режим, аналогичный предыдущему, с той лишь разницей, что сетки обеих задач должны быть геометрически идентичными в подобласти для сравнения Ω' . За счёт такого ограничения сравнение решений выполняется значительно быстрее. Опции командной строки, доступные в таком режиме:

```
programname -diff_simple [-nosolve|-continue] [-nopost]  
master.ini [-nosolve|-continue] [-nopost] slave.ini diff.ini
```

все параметры идентичны параметрам в режиме сравнения.

Для тонкой настройки работы программы доступны также некоторые переменные окружения:

- `NO_PROGRESS` – отключает вывод прогресса текущей операции (по умолчанию прогресс показывается);
- `OMP_NUM_THREADS=N` – устанавливает количество доступных потоков в подпрограммах, использующих OpenMP (по умолчанию – 1);
- `MKL_NUM_THREADS=N` – устанавливает количество доступных потоков в подпрограммах, использующих Intel® MKL (по умолчанию – 1);
- `GMRES_M=N` – устанавливает глубину метода для решателей на базе GMRES (по умолчанию – 5).

Заключение

В работе были реализованы алгоритмы на базе векторного метода конечных элементов. Эти алгоритмы были положены в основу программного комплекса, который позволяет моделировать электромагнитные поля в областях разнообразной структуры. С помощью этого программного комплекса были решены модельные задачи морской геоэлектрики на низких частотах, проведены исследования возможности сокращения области моделирования без внесения дополнительных погрешностей.

На основании исследований были сделаны выводы, что расчёты, в которых в область моделирования не включается воздух, допустимы только при расположении источника электромагнитного поля на большой глубине, иначе, из-за неправильного учёта физических процессов, происходящих в воздухе, полученное решение будет неверным.

Применение PML-слоя позволило получить достаточно точные решения, однако его применение не привело к резкому уменьшению размерности систем уравнений и, как следствие, к уменьшению времени решения. Было выдвинуто предположение, что увеличить эффективность применения PML-слоя можно с помощью применения неконформных методов, в которых конечноэлементная сетка может содержать геометрические носители разного типа: тетраэдры и параллелепипеды.

Расчёты на модельной задаче с проводящим и непроводящим объектами показали, что проводящий объект хорошо «виден» на некотором расстоянии от морского дна, а непроводящий – только вблизи дна или при небольшом заглублении приёмника в грунт. Наибольший отклик на источник электромагнитного возмущения для непроводящего объекта наблюдался в том случае, когда источник располагался со смещением от центра симметрии объекта.

Список литературы

1. Шурина, Э.П. Морская геоэлектрика – задачи и перспективы / Э.П. Шурина, М.И. Эпов, А.В. Мариенко // Тез. докл. всеросс. науч.-техн. конф. «Научное и техническое обеспечение исследования и освоения шельфа Северного Ледовитого океана». – 2010. – 9-13 августа. – С. 7-12.
2. Gabrielsen, P.T. 3D CSEM for Hydrocarbon Exploration in the Barents Sea / P.T. Gabrielsen, D.V. Shantsev, S. Fanavoll // 5th Saint Petersburg International Conference & Exhibition – Geosciences: Making the most of the Earth's resources. – 2012. – 2-5 April. – Pp. 1-5.
3. Anderson, C. An integrated approach to marine electromagnetic surveying using a towed streamer and source / C. Anderson, J. Mattsson // First Break. – May 2010. – Vol. 28, Iss. 5. – Pp. 71-75.
4. Жигалов, П.С. Влияние слоя воздуха в задачах морской геоэлектрики / П.С. Жигалов, Б.В. Рак // Наука. Технологии. Инновации. : сб. науч. трудов – Новосибирск : Изд-во НГТУ, 2015 – 01-05 декабря – Часть 2 – С. 56-57.
5. Mur G. Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations // Electromagnetic Compatibility, IEEE Transactions on. – 1981. – No. 4. – Pp. 377-382.
6. Berenger, J.P. A perfectly matched layer for the absorption of electromagnetic waves / J.P. Berenger // Journal of computation physics. – 1994 – No. 114. – Pp. 185-200.
7. Wiik, T. A Discontinuous Galerkin Method for Modelling Marine Controlled Source Electromagnetic Data / T. Wiik, M.V. De Hoop, B. Ursin // Proceedings of the Project Review, Geo-Mathematical Imaging Group, Purdue University, West Lafayette, IN. – 2013 – Vol. 1 – P. 75-102.
8. Баландин, М.Ю. Векторный метод конечных элементов : Учеб. пособие / М.Ю. Баландин, Э.П. Шурина. – Новосибирск : Изд-во НГТУ, 2001. – 69 с.
9. Monk P. Finite element methods for Maxwell's equations. / P. Monk – Oxford University Press, 2003.

10. Zhigalov, P. Analysis of the Source-Receiver Systems in Marine Geoelectric Problems / P. Zhigalov // Progress Through Innovation : Тез. городск. науч.-практ. конф. аспирантов и магистрантов. – Новосибирск : 2015. – 2 апреля. – С. 30-31.
11. Жигалов, П.С. Анализ систем источник-приемник в задачах морской геоэлектрики / П.С. Жигалов // МНСК-2015 : матер. 53 междунар. научн. студенч. конф. – Новосибирск : 2015 – 11-17 апреля – Математика – С. 216.
12. Шурина, Э.П. Математическое моделирование электромагнитного поля для задачи морской геоэлектрики / Э.П. Шурина, Б.В. Рак, П.С. Жигалов // Аналитические и численные методы моделирования естественно-научных и социальных проблем : сб. ст. X междунар. науч.-техн. конф. – Пенза : Изд-во ПГУ, 2015. – 28-30 октября. – С. 131-137.
13. Жигалов, П.С. Идеально согласованные слои / П.С. Жигалов, Б.В. Рак // Обработка информации и математическое моделирование. Росс. науч.-техн. конф. : матер. конф. – Новосибирск : 2015 – С. 106-114.
14. Эпов, М.И. Параллельные конечноэлементные вычислительные схемы в задачах геоэлектрики / М.И. Эпов, Э.П. Шурина, Д.А. Архипов // Вычислительные технологии. – 2013. – Том 18, №2. – С. 94-112.
15. Schwarzbach C. Stability of finite element solutions to Maxwell's equations in frequency domain. / C. Schwarzbach – 2009.
16. Hiptmair R. Multigrid method for Maxwell's equations / R. Hiptmair // SIAM Journal on Numerical Analysis. – 1998. – Vol. 36. – No. 1. – Pp. 204-225.
17. Nédélec J.C. Mixed finite elements in \mathbb{R}^3 / J.C. Nédélec // Numerische Mathematik. – 1980. – Vol. 35. – No. 3. – Pp. 315-341.
18. Nédélec J.C. A new family of mixed finite elements in \mathbb{R}^3 / J.C. Nédélec // Numerische Mathematik. – 1986. – Vol. 50. – No. 1. – Pp. 57-81.
19. Webb J.P. Edge elements and what they can do for you / J.P. Webb // Magnetics, IEEE Transactions on. – 1993. – Vol. 29. – No. 2. – Pp. 1460-1465.
20. Баландин М.Ю. Методы решения СЛАУ большой размерности: Учеб. пособие / М.Ю. Баландин, Э.П. Шурина. – Новосибирск : Изд-во НГТУ, 2000.

– 70 с.

21. Соловейчик, Ю.Г. Метод конечных элементов для решения скалярных и векторных задач : учеб. пособие / Ю.Г. Соловейчик, М.Э. Рояк, М.Г. Персова. – Новосибирск : Изд-во НГТУ, 2007. – 896 с.
22. Webb, J.P. Hierarchical Vector Basis Functions of Arbitrary Order for Triangular and Tetrahedral Finite Elements / J.P. Webb // IEEE transactions on antennas and propagation. – 1999. – Vol. 47. – Pp. 1244-1253.
23. Nечаев, О.В. Multilevel iterative solver for the edge fem solution of the 3D Maxwell equation / О.В. Нечаев, Е.П. Шурина, М.А. Ботчев // Computers and Mathematics with Applications. – 2008. – No. 55. – Pp. 2346-2362.
24. Михайлова Е.И., Шурина Э.П. Математическое моделирование высокочастотного электромагнитного поля в волноводных устройствах / Е.И. Михайлова, Э.П. Шурина // Вестник НГУ. Серия: Математика, механика, информатика. – 2013. – Т. 13. – №. 4. – С. 102-118.
25. Мысовских И.П. Интерполяционные кубатурные формулы. – Наука. Гл. ред. физ.-мат. лит., 1981.
26. Zhang L. et al. A set of symmetric quadrature rules on triangles and tetrahedra / L. Zhang, T. Cui, H. Liu // J. Comput. Math. – 2009. – Vol. 27. – No. 1. – Pp. 89-96.
27. Numerical Integration over the Tetrahedral Domain [Электронный ресурс]. – Режим доступа: <https://people.fh-landshut.de/~maurer/numeth/node148.html>.
28. Numerical Integration over the Triangular Domain [Электронный ресурс]. – Режим доступа: <https://people.fh-landshut.de/~maurer/numeth/node147.html>.
29. Шокин Ю.И. Современные многосеточные методы. – Часть II. Многоуровневые методы. Применение многомасштабных методов.: Учеб. пособие / Ю.И. Шокин, Э.П. Шурина, Н.Б. Иткина. – Новосибирск : Изд-во НГТУ, 2012. – 98 с.
30. Saad Y. Iterative methods for sparse linear systems. / Y. Saad – Siam, 2003.
31. Van der Vorst H.A. Iterative Krylov methods for large linear systems. / H.A. Van der Vorst – Cambridge University Press, 2003. – Vol. 13.

32. Barrett R. et al. Templates for the solution of linear systems: building blocks for iterative methods. / R. Barrett et al. – Siam, 1994. – Vol. 43.
33. Solin P., Segeth K., Dolezel I. Higher-order finite element methods. / P. Solin, K. Segeth, I. Dolezel – CRC Press, 2003. – 388 p.
34. Bergot M., Duruflé M. High-order optimal edge elements for pyramids, prisms and hexahedra / M. Bergot, M. Duruflé // Journal of Computational Physics. – 2013. – Vol. 232. – No. 1. – Pp. 189-213.
35. Dosopoulos S., Zhao B., Lee J.F. Non-conformal and parallel discontinuous Galerkin time domain method for Maxwell's equations: EM analysis of IC packages / S. Dosopoulos, B. Zhao, J.F. Lee // Journal of Computational Physics. – 2013. – Vol. 238. – Pp. 48-70.
36. Perugia I., Schötzau D. The hp -local discontinuous Galerkin method for low-frequency time-harmonic Maxwell equations / I. Perugia, D. Schötzau // Mathematics of Computation. – 2003. – Vol. 72. – No. 243. – Pp. 1179-1214.
37. Dosopoulos S., Lee J.F. Interconnect and lumped elements modeling in interior penalty discontinuous Galerkin time-domain methods / S. Dosopoulos, J.F. Lee // Journal of Computational Physics. – 2010. – Vol. 229. – No. 22. – Pp. 8521-8536.
38. Christophe A. et al. An Overlapping Nonmatching Grid Mortar Element Method for Maxwell's Equations / A. Christophe et al. // Magnetics, IEEE Transactions on. – 2014. – Vol. 50. – Iss. 2. – Pp. 409-412.
39. Gopalakrishnan J., Pasciak J.E. Multigrid for the mortar finite element method / J. Gopalakrishnan, J.E. Pasciak // SIAM Journal on Numerical Analysis. – 2000. – Vol. 37. – No. 3. – Pp. 1029-1052.
40. International organization for standardization, ISO/IEC 14882:2003: Programming languages: C++, 2nd ed. – Geneva, Switzerland, 2003 – 757 p.
41. Intel® Math Kernel Library (Intel® MKL) [Электронный ресурс]. – Режим доступа: <https://software.intel.com/en-us/intel-mkl>.
42. OpenMP Application Programming Interface: Version 4.5 November 2015 – OpenMP Architecture Review Board. – 2015. – 368 p.

43. Geuzaine C. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities / C. Geuzaine J.F. Remacle. // International Journal for Numerical Methods in Engineering – 2009 – Vol. 79 – Iss. 11 – Pp. 1309-1331
44. TetMeshGen: A regular tetrahedral finite element mesh generator [Электронный ресурс]. – Режим доступа: https://github.com/AlienCowEatCake/tet_mesh_gen.
45. Tecplot 10 User's Manual – Amtec Engineering, Inc., Bellevue, Washington., 2003.
46. FEM Draw: The software for visualization of scalar and vector fields [Электронный ресурс]. – Режим доступа: https://github.com/AlienCowEatCake/fem_draw.
47. Williams T. gnuplot 5.0: An Interactive Plotting Program / T. Williams, C. Kelley – 2016. – 254 p.
48. Gmsh Reference Manual [Электронный ресурс]. – Режим доступа: <http://gmsh.info/doc/texinfo/gmsh.html>.
49. .ini: Формат файла [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/.ini>.

Приложение А. Примеры конфигурационных файлов

Пример основного конфигурационного файла

```
; Общие параметры программного комплекса
[VFEM]
; Порядок базиса, 1 или 2
; [по умолчанию - 1]
basis_order = 1
; Тип базиса, 1 (неполный) или 2 (полный)
; [по умолчанию - 2]
basis_type = 2
; Порядок интегрирования на тетраэдральных элементах, от 2 до 14 включительно
; [по умолчанию - 8]
tet_integration_order = 4
; Порядок интегрирования на треугольных элементах, от 2 до 29 включительно
; [по умолчанию - 8]
tr_integration_order = 5
; Точность решения основной СЛАУ
; [по умолчанию - 1e-10]
eps_slae = 1e-10
; Точность решения СЛАУ неоднородных краевых условий первого типа
; [по умолчанию - 1e-14]
eps_slae_bound = 1e-15
; Использовать двухуровневый решатель или нет?
; [по умолчанию - no]
v_cycle_enabled = yes
; Точность нахождения начального приближения в двухуровневом решателе
; [по умолчанию - 0.01]
gamma_v_cycle_0 = 0.1
; Точность решения СЛАУ на полном пространстве в двухуровневом решателе
; [по умолчанию - 0.05]
gamma_v_cycle_full = 0.5
; Точность решения СЛАУ на пространстве ядра в двухуровневом решателе
; [по умолчанию - 0.01]
gamma_v_cycle_ker = 0.1
; Максимальное количество итераций для не двухуровневых решателей
; [по умолчанию - 15000]
max_iter = 15000
; Максимальное количество итераций при решении СЛАУ на полном пространстве и пространстве ядра
; в двухуровневом решателе
; [по умолчанию - 500]
max_iter_v_cycle_local = 100
; Описание типа решателя состоит из двух частей: название собственно решателя и название
; предобуславливателя. Общий синтаксис для описания решателей:
; "SolverName<PreconditionerName>""
; Описание всех используемых решателей:
; * "COCG" - COCG
; * "COCG_Smooth" - COCG со слаживанием невязки
; * "COCG_Smooth_OpenMP" - COCG со слаживанием невязки, распараллеленный через OpenMP
; * "COCG_Smooth_MKL" - COCG со слаживанием невязки, использующий библиотеку MKL
; * "COCR" - COCR
; * "COCR_Smooth" - COCR со слгавживанием невязки
; * "BicG_Complex" - Комплексный BicG
; * "BicG_Complex_Smooth" - Комплексный BicG со слаживанием невязки
; * "BicGSTab_Complex" - Комплексный BicGSTab
; * "BicGSTab_Complex_Smooth" - Комплексный BicGSTab со слаживанием невязки
; * "GMRES_Complex" - Комплексный GMRES
; * "GMRES_Complex_OpenMP" - Комплексный GMRES, распараллеленный через OpenMP
; * "GMRES_Complex_MKL" - Комплексный GMRES, использующий библиотеку MKL
; Описание всех используемых предобуславливателей:
; * "Nothing" - стандартный пустой предобуславливатель
; * "Nothing_OpenMP" - пустой предобуславливатель, распараллеленный через OpenMP
; * "Nothing_MKL" - пустой предобуславливатель, использующий библиотеку MKL
; * "Di" - стандартный диагональный предобуславливатель ( $A=S^*Q : S=Di, Q=I$ )
; * "Di_OpenMP" - диагональный предобуславливатель, распараллеленный через OpenMP
; * "Di_MKL" - диагональный предобуславливатель, использующий библиотеку MKL
; * "LLT" - стандартный предобуславливатель неполным  $LL^T$  разложением ( $A=S^*Q : S=L, Q=L^T$ )
; * "LLT_MKL" - предобуславливатель неполным  $LL^T$  разложением, использующий библиотеку MKL
; * "LDLT" - стандартный предобуславливатель неполным  $LDL^T$  разложением ( $A=S^*Q : S=L^*D, Q=L^T$ )
; * "GS" - стандартный предобуславливатель типа Гаусса-Зейделя ( $A=S^*Q : S=(D+L)^{-1}*D^*(D+U)^{-1}, Q=I$ )
; Название решателя для решения основной СЛАУ
; [по умолчанию - COCG_Smooth<LLT>]
solver_name = "COCG_Smooth<LLT>"
```

```

; Названия решателя для решения СЛАУ неоднородных краевых условий первого типа
; [по умолчанию - COCG_Smooth<LLT>]
solver_name_bound = "COCG_Smooth<LLT>"
; Название решателя для решения СЛАУ на полном пространстве в двухуровневом решателе
; [по умолчанию - COCG_Smooth<LLT>]
solver_name_v_cycle_full = "GMRES<LLT>"
; Название решателя для решения СЛАУ на пространстве ядра в двухуровневом решателе
; [по умолчанию - COCG_Smooth<LLT>]
solver_name_v_cycle_ker = "GMRES<LLT>"
; Путь к файлу с конечноэлементной сеткой
; [по умолчанию - mesh.msh]
filename_mesh = "mesh.msh"
; Путь к файлу с параметрами физических областей
; [по умолчанию - phys.ini]
filename_phys = "phys.ini"
; Путь к файлу с параметрами PML-слоя
; [по умолчанию - пустая строка (файл не задан, то есть PML-слой не используется)]
filename_pml = "config_pml.ini"
; Путь к файлу для сохранения или загрузки весов решения
; [по умолчанию - пустая строка (файл не задан, то есть загрузка/сохранение не производятся)]
filename_slae = "sляе.txt"
; Для ускорения вычисления матрицы СЛАУ, выражения для краевых условий, правых частей и
; аналитического решения могут быть скомпилированы "на лету". В настоящий момент такая
; возможность предоставляется для процессорных архитектур x86, x86_64 и x86_32 и операционных
; систем, использующих MSVC, MinGW или SysV ABI. Доступные значения:
; * disable - не компилировать
; * extcall - использовать математические операции из стандартной математической
;   библиотеки C++
; * inline - компилировать, использовать собственные реализации математических операций (в
;   большинстве случаев такой метод работает значительно быстрее, но не удовлетворяет IEEE 754
; [по умолчанию - disable]
jit_type = disable

; Параметры краевых условий первого типа (по умолчанию)
[Boundary]
; Включено использование неоднородных краевых условий первого типа или нет?
; [по умолчанию - no]
enabled = yes
; Покомпонентно заданная функция от x, y, z, omega, epsilon, sigma, mu, k2
x = 0.0
y = 0.0
z = 0.0

; Параметры краевых условий первого типа (двумерная физическая область 42)
[Boundary.42]
; Покомпонентно заданная функция от x, y, z, omega, epsilon, sigma, mu, k2
x = x + y * sigma - z + i * epsilon / mu ^ omega
y = x + y * sigma - z + i * epsilon / mu ^ omega
z = x + y * sigma - z + i * epsilon / mu ^ omega

; Параметры правой части (по умолчанию)
[Right]
; Включено использование неоднородной правой части или нет?
; [по умолчанию - no]
enabled = yes
; Покомпонентно заданная функция от x, y, z, omega, epsilon, sigma, mu, k2
x = 0.0
y = 0.0
z = 0.0

; Параметры правой части (трехмерная физическая область 42)
[Right.42]
; Покомпонентно заданная функция от x, y, z, omega, epsilon, sigma, mu, k2
x = x + y * sigma - z + i * epsilon / mu ^ omega
y = x + y * sigma - z + i * epsilon / mu ^ omega
z = x + y * sigma - z + i * epsilon / mu ^ omega

; Параметры аналитического решения (по умолчанию)
[Analytical]
; Включено использование аналитического решения или нет?
; [по умолчанию - no]
enabled = yes
; Покомпонентно заданная функция от x, y, z, omega, epsilon, sigma, mu, k2
x = 0.0
y = 0.0
z = 0.0

```

```

; Параметры аналитического решения (трехмерная физическая область область 42)
[Analytical.42]
; Покомпонентно заданная функция от x, y, z, omega, epsilon, sigma, mu, k2
x = x + y * sigma - z + i * epsilon / mu ^ omega
y = x + y * sigma - z + i * epsilon / mu ^ omega
z = x + y * sigma - z + i * epsilon / mu ^ omega

; Параметры 3D постпроцессора
[Postprocessing.1]
; Тип 1d, 2d или 3d
; [по умолчанию - 3d]
type = 3d
; Для 3d можно настроить только имя файла
; [по умолчанию - plot.plt]
filename = "plot.plt"
; Добавить ли метку времени запуска расчетов в имя файла?
; [по умолчанию - no]
timestamp = yes

; Параметры 2D постпроцессора
[Postprocessing.2]
; Тип 1d, 2d или 3d
; [по умолчанию - 3d]
type = 2d
; Имя файла
; [по умолчанию - plot.plt]
filename = "plot_y=0.dat"
; Добавить ли метку времени запуска расчетов в имя файла?
; [по умолчанию - no]
timestamp = yes
; Имя и значение фиксированной переменной
slice_var_name = Y
slice_var_value = 0.0
; Имя первой свободной переменной + откуда докуда брать значения + к-во точек
var1_name = X
var1_from = -700
var1_to = 700
var1_num = 70
; Имя второй свободной переменной + откуда докуда брать значения + к-во точек
var2_name = Z
var2_from = -700
var2_to = 700
var2_num = 70

; Параметры 1D постпроцессора
[Postprocessing.3]
; Тип 1d, 2d или 3d
; [по умолчанию - 3d]
type = 1d
; Имя файла
; [по умолчанию - plot.plt]
filename = "z=-100.dat"
; Добавить ли метку времени запуска расчетов в имя файла?
; [по умолчанию - no]
timestamp = yes
; Имена и значения двух фиксированных переменных
line_var1_name = Y
line_var1_value = 0.0
line_var2_name = Z
line_var2_value = -100
; Имя свободной переменной + откуда докуда брать значения + к-во точек
var_name = X
var_from = -600
var_to = 600
var_num = 70

```

Пример конфигурационного файла физических областей

```
; Глобальные физические параметры
[Global]
; Частота источника поля, Гц
frequency = 15.915494309189533576888
; Магнитная проницаемость (относительная)
; [по умолчанию - mu0]
mu = 1
; Диэлектрическая проницаемость (относительная)
; [по умолчанию - eps0]
eps = 1
; Электрическая проводимость, См/м
; [по умолчанию - 0]
sigma = 10

; Параметры среды для конкретной трехмерной физической области с номером 27
[Phys3D.27]
; Магнитная проницаемость (относительная)
; [по умолчанию как в Global]
mu = 1
; Диэлектрическая проницаемость (относительная)
; [по умолчанию как в Global]
eps = 1
; Электрическая проводимость, См/м
; [по умолчанию как в Global]
sigma = 10

; Параметры среды для конкретной двумерной физической области с номером 28
; (граница трехмерной области 27)
[Phys2D.28]
; Родительская область (трехмерная)
; [по умолчанию наследуется от Global]
parent = 27
; Тип краевого условия, 1 или 2
; [по умолчанию - 2]
boundary = 1

; Параметры среды для конкретной одномерной физической области с номером 1
; (ребра для описания петли, являющейся источником поля)
[Phys1D.1]
; Родительская область (трехмерная)
; [по умолчанию наследуется от Global]
parent = 27
; Электрический ток в линии (A)
; [по умолчанию - 0]
current = 1.0
```

Пример конфигурационного файла PML-слоя

```
; Глобальные параметры PML-слоя
[PML]
; Границы внутренней области без PML-слоя
; [если не указано - определяются автоматически]
x0 = -600
x1 = 600
y0 = -600
y1 = 600
z0 = -600
z1 = 600
; Вещественная и мнимая части коэффициента комплексного растяжения
; [по умолчанию растяжение производиться не будет]
chi_real = 2
chi_imag = 3
; Степенной параметр m
; [по умолчанию m = 3]
m = 3
; Толщина PML-слоя
; [по умолчанию width = 100]
width = 100

; Параметры PML-слоя для конкретной физической области с номером 21
[PML.21]
```

```

; вещественная и мнимая части коэффициента комплексного растяжения
; [по умолчанию как в секции PML]
chi_real = 5
chi_imag = 0
; Степенной параметр m
; [по умолчанию как в секции PML]
m = 3
; Толщина PML-слоя
; [по умолчанию как в секции PML]
width = 100

```

Пример конфигурационного файла сравнения решений

```

; Область, внутри которой нужно смотреть разницу
[Area.1]
; Тип области, раз смотрим разницу, значит included
type = included
; Координаты области
x0 = -500
x1 = 500
y0 = -500
y1 = 500
z0 = -500
z1 = 500

; Область, внутри которой НЕ нужно смотреть разницу
[Area.2]
; Тип области, раз НЕ смотрим разницу, значит excluded
type = excluded
; Координаты области
x0 = -180
x1 = 180
y0 = -180
y1 = 180
z0 = -180
z1 = 180

```

Приложение Б. Интерфейсы основных классов программного комплекса

Файл vfem/vfem/vfem.h

```
#if !defined(VFEM_H_INCLUDED)
#define VFEM_H_INCLUDED

#include "../common/common.h"
#include "../common/config.h"
#include "../elements/edge.h"
#include "../elements/triangle.h"
#include "../elements/tetrahedron.h"
#include "../vfem/phys.h"
#include "../vfem/slae.h"

#define VFEM_USE_PML

#if defined(VFEM_USE_PML)
typedef tetrahedron_pml finite_element;
#else
typedef tetrahedron finite_element;
#endif

// Правая часть
cvector3 func_rp(const point & p, phys_area & phys, void * data);

// Функция неоднородных первых краевых условий
cvector3 func_b1(const point & p, phys_area & phys, void * data);

// Функция аналитического решения
cvector3 func_true(const point & p, phys_area & phys, void * data);

#if defined(VFEM_USE_PML)
// Коэффициент S для PML
cvector3 get_s(const point & p, const tetrahedron_pml * fe, const phys_pml_area * phys_pml);

// Проверка, PML или нет
bool is_pml(const point & p, const finite_element * fe, const phys_pml_area * phys_pml);
#endif

// Класс векторный МКЭ
class VFEM
{
public:
    VFEM();

    // Конфигурация
    config_type config;

    // Ввод данных
    bool input_phys(const string & phys_filename);
    bool input_mesh(const string & gmsh_filename);
    // Процедура для сборки СЛАУ
    void make_struct();
    void make_data();
    // Запуск решения СЛАУ
    void solve();
    // Вывод данных в 3D сетке
    bool output(const string & tecplot_filename);
    // Вывод данных в 2D сетке
    bool output_slice(const string & tecplot_filename, char slice_var, double slice_val,
                      char var1, double min_var1, double max_var1, size_t num_var_1,
                      char var2, double min_var2, double max_var2, size_t num_var_2);
    // Вывод данных по линии
    bool output_line(const string & tecplot_filename, char line_var1, double line_val1,
                     char line_var2, double line_val2, char var3, double min_var3,
                     double max_var3, size_t num_var);

    // Поиск конечного элемента по точке
    finite_element * get_fe(const point & p) const;

    // Решение в точке
    cvector3 solution(const point & p) const;
    cvector3 solution(const point & p, const finite_element * fe) const;
```

```

// Ротор решения в точке
cvector3 rotor(const point & p) const;
cvector3 rotor(const point & p, const finite_element * fe) const;
// Дивергенция решения в точке
complex<double> divergence(const point & p) const;
complex<double> divergence(const point & p, const finite_element * fe) const;

// Конечные элементы (тетраэдры)
vector<finite_element> fes;

void calculate_diff();

// Основная СЛАУ
SLAE slae;
// СЛАУ на ядре
SLAE ker_slae;
// СЛАУ по границе
SLAE surf_slae;

// Узлы
vector<point> nodes;
// Ребра
set<edge> edges;
// Границы
set<face> faces;
// Ребра с источниками
vector<edge_src> edges_src;
// Треугольники
vector<triangle *> trs;
// Треугольники простые
// (не используется при config.boundary_enabled == true)
vector<triangle_base> trs_base;
// Треугольники полные, для первых краевых
// (не используется при config.boundary_enabled == false)
vector<triangle_full> trs_full;
// Точечные источники
vector<pair<point, cvector3>> pss;
// Физические области
map<phys_id, phys_area> phys;

// Число степеней свободы
size_t dof_num;
// Число степеней свободы ядра
size_t ker_dof_num;
// Соответствие глобальных степеней свободы и по границе
// (не используется при config.boundary_enabled == false)
map<size_t, size_t> global_to_local;
// Степени свободы с первыми краевыми
// (не используется при config.boundary_enabled == true)
set<size_t> dof_first;
// Степени свободы с первыми краевыми у ядра
set<size_t> ker_dof_first;
// Восьмичное дерево поиска
octree<finite_element> tree;

// Получение количества степеней свободы тетраэдра
size_t get_tet_dof_num(const tetrahedron_base * fe) const;
// Получение степеней свободы тетраэдра в глобальной матрице
size_t get_tet_dof(const tetrahedron_base * fe, size_t i) const;
// Получение количества степеней свободы ядра тетраэдра
size_t get_tet_ker_dof_num(const tetrahedron_base * fe) const;
// Получение степеней свободы тетраэдра в матрице ядра
size_t get_tet_ker_dof(const tetrahedron_base * fe, size_t i) const;
// Получение степеней свободы треугольника в глобальной матрице
size_t get_tr_dof(const triangle * tr, size_t i) const;
// Получение степеней свободы треугольника в матрице ядра
size_t get_tr_ker_dof(const triangle * tr, size_t i) const;
// Получение степеней свободы треугольника в матрице по границе
size_t get_tr_surf_dof(const triangle * tr, size_t i) const;

// Генерация портрета глобальной матрицы
void generate_portrait();
// Генерация портрета глобальной матрицы ядра
void generate_ker_portrait();
// Сборка глобальной матрицы
void assemble_matrix();

```

```

// Применение краевых условий
void applying_bound();
// Генерация портрета по границе
void generate_surf_portrait();
// Применение источников на ребрах
void apply_edges_sources();
// Применение точечных источников
void apply_point_sources();
// Применение электродов
void apply_electrodes();

protected:
    // Добавление локальной матрицы от одного КЭ в глобальную
    template<typename T>
    void process_fe_MpG(T * curr_fe);
    // Добавление локальной матрицы ядра от одного КЭ в глобальную матрицу ядра
    template<typename T>
    void process_fe_K(T * curr_fe);
    // Добавление локальной правой части от одного КЭ в глобальную правую часть
    template<typename T>
    void process_fe_rp(T * curr_fe);

    // Генерация абстрактного портрета
    template<typename U, typename V>
    void generate_abstract_portrait(size_t all_dof_num, const vector<U> & elems, SLAE & slae_curr,
                                    size_t (VFEM::*get_dof_num)(const V *) const,
                                    size_t (VFEM::*get_dof)(const V *, size_t) const) const;

    // Добавление ребра в множество ребер
    size_t add_edge(edge ed, set<edge> & edges_set);
    // Добавление грани в множество граней
    size_t add_face(face fc, set<face> & faces_set);

#if defined(VFEM_USE_PML)
    cpoint convert_point_to_pml(const point * p, const finite_element * fefe) const;
    void input_pml();
    // Параметры PML
    phys_pml_area phys_pml;
#endif

    // Проектирование на пространство ядра
    void to_kernel_space(const complex<double> * in, complex<double> * out) const;
    // Интерполяция на полное пространство
    void to_full_space(const complex<double> * in, complex<double> * out) const;
    // Скалярное произведение
    double dot_prod_self(const complex<double> * a) const;
    // Умножение матрицы с полного пространства на вектор
    void mul_matrix(const complex<double> * f, complex<double> * x) const;
    // Подсчет невязки
    void calc_residual(const complex<double> * x0, complex<double> * p) const;
};

#endif // VFEM_H_INCLUDED

```

Файл vfem/vfem/phys.h

```

#if !defined(PHYS_H_INCLUDED)
define PHYS_H_INCLUDED

#include "../common/common.h"

namespace consts
{
    static const double c = 299792458.0;                                // Скорость света
    static const double mu0 = 4.0e-7 * M_PI;                            // Магн. пр. вакуума
    static const double epsilon0 = 1.0 / (mu0 * c * c); // Диэл. пр. вакуума
}

// Класс физическая область
class phys_area
{
public:
    double omega;                                              // Циклическая частота
    double mu;                                                 // Магнитная проницаемость (относительная)
}

```

```

array_t<evaluator_xyz<double>, 3> sigma;      // Электрическая проводимость
double epsilon;                                // Диэлектрическая проницаемость (относительная)
size_t gmsh_num;                             // Номер области в Gmsh
size_t type_of_elem;                         // Тип элементов
size_t type_of_bounds;                      // Тип краевого условия
double J0;                                    // Мощность источника
double E0;                                    // Электрическое поле от электрода
phys_area();                                 // Конструктор по умолчанию
};

// Класс для индексации физических областей
class phys_id
{
public:
    size_t type_of_element;
    size_t gmsh_num;
    phys_id();
    phys_id(size_t type_of_element, size_t gmsh_num);
    bool operator < (const phys_id & other) const;
};

// Класс для хранения параметров PML для одной физической области
class pml_config_parameter
{
public:
    complex<double> chi;
    double width;
    double m;
    pml_config_parameter(complex<double> n_chi = 1.0, double n_width = 100.0, double n_m = 3.0);
};

// Класс для хранения параметров PML
class phys_pml_area
{
public:
    double x0, x1;
    double y0, y1;
    double z0, z1;
    map<size_t, pml_config_parameter> params;
};

#endif // PHYS_H_INCLUDED

```

Файл vfem/vfem/slae.h

```

#ifndef !defined(SLAE_H_INCLUDED)
#define SLAE_H_INCLUDED

#include "../common/common.h"

// Класс СЛАУ
class SLAE
{
public:
    SLAE();
    ~SLAE();
    void solve(const string & name, double eps, size_t max_iter);
    void init(const string & name);
    void step_solve(complex<double> * solution, const complex<double> * rp_s,
                    double eps, size_t max_iter);
    void alloc_all(size_t n_size, size_t gg_size);
    void dealloc_all();
    void add(size_t i, size_t j, const complex<double> & elem);
    complex<double> * gg, * di, * rp, * x;
    size_t * ig, * jg;
    size_t n;
    bool dump(const string & filename) const;
    bool restore(const string & filename);
    bool dump_x(const string & filename) const;
    bool restore_x(const string & filename);
private:
    preconditioner_interface<complex<double>, size_t> * precond;
    symmetric_solver_interface<complex<double>, size_t> * solver;
};

```

```
#endif // SLAE_H_INCLUDED
```

Файл vfem/problems/problems.h

```
#if !defined(PROBLEMS_H_INCLUDED)
#define PROBLEMS_H_INCLUDED

#include "../common/common.h"
#include "../vfem/vfem.h"

// Контеинер области для сравнения
class diff_area
{
public:
    bool included;
    point p1, p2;
    diff_area();
};

// Постпроцессор
void postprocessing(VFEM & v, const char * timebuf);

// Простое сравнение (на геометрически идентичных подобластях сеток и одинаковых базисах)
void compare_simple(VFEM & master, VFEM & slave, const vector<diff_area> & areas);

// Сложное сравнение (на произвольных подобластях сеток и базисах)
void compare_complex(VFEM & master, VFEM & slave, const vector<diff_area> & areas);

// Чтение конфигурационного файла для сравнения
bool input_diff(const string & filename, vector<diff_area> & areas);

#endif // PROBLEMS_H_INCLUDED
```

Файл vfem/elements/tetrahedron.h

```
#if !defined(TETRAHEDRON_H_INCLUDED)
#define TETRAHEDRON_H_INCLUDED

#include "../common/common.h"
#include "../common/config.h"
#include "../elements/edge.h"
#include "../vfem/phys.h"

typedef cvector3(* eval_func)(const point &, phys_area &, void *);

// Индексы для построения базисных функций на тетраэдрах
namespace tet_basis_indexes
{
    // Edges (Ребра) // k, l : k < l
    extern const size_t ind_e[6][2];
    // Faces (Границы) // j, k, l : j < k < l
    extern const size_t ind_f[4][3];
}

// Класс тетраэдр (абстрактный)
class tetrahedron_base : public tetrahedron_basic_3d<edge, face, phys_area>
{
public:
    void init(const basis_type * basis);

    const basis_type * basis; // Параметры базиса

    // Базисные функции
    vector3 w(size_t i, const point & p) const;
    // Дивергенции базисных функций
    double divw(size_t i, const point & p) const;
    // Роторы базисных функций
    vector3 rotw(size_t i, const point & p) const;
    // Базисные функции ядра
    vector3 kerw(size_t i, const point & p) const;
```

```

        trio<double, vector3, cvector3>
        diff_normL2(const array_t<complex<double> > & q, eval_func func, void * data);
        trio<double, vector3, cvector3>
        diff_normL2(const array_t<complex<double> > & q, const array_t<complex<double> > & q_true);
        trio<double, vector3, cvector3>
        normL2(eval_func func, void * data);
        trio<double, vector3, cvector3>
        normL2(const array_t<complex<double> > & q_true);
    };

    // Класс тетраэдр (обычный)
    class tetrahedron : public tetrahedron_base
    {
    public:
        // Локальная правая часть
        array_t<complex<double> > rp(eval_func func, void * data);
        // Локальная матрица полного пространства
        matrix_t<complex<double> > MpG();
        // Локальная матрица ядра
        matrix_t<complex<double> > K();
    };

    // Класс тетраэдр (для работы с PML-краевыми)
    class tetrahedron_pml : public tetrahedron
    {
    public:
        tetrahedron_pml();
        // Инициализация PML-координат
        void init_pml(cvector3(* get_s)(const point &, const tetrahedron_pml *, const phys_pml_area *),
                      const phys_pml_area * phys_pml, const cpoint * nodes_pml);

        // Локальная правая часть
        array_t<complex<double> > rp(eval_func func, void * data);
        // Локальная матрица полного пространства
        matrix_t<complex<double> > MpG();
        // Локальная матрица ядра
        matrix_t<complex<double> > K();

        // Получить указатель на обычный тетраэдр
        const tetrahedron * to_std() const;
        tetrahedron * to_std();

    protected:
        cvector3(* get_s)(const point &, const tetrahedron_pml *, const phys_pml_area *);
        const phys_pml_area * phys_pml;

        // Матрица L-координат (в PML)
        matrix_t<complex<double> > L_pml;
        // Градиент L-координаты (в PML)
        cvector3 grad_lambda_pml(size_t i) const;
        // L-координаты (в PML)
        complex<double> lambda_pml(size_t i, const cpoint & p) const;

        // Точки Гаусса (в PML)
        array_t<cpoint> gauss_points_pml;
        // Якобиан (в PML)
        complex<double> jacobian_pml;

        // Базисные функции (в PML)
        cvector3 w_pml(size_t i, const cpoint & p) const;
        // Роторы базисных функций (в PML)
        cvector3 rotw_pml(size_t i, const cpoint & p, const point & p_non_PML) const;
        // Базисные функции ядра (в PML)
        cvector3 kerw_pml(size_t i, const cpoint & p, const point & p_non_PML) const;
    };
}

#endif // TETRAHEDRON_H_INCLUDED

```

Файл *vfem/elements/triangle.h*

```
#if !defined(TRIANGLE_H_INCLUDED)
#define TRIANGLE_H_INCLUDED

#include "../common/common.h"
#include "../common/config.h"
#include "../elements/edge.h"
#include "../vfem/phys.h"

typedef cvector3(* eval_func)(const point &, phys_area &, void *);

// Класс треугольник (обычный)
class triangle_base : public virtual triangle_basic<point, edge, face, phys_area>
{
public:
    virtual void init(const basis_type * ) {}
    virtual matrix_t<double> M() const;
    virtual array_t<complex<double>> rp(eval_func, void *);
};

// Класс треугольник (полный, для работы с первыми неоднородными краевыми)
class triangle_full : public triangle_base, private triangle_basic_3d<edge, face, phys_area>
{
public:
    triangle_full(const triangle_base & other = triangle_base());
    virtual void init(const basis_type * basis);
    // Локальная матрица массы
    virtual matrix_t<double> M() const;
    // Локальная правая часть
    virtual array_t<complex<double>> rp(eval_func func, void * data);

protected:
    // Параметры базиса
    const basis_type * basis;
    // Базисные функции
    vector3 w(size_t i, const point & p) const;

    double integrate_w(size_t i, size_t j) const;
    complex<double> integrate_fw(eval_func func, size_t i, void * data);
};

typedef triangle_base triangle;

#endif // TRIANGLE_H_INCLUDED
```

Файл *vfem/elements/edge.h*

```
#if !defined(EDGE_H_INCLUDED)
#define EDGE_H_INCLUDED

#include "../common/common.h"
#include "../vfem/phys.h"

// Структура ребро
struct edge : public edge_basic<point>
{
    edge(point * beg_ = NULL, point * end_ = NULL, size_t num_ = 0);
    edge(point & beg_, point & end_, size_t num_ = 0);
    double length();
};

// Структура ребро с источником
struct edge_src : public edge
{
    double direction; // Направление
    phys_area * phys; // Физическая область
    edge * edge_main; // Основное ребро
    edge_src();
    edge_src(const edge_src & f);
    const phys_area & get_phys_area() const;
```

```

    edge_src & operator = (const edge_src & other);
};

#endif // EDGE_H_INCLUDED

```

Файл vfem/common/config.h

```

#ifndef !defined(CONFIG_H)
#define CONFIG_H

#include "../common/common.h"
#include "../common/evaluator_helmholtz.h"
#include "../vfem/phys.h"

using namespace std;

// Распределение вычислителей по тредам
namespace threads_config
{
    static const size_t matrix_full = 0;
    static const size_t matrix_ker = 1;
    static const size_t right_part = 2;
    static const size_t boundary = 0;
    static const size_t analytical = 0;
}

// BASIS_ORDER BASIS_TYPE
// Первый порядок I типа (неполный) 1 1
// Первый порядок II типа (полный) 1 2
// Второй порядок I типа (неполный) 2 1
// Второй порядок II типа (полный) 2 2

// Конфигурация базиса
struct basis_type
{
    // Количество базисных функций на тетраэдрах
    size_t tet_bf_num;
    // Количество базисных функций ядра на тетраэдрах
    size_t tet_ker_bf_num;
    // Количество базисных функций на треугольниках
    size_t tr_bf_num;
    // Количество базисных функций ядра на треугольниках
    size_t tr_ker_bf_num;
    // Порядок базиса
    size_t order;
    size_t type;
    // Интегрирование
    tetrahedron_integration tet_int;
    triangle_integration tr_int;
};

// Вычислитель для вычисляемых значений
class evaluator3
{
public:
    evaluator3();
    // Вычислители по физическим областям
    map<size_t, array_t<evaluator_helmholtz, 3>> values;
    // Вычислители по умолчанию
    array_t<evaluator_helmholtz, 3> default_value;
    // Тип JIT-компилатора в вычислителях
    enum jit_types
    {
        JIT_DISABLE,
        JIT_INLINE,
        JIT_EXTCALL
    };
};

// Конфигурация 1D постпроцессора
struct postprocessor_1d
{
    char line_var1_name;
    double line_var1_value;
};

```

```

char line_var2_name;
double line_var2_value;
char var_name;
double var_from;
double var_to;
size_t var_num;
};

// Конфигурация 2D постпроцессора
struct postprocessor_2d
{
    char slice_var_name;
    double slice_var_value;
    char var1_name;
    double var1_from;
    double var1_to;
    size_t var1_num;
    char var2_name;
    double var2_from;
    double var2_to;
    size_t var2_num;
};

// Общая конфигурация постпроцессора
class postprocessor
{
public:
    unsigned char type;
    string filename;
    bool timestamp;
    union
    {
        postprocessor_1d param_1d;
        postprocessor_2d param_2d;
    };
    postprocessor();
};

// Класс конфигурации
class config_type
{
public:
    config_type();
    // Загрузка значений из файла
    bool load(const string & filename);
    bool load_pml(const string & filename);

    // ===== VFEM =====

    // Конфигурация базиса
    basis_type basis;
    // Точность решения СЛАУ
    double eps_slae;
    // Точность решения СЛАУ для краевых
    double eps_slae_bound;
    // Включен V-цикль или нет
    bool v_cycle_enabled;
    // Точность начального решения перед запуском v-цикла
    double gamma_v_cycle_0;
    // Точность решения на полном пространстве
    double gamma_v_cycle_full;
    // Точность решения на пространстве ядра
    double gamma_v_cycle_ker;
    // Максимальное количество итераций для не V-цикловых решателей
    size_t max_iter;
    // Максимальное локальное число итераций V-цикла
    size_t max_iter_v_cycle_local;
    // Решатель для обычной СЛАУ
    string solver_name;
    // Решатель для СЛАУ с краевыми
    string solver_name_bound;
    // Решатель для СЛАУ на полном пространстве
    string solver_name_v_cycle_full;
    // Решатель для СЛАУ на пространстве ядра
    string solver_name_v_cycle_ker;
    // Сетка

```

```

string filename_mesh;
// Параметры физических областей
string filename_phys;
// Куда сохранять веса решения
string filename_slae;
// Тип JIT-компилятора в вычислителях
evaluator3::jit_types jit_type;

// ===== Boundary =====

evaluator3 boundary;
bool boundary_enabled;

// ===== Right =====

evaluator3 right;
bool right_enabled;

// ===== Analytical =====

evaluator3 analytical;
bool analytical_enabled;

// ===== Postprocessing =====

map<size_t, postprocessor> post;

// ===== PML =====

phys_pml_area phys_pml;
string filename_pml;

protected:
    // Загрузка значений по-умолчанию
    void load_defaults();
    // Пост-загрузочная инициализация
    bool init(bool status);
};

#endif // CONFIG_H

```

Файл vfem/common/evaluator_helmholtz.h

```

#ifndef !defined(EVALUATOR_HELMHOLTZ_H)
#define EVALUATOR_HELMHOLTZ_H

#include "../common/common.h"

// Вычислитель, заточенный под параметры уравнения Гельмгольца
class evaluator_helmholtz : public evaluator<std::complex<double>>
{
public:
    evaluator_helmholtz();
    evaluator_helmholtz(const std::string & str);
    evaluator_helmholtz(const evaluator_helmholtz & other);
    const evaluator_helmholtz & operator = (const evaluator_helmholtz & other);
    void set_x(const std::complex<double> & x);
    void set_y(const std::complex<double> & y);
    void set_z(const std::complex<double> & z);
    void set_J0(const std::complex<double> & J0);
    void set_omega(const std::complex<double> & omega);
    void set_epsilon(const std::complex<double> & epsilon);
    void set_sigma(const std::complex<double> & sigma);
    void set_mu(const std::complex<double> & mu);
    void set_k2(const std::complex<double> & k2);

private:
    std::complex<double> * m_x, * m_y, * m_z;
    std::complex<double> * m_J0, * m_omega, * m_epsilon, * m_sigma, * m_mu, * m_k2;
    void update_cache();
};

#endif // EVALUATOR_HELMHOLTZ_H

```

Файл *vfem/common/common.h*

```
#if !defined(COMMON_H_INCLUDED)
#define COMMON_H_INCLUDED

#if defined(_MSC_VER) && !defined(_CRT_SECURE_NO_WARNINGS)
#define _CRT_SECURE_NO_WARNINGS
#endif

#if defined(_MSC_VER)
#if !defined(_USE_MATH_DEFINES)
#define _USE_MATH_DEFINES
#endif
#include <math.h>
#else
#include <cmath>
#endif

#if !defined(M_PI)
#define M_PI 3.14159265358979323846
#endif

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <cfloat>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <string>
#include <complex>
#include <map>
#include <utility>
#include <cassert>
#include <cctype>
#include <list>
#include <ctime>

#include "../../../../core/containers/fem/triangle_basic_3d.h"
#include "../../../../core/containers/fem/tetrahedron_basic_3d.h"
#include "../../../../core/containers/generic/trio.h"
#include "../../../../core/containers/generic/array_t.h"
#include "../../../../core/containers/generic/matrix_t.h"
#include "../../../../core/containers/geometry/edge_basic.h"
#include "../../../../core/containers/geometry/face_triangle_basic.h"
#include "../../../../core/containers/geometry/point3_t.h"
#include "../../../../core/containers/geometry/vector3_t.h"
#include "../../../../core/containers/tree/quadtree.h"
#include "../../../../core/containers/tree/octree.h"
#include "../../../../core/cubatures/triangle_integration.h"
#include "../../../../core/cubatures/tetrahedron_integration.h"
#include "../../../../core/utils/cxxversion.h"
#include "../../../../core/utils/fpu.h"
#include "../../../../core/utils/nosighup.h"
#include "../../../../core/utils/strings.h"
#include "../../../../core/utils/progress.h"
#include "../../../../core/utils/timers.h"
#include "../../../../core/utils/inifile.h"
#include "../../../../core/evaluator/evaluator.h"
#include "../../../../core/evaluator/evaluator_xyz.h"
#include "../../../../core/solvers/CSLR/solvers_factory.h"
#include "../../../../core/wrappers/omp_wrapper.h"

using namespace std;
using namespace core::containers::fem;
using namespace core::containers::generic;
using namespace core::containers::geometry;
using namespace core::containers::tree;
using namespace core::cubatures;
using namespace core::utils;
using namespace core::utils::nosighup;
using namespace core::utils::strings;
using namespace core::utils::progress;
```

```

using namespace core::utils::timers;
using namespace core::utils::fpu;
using namespace core::solvers::CSLR::factory;
using namespace core::solvers::CSLR::preconditioners;
using namespace core::solvers::CSLR::symmetric;
using namespace core::wrappers::omp;

typedef point3_t<double> point;
typedef point3_t<complex<double>> cpoint;
typedef vector3_t<double> vector3;
typedef vector3_t<complex<double>> cvector3;

typedef face_triangle_basic<point> face;

#endif // COMMON_H_INCLUDED

```

Файл core/containers/fem/tetrahedron_basic.h

```

#ifndef !defined(CONTAINERS_FEM_TETRAHEDRON_BASIC_H_INCLUDED)
#define CONTAINERS_FEM_TETRAHEDRON_BASIC_H_INCLUDED

#include <cassert>
#include <cstddef>

namespace core { namespace containers { namespace fem {

/**
 * @brief Класс тетраэдр
 */
template<typename point, typename edge, typename face, typename phys_area>
class tetrahedron_basic
{
public:
    tetrahedron_basic();

    const point & get_node(std::size_t i) const;
    point & get_node(std::size_t i);
    const point * get_node_ptr(std::size_t i) const;
    point * get_node_ptr(std::size_t i);
    void set_node(std::size_t i, point * new_node);
    void set_node(std::size_t i, point & new_node);

    const edge & get_edge(std::size_t i) const;
    edge & get_edge(std::size_t i);
    const edge * get_edge_ptr(std::size_t i) const;
    edge * get_edge_ptr(std::size_t i);
    void set_edge(std::size_t i, edge * new_edge);
    void set_edge(std::size_t i, edge & new_edge);

    const face & get_face(std::size_t i) const;
    face & get_face(std::size_t i);
    const face * get_face_ptr(std::size_t i) const;
    face * get_face_ptr(std::size_t i);
    void set_face(std::size_t i, face * new_face);
    void set_face(std::size_t i, face & new_face);

    const phys_area & get_phys_area() const;
    phys_area & get_phys_area();
    const phys_area * get_phys_area_ptr() const;
    phys_area * get_phys_area_ptr();
    void set_phys_area(phys_area * new_phys_area);
    void set_phys_area(phys_area & new_phys_area);

private:
    point * m_nodes[4];    // Узлы
    edge * m_edges[6];     // Ребра
    face * m_faces[4];    // Границы
    phys_area * m_phys;   // Физическая область
};

}}} // namespace core::containers::fem

#endif // CONTAINERS_FEM_TETRAHEDRON_BASIC_H_INCLUDED

```

Файл core/containers/fem/tetrahedron_basic_3d.h

```
#if !defined(CONTAINERS_FEM_TETRAHEDRON_BASIC_3D_H_INCLUDED)
#define CONTAINERS_FEM_TETRAHEDRON_BASIC_3D_H_INCLUDED

#include <cmath>
#include "tetrahedron_basic.h"
#include "../generic/array_t.h"
#include "../generic/matrix_t.h"
#include "../geometry/point3_t.h"
#include "../geometry/vector3_t.h"
#include "../../../cubatures/tetrahedron_integration.h"

namespace core { namespace containers { namespace fem {

/**
 * @brief Класс тетраэдр в трехмерном пространстве
 */
template<typename edge, typename face, typename phys_area>
class tetrahedron_basic_3d
    : public tetrahedron_basic<geometry::point3_t<double>, edge, face, phys_area>
{
public:
    tetrahedron_basic_3d()
    {
        m_jacobian = 0.0;
    }

    /**
     * @brief Инициализация для работы в трехмерном пространстве
     * @param[in] tet_integration Параметры интегрирования
     */
    void init_3d(const cubatures::tetrahedron_integration & tet_integration);

    /**
     * @brief Определить, внутри тетраэдра точка или нет
     * @param[in] p Точка
     * @return true, если внутри ; false, если иначе
     */
    bool inside(const geometry::point3_t<double> & p) const;

    /**
     * @brief Определить, внутри тетраэдра точка или нет
     * @param[in] x Координата x точки
     * @param[in] y Координата y точки
     * @param[in] z Координата z точки
     * @return true, если внутри ; false, если иначе
     */
    bool inside(double x, double y, double z) const;

    /**
     * @brief Определить, лежит ли тетраэдр внутри куба или нет
     * @param[in] x0 Координата x начала куба
     * @param[in] x1 Координата x конца куба
     * @param[in] y0 Координата y начала куба
     * @param[in] y1 Координата y конца куба
     * @param[in] z0 Координата z начала куба
     * @param[in] z1 Координата z конца куба
     * @return true, если внутри ; false, если иначе
     */
    bool in_cube(double x0, double x1, double y0, double y1, double z0, double z1) const;

    /**
     * @brief Получить барицентр тетраэдра
     * @return Барицентр тетраэдра
     */
    const geometry::point3_t<double> & get_barycenter() const;

    /**
     * @brief Получить градиент L-координаты
     * @param[in] i Номер L-координаты
     * @return Градиент L-координаты
     */
    geometry::vector3_t<double> grad_lambda(std::size_t i) const;
```

```

    /**
     * @brief Получить значение L-координаты
     * @param[in] i Номер L-координаты
     * @param[in] p Точка, в которой нужно узнать значение
     * @return Значение L-координаты
     */
    double lambda(std::size_t i, const geometry::point3_t<double> & p) const;

    /**
     * @brief Получить якобиан для интегрирования
     * @return Якобиан
     */
    double get_jacobian() const;

    /**
     * @brief Получить точку интегрирования (точку Гаусса)
     * @param[in] i Номер точки интегрирования
     * @return Точка интегрирования
     */
    const geometry::point3_t<double> & get_gauss_point(std::size_t i) const;

private:

    /**
     * @brief Барицентр тетраэдра
     */
    geometry::point3_t<double> m_barycenter;

    /**
     * @brief Матрица L-координат
     */
    generic::matrix_t<double> m_L;

    /**
     * @brief Точки Гаусса
     */
    generic::array_t<geometry::point3_t<double> > m_gauss_points;

    /**
     * @brief Якобиан
     */
    double m_jacobian;

    /**
     * @brief Параметры прямых для дерева
     */
    generic::matrix_t<double> m_edges_a, m_edges_b;
};

} } // namespace core::containers::fem

#endif // CONTAINERS_FEM_TETRAHEDRON_BASIC_3D_H_INCLUDED

```

Файл core/containers/fem/triangle_basic.h

```

#ifndef !defined(CONTAINERS_FEM_TRIANGLE_BASIC_H_INCLUDED)
#define CONTAINERS_FEM_TRIANGLE_BASIC_H_INCLUDED

#include <cassert>
#include <cstddef>

namespace core { namespace containers { namespace fem {

    /**
     * @brief Класс треугольник
     */
    template<typename point, typename edge, typename face, typename phys_area>
    class triangle_basic
    {
public:
    triangle_basic();

    const point & get_node(std::size_t i) const;
    point & get_node(std::size_t i);

```

```

const point * get_node_ptr(std::size_t i) const;
point * get_node_ptr(std::size_t i);
void set_node(std::size_t i, point * new_node);
void set_node(std::size_t i, point & new_node);

const edge & get_edge(std::size_t i) const;
edge & get_edge(std::size_t i);
const edge * get_edge_ptr(std::size_t i) const;
edge * get_edge_ptr(std::size_t i);
void set_edge(std::size_t i, edge * new_edge);
void set_edge(std::size_t i, edge & new_edge);

const face & get_face() const;
face & get_face();
const face * get_face_ptr() const;
face * get_face_ptr();
void set_face(face * new_face);
void set_face(face & new_face);

const phys_area & get_phys_area() const;
phys_area & get_phys_area();
const phys_area * get_phys_area_ptr() const;
phys_area * get_phys_area_ptr();
void set_phys_area(phys_area * new_phys_area);
void set_phys_area(phys_area & new_phys_area);

private:

    point * m_nodes[3]; // Узлы
    edge * m_edges[3]; // Ребра
    face * m_faces; // Границы
    phys_area * m_phys; // Физическая область
};

} } // namespace core::containers::fem

#endif // CONTAINERS_FEM_TRIANGLE_BASIC_H_INCLUDED

```

Файл core/containers/fem/triangle_basic_3d.h

```

#ifndef !defined(CONTAINERS_FEM_TRIANGLE_BASIC_3D_H_INCLUDED)
#define CONTAINERS_FEM_TRIANGLE_BASIC_3D_H_INCLUDED

#include <cmath>
#include "triangle_basic.h"
#include "../generic/array_t.h"
#include "../generic/matrix_t.h"
#include "../geometry/point3_t.h"
#include "../geometry/vector3_t.h"
#include "../../../cubatures/triangle_integration.h"

namespace core { namespace containers { namespace fem {

/**
 * @brief Класс треугольник в трехмерном пространстве
 */
template<typename edge, typename face, typename phys_area>
class triangle_basic_3d
    : public virtual triangle_basic<geometry::point3_t<double>, edge, face, phys_area>
{
public:
    triangle_basic_3d();

    /**
     * @brief Инициализация для работы в трехмерном пространстве
     * @param[in] tr_integration Параметры интегрирования
     */
    void init_3d(const cubatures::triangle_integration & tr_integration);

    /**
     * @brief Получить значение L-координаты
     * @param[in] i Номер L-координаты
     * @param[in] p Точка, в которой нужно узнать значение
     * @return Значение L-координаты
     */
}

```

```

*/
double lambda(std::size_t i, const geometry::point3_t<double> & p) const;

/**
 * @brief Получить градиент L-координат (в глобальных координатах)
 * @param[in] i Номер L-координаты
 * @return Градиент L-координаты в глобальных координатах
 */
geometry::vector3_t<double> grad_lambda(std::size_t i) const;

/**
 * @brief Перевод точки в локальную систему координат треугольника
 * @param[in] p Точка в глобальной системе координат
 * @return Точка в локальной системе координат
 */
geometry::point3_t<double> to_local(const geometry::point3_t<double> & p) const;

/**
 * @brief Перевод точки в глобальную систему координат
 * @param[in] p Точка в локальной системе координат
 * @return Точка в глобальной системе координат
 */
geometry::point3_t<double> to_global(const geometry::point3_t<double> & p) const;

/**
 * @brief Перевод вектора в глобальную систему координат
 * @param[in] v Вектор в локальной системе координат
 * @return Вектор в глобальной системе координат
 */
geometry::vector3_t<double> to_global(const geometry::vector3_t<double> & v) const;

/**
 * @brief Получить якобиан для интегрирования
 * @return Якобиан
 */
double get_jacobian() const;

/**
 * @brief Получить точку интегрирования (точку Гаусса)
 * @param[in] i Номер точки интегрирования
 * @return Точка интегрирования
 */
const geometry::point3_t<double> & get_gauss_point(std::size_t i) const;

private:

/**
 * @brief Матрица L-координат
 */
generic::matrix_t<double> m_L;

/**
 * @brief Матрица перехода между локальной и глобальной с.к.
 */
generic::matrix_t<double>, 3, 3> m_transition_matrix;

/**
 * @brief Точки Гаусса
 */
generic::array_t<geometry::point3_t<double> > m_gauss_points;

/**
 * @brief Якобиан
 */
double m_jacobian;
};

}} // namespace core::containers::fem

#endif // CONTAINERS_FEM_TRIANGLE_BASIC_3D_H_INCLUDED

```

Файл core/containers/tree/octree.h

```
#if !defined(CONTAINERS_TREE_OCTREE_H_INCLUDED)
#define CONTAINERS_TREE_OCTREE_H_INCLUDED

#include <vector>
#include <cstdlib>
#include <cmath>
#include <algorithm>

namespace core { namespace containers { namespace tree {

template<class element_type>
class octree
{
public:
    void make(double x0, double x1, double y0, double y1, double z0, double z1,
              std::vector<element_type> & elements);
    element_type * find(double x, double y, double z) const;
    void clear();

private:
    class octree_node
    {
public:
    octree_node();
    ~octree_node();
    void clear();
    void init_node(double x0, double x1, double y0, double y1, double z0, double z1,
                  std::size_t split_constant, std::size_t level_barier);
    void add_element(std::vector<element_type *> & added_elements);
    bool add_element(element_type * added_element);
    element_type * find(double x, double y, double z) const;

    std::size_t level;

private:
    bool point_in_node(double x, double y, double z) const;
    bool split_condition();
    void split();

    std::vector<element_type *> elements;
    octree_node * sub_nodes;
    double x0, x1, y0, y1, z0, z1;
    double eps_x, eps_y, eps_z;
    std::size_t split_constant;
    std::size_t level_barier;
};

    octree_node root;
};

}} // core::containers::tree

#endif // CONTAINERS_TREE_OCTREE_H_INCLUDED
```

Файл core/containers/geometry/edge_basic.h

```
#if !defined(CONTAINERS_GEOMETRY_EDGE_BASIC_H_INCLUDED)
#define CONTAINERS_GEOMETRY_EDGE_BASIC_H_INCLUDED

#include <cassert>
#include <cstddef>

namespace core { namespace containers { namespace geometry {

/***
 * @brief Структура ребро из двух узлов
 */
template<typename point>
struct edge_basic
{
```

```

point * nodes[2];    ///< Узлы ребра
std::size_t num;     ///< Номер ребра

edge_basic(const edge_basic & f);
edge_basic(point * beg_ = NULL, point * end_ = NULL, std::size_t num_ = 0);
edge_basic(point & beg_, point & end_, std::size_t num_ = 0);

point & operator [] (std::size_t i);
const point & operator [] (std::size_t i) const;

bool operator < (const edge_basic & t) const;
bool operator == (const edge_basic & t) const;
edge_basic & operator = (const edge_basic & other);
};

}} // namespace core::containers::geometry

#endif // CONTAINERS_GEOMETRY_EDGE_BASIC_H_INCLUDED

```

Файл core/containers/geometry/face_triangle_basic.h

```

#if !defined(CONTAINERS_GEOMETRY_FACE_TRIANGLE_BASIC_H_INCLUDED)
#define CONTAINERS_GEOMETRY_FACE_TRIANGLE_BASIC_H_INCLUDED

#include <cassert>
#include <cstddef>

namespace core { namespace containers { namespace geometry {

/**
 * @brief Структура треугольная грань из трех узлов
 */
template<typename point>
struct face_triangle_basic
{
    point * nodes[3];    ///< Узлы грани
    std::size_t num;     ///< Номер грани

    face_triangle_basic();
    face_triangle_basic(const face_triangle_basic & f);
    face_triangle_basic(point * p1, point * p2, point * p3, std::size_t num_ = 0);
    face_triangle_basic(point & p1, point & p2, point & p3, std::size_t num_ = 0);

    point & operator [] (size_t i);
    const point & operator [] (size_t i) const;

    bool operator < (const face_triangle_basic & t) const;
    bool operator == (const face_triangle_basic & t) const;
    face_triangle_basic & operator = (const face_triangle_basic & other);
};

}} // namespace core::containers::geometry

#endif // CONTAINERS_GEOMETRY_FACE_TRIANGLE_BASIC_H_INCLUDED

```

Файл core/containers/geometry/vector3_t.h

```

#if !defined(CONTAINERS_GEOMETRY_VECTOR3_T_H_INCLUDED)
#define CONTAINERS_GEOMETRY_VECTOR3_T_H_INCLUDED

#include <cstring>
#include <fstream>
#include <complex>

#include "point3_t.h"
#include "../generic/array_t.h"
#include "../generic/matrix_t.h"

namespace core { namespace containers { namespace geometry {

// Структура трехмерный вектор

```

```

template<typename T>
struct vector3_t
{
    // Элементы
    T x, y, z;
    // Конструктор по-умолчанию
    vector3_t();
    // Конструктор из трех элементов
    template<typename U, typename V, typename R>
    vector3_t(U n_x, V n_y, R n_z);
    // Конструктор из точки
    template<typename U>
    vector3_t(const point3_t<U> & p);
    // Конструктор из двух точек - начала и конца
    template<typename U>
    vector3_t(const point3_t<U> & beg, const point3_t<U> & end);
    // Конструктор из другого вектора
    template<typename U>
    vector3_t(const vector3_t<U> & v);
    // Кастование к точке
    point3_t<T> to_point() const;
    // Операторы типа "скобка"
    T & operator [] (std::size_t i);
    const T & operator [] (std::size_t i) const;
    // Скалярное произведение
    T operator * (const vector3_t<T> & other) const;
    // Векторное произведение
    vector3_t<T> cross(const vector3_t<T> & other) const;
    // Получение сопряженного вектора
    vector3_t<T> conj() const;
    // Норма вектора
    double norm() const;
    // Квадрат нормы вектора
    double norm2() const;
    // Сложение и вычитание векторов
    vector3_t<T> operator + (const vector3_t<T> & other) const;
    vector3_t<T> operator - (const vector3_t<T> & other) const;
    vector3_t<T> & operator += (const vector3_t<T> & other);
    vector3_t<T> & operator -= (const vector3_t<T> & other);
    // Деление вектора на число
    vector3_t<T> operator / (const T & a) const;
    // Умножение вектора на число
    vector3_t<T> operator * (const T & a) const;
    vector3_t<T> & operator *= (const T & a);
    // Умножение числа на вектор
    template<typename U>
    friend vector3_t<U> operator * (const U & a, const vector3_t<U> & vec);
    // Умножение матрицы на вектор
    template<typename U, typename V>
    friend vector3_t<U> operator * (const generic::matrix_t<V, 3, 3> & matr, const vector3_t<U> & vec);
    // Вывод
    template<typename U>
    friend std::ostream & operator << (std::ostream & os, const vector3_t<U> & a)
    // Умножение числа на вектор
    template<typename U, typename V, typename R>
    friend R operator * (const U & a, const vector3_t<V> & vec);
    // Скалярное произведение
    template<typename U, typename V, typename R>
    friend R operator * (const vector3_t<U> & left, const vector3_t<V> & right);
    // Сложение и вычитание векторов
    template<typename U, typename V, typename R>
    friend R operator + (const vector3_t<U> & left, const vector3_t<V> & right);
    template<typename U, typename V, typename R>
    friend R operator - (const vector3_t<U> & left, const vector3_t<V> & right);
};

} } // namespace core::containers::geometry

#endif // CONTAINERS_GEOMETRY_VECTOR3_T_H_INCLUDED

```

Файл core/containers/geometry/point3_t.h

```
#if !defined(CONTAINERS_GEOMETRY_POINT3_T_H_INCLUDED)
#define CONTAINERS_GEOMETRY_POINT3_T_H_INCLUDED

#include <cstring>
#include <cassert>
#include <fstream>

namespace core { namespace containers { namespace geometry {

// Структура трехмерная точка
template<typename T>
struct point3_t
{
    // Координаты
    T x, y, z;
    // Номер точки
    std::size_t num;
    // Конструктор по-умолчанию
    point3_t();
    // Конструктор по трем координатам и номеру
    template<typename U>
    point3_t(U n_x, U n_y, U n_z, std::size_t n_num = 0);
    // Конструктор из другой точки
    template<typename U>
    point3_t(const point3_t<U> & p);
    // Операторы типа "скобка"
    T & operator [] (std::size_t i);
    const T & operator [] (std::size_t i) const;
    // Оператор меньше (по номеру)
    template<typename U>
    bool operator < (const point3_t<U> & t) const;
    // Оператор равенства (по номеру)
    template<typename U>
    bool operator == (const point3_t<U> & t) const;
    // Оператор присваивания
    point3_t<T> & operator = (const point3_t<T> & other);
    template<typename U>
    point3_t<T> & operator = (const point3_t<U> & other);
    // Вывод
    template<typename U>
    friend std::ostream & operator << (std::ostream & os, const point3_t<U> & a);
    // Проверка, лежит ли точка внутри параллелепипеда (для дерева)
    bool inside(double x0, double x1, double y0, double y1, double z0, double z1) const;
};

}} } // namespace core::containers::geometry

#endif // CONTAINERS_GEOMETRY_POINT3_T_H_INCLUDED
```

Файл core/cubatures/tetrahedron_integration.h

```
#if !defined(CUBATURES_TETRAHEDRON_INTEGRATION_H_INCLUDED)
#define CUBATURES_TETRAHEDRON_INTEGRATION_H_INCLUDED

#include <cstdint>
#include "../containers/generic/array_t.h"
#include "../containers/generic/matrix_t.h"

namespace core { namespace cubatures {

/**
 * @brief Класс для хранения точек интегрирования на тетраэдрах
 */
class tetrahedron_integration
{
public:
    tetrahedron_integration(std::size_t order = 8);

    /**
     * @brief Получить количество точек интегрирования
     * @return Количество точек интегрирования
     */
}
```

```

        */
    std::size_t get_gauss_num() const;

    /**
     * @brief Получить вес одной из точек интегрирования
     * @param[in] point_num
     * @return
     */
    double get_gauss_weight(std::size_t point_num) const;

    /**
     * @brief Получить одну из координат одной из точек интегрирования
     * @param[in] point_num Номер точки интегрирования
     * @param[in] coord_num Номер координаты интегрирования
     * @return Координата искомой точки интегрирования
     */
    double get_gauss_point_master(std::size_t point_num, std::size_t coord_num) const;

    /**
     * @brief Инициализация структуры для заданного порядка интегрирования
     * @param[in] order Порядок интегрирования
     */
    void init(std::size_t order);

private:

    /**
     * @brief Количество точек интегрирования
     */
    std::size_t m_gauss_num;

    /**
     * @brief Веса интегрирования
     */
    containers::generic::array_t<double> m_gauss_weights;

    /**
     * @brief Точки интегрирования на мастер-тетраэдре, в каждой строке по 4 L-координаты
     */
    containers::generic::matrix_t<double> m_gauss_points_master;

    /**
     * @brief Конвертер для внутренних нужд
     */
    void set(std::size_t num, const double weights[], const double points[][][4]);
};

} // namespace core::cubatures

#endif // CUBATURES_TETRAHEDRON_INTEGRATION_H_INCLUDED

```

Файл *core/cubatures/triangle_integration.h*

```

#ifndef !defined(CUBATURES_TRIANGLE_INTEGRATION_H_INCLUDED)
#define CUBATURES_TRIANGLE_INTEGRATION_H_INCLUDED

#include <cstddef>
#include <cassert>
#include "../containers/generic/array_t.h"
#include "../containers/generic/matrix_t.h"

namespace core { namespace cubatures {

    /**
     * @brief Класс для хранения точек интегрирования на треугольниках
     */
    class triangle_integration
    {
    public:
        triangle_integration(std::size_t order = 8);

        /**
         * @brief Получить количество точек интегрирования
         * @return Количество точек интегрирования
         */

```

```

        */
    std::size_t get_gauss_num() const;

    /**
     * @brief Получить вес одной из точек интегрирования
     * @param[in] point_num
     * @return
     */
    double get_gauss_weight(std::size_t point_num) const;

    /**
     * @brief Получить одну из координат одной из точек интегрирования
     * @param[in] point_num Номер точки интегрирования
     * @param[in] coord_num Номер координаты интегрирования
     * @return Координата искомой точки интегрирования
     */
    double get_gauss_point_master(std::size_t point_num, std::size_t coord_num) const;

    /**
     * @brief Инициализация структуры для заданного порядка интегрирования
     * @param[in] order Порядок интегрирования
     */
    void init(std::size_t order);

private:

    /**
     * @brief Количество точек интегрирования
     */
    std::size_t m_gauss_num;

    /**
     * @brief Веса интегрирования
     */
    containers::generic::array_t<double> m_gauss_weights;

    /**
     * @brief Точки интегрирования на мастер-треугольнике, в каждой строке по 3 L-координаты
     */
    containers::generic::matrix_t<double> m_gauss_points_master;

    /**
     * @brief Конвертер для внутренних нужд
     */
    void set(std::size_t num, const double weights[], const double points[][3]);
};

} // namespace core::cubatures

#endif // CUBATURES_TRIANGLE_INTEGRATION_H_INCLUDED

```

Файл *core/solvers/CSLR/preconditioners/preconditioner_interface.h*

```

#ifndef !defined(SOLVERS_CSLR_PRECONDITIONERS_INTERFACE_H_INCLUDED)
#define SOLVERS_CSLR_PRECONDITIONERS_INTERFACE_H_INCLUDED

#include <cstddef>
#include <string>

namespace core { namespace solvers { namespace CSLR { namespace preconditioners {

    /**
     * @brief Интерфейс для предобуславливателей вида  $A = S * Q$ 
     */
    template<typename val_type, typename ind_type = std::size_t>
    class preconditioner_interface
    {
public:
    /**
     * @brief Конструктор для несимметричного предобуславливателя
     * @param[in] gi Массив ig в разреженном строчно-столбцовом представлении
     * @param[in] gj Массив jg в разреженном строчно-столбцовом представлении
     * @param[in] di Массив диагональных эл-тов в разреженном строчно-столбцовом представлении
     * @param[in] gl Массив нижнего треугольника в разреженном строчно-столбцовом представлении
     */

```

```

* @param[in] gu Массив верхнего треугольника в разреженном строчно-столбцовом представлении
* @param[in] n Размерность матрицы
*/
preconditioner_interface(const ind_type * gi, const ind_type * gj, const val_type * di,
const val_type * gl, const val_type * gu, ind_type n);

/**
* @brief Конструктор для симметричного предобуславливателя
* @param[in] gi Массив ig в разреженном строчно-столбцовом представлении
* @param[in] gj Массив jg в разреженном строчно-столбцовом представлении
* @param[in] di Массив диагональных эл-тов в разреженном строчно-столбцовом представлении
* @param[in] gg Массив одного из треугольников в разреженном строчно-столбцовом представлении
* @param[in] n Размерность матрицы
*/
preconditioner_interface(const ind_type * gi, const ind_type * gj, const val_type * di,
const val_type * gg, ind_type n);

/**
* @brief Узнать название предобуславливателя
* @return Название предобуславливателя
*/
virtual std::string get_name() const = 0;

/**
* @brief Решает СЛАУ вида  $x = S^{-1} * f$ 
* @param[in] f
* @param[out] x
*/
virtual void solve_S(const val_type * f, val_type * x) const = 0;

/**
* @brief Решает СЛАУ вида  $x = S^{-T} * f$ 
* @param[in] f
* @param[out] x
*/
virtual void solve_ST(const val_type * f, val_type * x) const = 0;

/**
* @brief Решает СЛАУ вида  $x = Q^{-1} * f$ 
* @param[in] f
* @param[out] x
*/
virtual void solve_Q(const val_type * f, val_type * x) const = 0;

/**
* @brief Решает СЛАУ вида  $x = Q^{-T} * f$ 
* @param[in] f
* @param[out] x
*/
virtual void solve_QT(const val_type * f, val_type * x) const = 0;

/**
* @brief Вычисляет  $x = Q * f$ 
* @param[in] f
* @param[out] x
*/
virtual void mul_Q(const val_type * f, val_type * x) const = 0;

/**
* @brief Виртуальный деструктор
*/
virtual ~preconditioner_interface();

protected:
preconditioner_interface();

private:
preconditioner_interface(const preconditioner_interface & other);
const preconditioner_interface & operator = (const preconditioner_interface & other);
};

}}}} // namespace core::solvers::CSLR::preconditioners

#endif // SOLVERS_CSLR_PRECONDITIONERS_INTERFACE_H_INCLUDED

```

Файл core/solvers/CSLR/symmetric/symmetric_solver_interface.h

```
#if !defined(SOLVERS_SYMMETRIC_INTERFACE_H_INCLUDED)
#define SOLVERS_SYMMETRIC_INTERFACE_H_INCLUDED

#include <cstdlib>

#include "../preconditioners/preconditioner_interface.h"

namespace core { namespace solvers { namespace CSLR { namespace symmetric {

/***
 * @brief Интерфейс для симметричных решателей
 */
template<typename val_type, typename ind_type = std::size_t>
class symmetric_solver_interface
{
public:

    /**
     * @brief Инициализация симметричной
     * @param[in] gi Массив ig в разреженном строчно-столбцовом представлении
     * @param[in] gj Массив gj в разреженном строчно-столбцовом представлении
     * @param[in] di Массив диагональных эл-тов в разреженном строчно-столбцовом представлении
     * @param[in] gg Массив нодного из треугольников в разреженном строчно-столбцовом представлении
     * @param[in] n Размерность матрицы
     * @param[in] precond Предобуславливатель
     */
    virtual void init(const ind_type * gi, const ind_type * gj, const val_type * di,
                      const val_type * gg, ind_type n,
                      preconditioners::preconditioner_interface<val_type, ind_type> * precond) = 0;

    /**
     * @brief Запуск решения
     * @param[inout] solution На вход идет начальное приближение, на выход - решение
     * @param[in] rp Правая часть
     * @param[in] eps Целевая относительная невязка
     * @param[in] max_iter Максимальное количество итераций
     */
    virtual void solve(val_type * solution, const val_type * rp,
                      double eps, ind_type max_iter) = 0;

    /**
     * @brief Виртуальный деструктор
     */
    virtual ~symmetric_solver_interface();
};

}}} // namespace core::solvers::CSLR::symmetric

#endif // SOLVERS_SYMMETRIC_INTERFACE_H_INCLUDED
```

Файл core/solvers/CSLR/symmetric/solvers_factory.h

```
#if !defined(SOLVERS_CSLR_FACTORY_H_INCLUDED)
#define SOLVERS_CSLR_FACTORY_H_INCLUDED

#include <complex>
#include <string>

#include "preconditioners/preconditioner_interface.h"
#include "symmetric/symmetric_solver_interface.h"

namespace core { namespace solvers { namespace CSLR { namespace factory {

/***
 * @brief Функция, создающая новый симметричный комплексный предобуславливатель
 * @param[in] name Название предобуславливателя
 * @param[in] gi Массив ig в разреженном строчно-столбцовом представлении
 * @param[in] gj Массив gj в разреженном строчно-столбцовом представлении
 * @param[in] di Массив диагональных эл-тов в разреженном строчно-столбцовом представлении
 * @param[in] gg Массив одного из треугольников в разреженном строчно-столбцовом представлении
 * @param[in] n Размерность матрицы
 */
```

```

    * @return Новый симметричный комплексный предобуславливатель (удаляет вызывающая сторона)
    */
preconditioners::preconditioner_interface<std::complex<double>, std::size_t> *
create_symmetric_complex_preconditioner(
    const std::string & name, const std::size_t * gi, const std::size_t * gj,
    const std::complex<double> * di, const std::complex<double> * gg, std::size_t n);

/**
 * @brief Функция, создающая новый симметричный комплексный решатель
 * @param[in] name Название решателя
 * @param[in] gi Массив ig в разреженном строчно-столбцовом представлении
 * @param[in] gj Массив jg в разреженном строчно-столбцовом представлении
 * @param[in] di Массив диагональных эл-тов в разреженном строчно-столбцовом представлении
 * @param[in] gg Массив одного из треугольников в разреженном строчно-столбцовом представлении
 * @param[in] n Размерность матрицы
 * @param[in] precond Предобуславливатель
 * @return Новый симметричный комплексный решатель (удаляет вызывающая сторона)
*/
symmetric::symmetric_solver_interface<std::complex<double>, std::size_t> *
create_symmetric_complex_solver(
    const std::string & name, const std::size_t * gi, const std::size_t * gj,
    const std::complex<double> * di, const std::complex<double> * gg, std::size_t n,
    preconditioners::preconditioner_interface<std::complex<double>, std::size_t> * precond);

}}} // namespace core::solvers::CSLR::factory

#endif // SOLVERS_CSLR_FACTORY_H_INCLUDED

```

Файл core/utils/inifile.h

```

#ifndef !defined(UTILS_INIFILE_H_INCLUDED)
#define UTILS_INIFILE_H_INCLUDED

#include <list>
#include <map>
#include <string>
#include <sstream>
#include <iostream>

#include "strings.h"

namespace core { namespace utils {

/**
 * @brief Класс для работы с ini-файлами
 */
class inifile
{
public:
    /**
     * @brief Конструктор по-умолчанию
     */
    inifile();

    /**
     * @brief Конструктор с именем входного файла
     * @param[in] filename имя входного файла
     */
    inifile(const std::string & filename);

    /**
     * @brief Загрузка ini-файла
     * @param[in] filename имя входного файла
     * @return true - файл успешно считан, false - возникли ошибки
     */
    bool load(const std::string & filename);

    /**
     * @brief Статус, разобран ли ini-файл
     * @return true - файл успешно разобран, false - возникли ошибки
     */
    bool good() const;

}

```

```

* @brief Получить значение параметра с именем "parameter" в секции "section" подсекции "subsection"
* @param[in] section секция, в которой нужно искать
* @param[in] subsection подсекция, в которой нужно искать
* @param[in] parameter параметр, который нужно искать
* @param[in] fallback значение по-умолчанию, которое возвращается, если искомый параметр не найден
* @return значение параметра или fallback, если параметр не найден
* @note Функция для типа string
*/
template<typename U>
std::string get(const std::string & section, const U & subsection, const std::string & parameter,
               const std::string & fallback) const;

/**
* @brief Получить значение параметра с именем "parameter" в секции "section" подсекции "subsection"
* @param[in] section секция, в которой нужно искать
* @param[in] subsection подсекция, в которой нужно искать
* @param[in] parameter параметр, который нужно искать
* @param[in] fallback значение по-умолчанию, которое возвращается, если искомый параметр не найден
* @return значение параметра или fallback, если параметр не найден
* @note Функция для типа char *, его приводим к std::string
*/
template<typename U>
std::string get(const std::string & section, const U & subsection, const std::string & parameter,
               const char * fallback) const;

/**
* @brief Получить значение параметра с именем "parameter" в секции "section" подсекции "subsection"
* @param[in] section секция, в которой нужно искать
* @param[in] subsection подсекция, в которой нужно искать
* @param[in] parameter параметр, который нужно искать
* @param[in] fallback значение по-умолчанию, которое возвращается, если искомый параметр не найден
* @return значение параметра или fallback, если параметр не найден
* @note Функция для типа bool, нужно сделать дополнительное преобразование yes|true|1 -> true
*/
template<typename U>
bool get(const std::string & section, const U & subsection, const std::string & parameter,
         const bool & fallback) const;

/**
* @brief Получить значение параметра с именем "parameter" в секции "section" подсекции "subsection"
* @param[in] section секция, в которой нужно искать
* @param[in] subsection подсекция, в которой нужно искать
* @param[in] parameter параметр, который нужно искать
* @param[in] fallback значение по-умолчанию, которое возвращается, если искомый параметр не найден
* @return значение параметра или fallback, если параметр не найден
* @note Обобщенная функция для типов, которые не являются типами bool или string
*/
template<typename T, typename U>
T get(const std::string & section, const U & subsection, const std::string & parameter,
      const T & fallback) const;

/**
* @brief Получить список из всех подсекций секции "section"
* @param[in] section секция, в которой нужно искать подсекции
* @param[in] type фиктивный указатель, из которого определяется тип возвращаемого результата
* @return список из всех подсекций секции "section"
* @note type используется как флаг типа, передавать в него нужно, например (size_t*) (NULL)
*/
template<typename T>
std::list<T> enumerate(const std::string & section, const T * type) const;

/**
* @brief Проверить, что в файле нет никаких секций, кроме секций из списка "whitelist"
* @param[in] whitelist список разрешенных секций
* @return true - все секции присутствуют в "whitelist", false - есть неразрешенная секция
* @note Полезно для контроля опечаток, так как в остальных местах это штатная ситуация
*/
bool check_sections(const std::list<std::string> & whitelist) const;

/**
* @brief Проверить, что во всех подсекциях секции нет параметров не из белого списка
* @param[in] section секция, для которой будет выполнена проверка
* @param[in] whitelist список разрешенных параметров
* @return true - все параметры присутствуют в "whitelist", false - есть неразрешенный параметр
* @note Полезно для контроля опечаток, так как в остальных местах это штатная ситуация
*/

```

```

bool check_parameters(const std::string & section, const std::list<std::string> & whitelist) const;
protected:

    /**
     * @brief Костыльный контейнер подсекции для исправления багов MSVC
     * @note https://msdn.microsoft.com/en-us/library/074af4b6.aspx
     */
    struct subsection_t
    {
        std::map<std::string, std::string> _;
    };

    /**
     * @brief Костыльный контейнер секции для исправления багов MSVC
     * @note https://msdn.microsoft.com/en-us/library/074af4b6.aspx
     */
    struct section_t
    {
        std::map<std::string, subsection_t> _;
    };

    /**
     * @brief Статус, разобран ли входной конфиг
     */
    bool m_status;

    /**
     * @brief Конетейнер значений параметров, сгруппированных в подсекции, сгруппированные в секции
     */
    std::map<std::string, section_t> m_values;

    /**
     * @brief Получить значение параметра с именем "parameter" в секции "section" подсекции "subsection"
     * @param[in] section секция
     * @param[in] subsection подсекция
     * @param[in] parameter параметр
     * @return значение параметра или NULL, если такого параметра нет
     */
    const std::string * get_internal(const std::string & section, const std::string & subsection,
                                    const std::string & parameter) const;

    /**
     * @brief Преобразование подсекции в строковое представление
     * @param[in] subsection подсекция
     * @return строковое представление подсекции
     * @note Если это строка, то она уже итак задана как надо
     */
    std::string subsection_to_str(const std::string & subsection) const;

    /**
     * @brief Преобразование подсекции в строковое представление
     * @param[in] subsection подсекция
     * @return строковое представление подсекции
     * @note Если это char *, то просто соберем из него строку
     */
    std::string subsection_to_str(const char * subsection) const;

    /**
     * @brief Преобразование подсекции в строковое представление
     * @param[in] subsection подсекция
     * @return строковое представление подсекции
     */
    template<typename T>
    std::string subsection_to_str(const T & subsection) const;
};

} } // namespace core::utils

#endif // UTILS_INIFILE_H_INCLUDED

```

Файл core/evaluator/evaluator.h

```
#if !defined(EVALUATOR_H)
#define EVALUATOR_H

/*
// This is the module to parse, simplify and evaluate expressions with support of a JIT-compilation
//
// Usage example:
//
// #include "evaluator/evaluator.h"
// ...
// evaluator<double> p;
// if(!p.parse("exp(-(0.5-x)*(0.5-x)-(0.5-z)*(0.5-z))"))
//     cerr << p.get_error() << endl;
// else if(!p.simplify()) // simplify is optional step
//     cerr << p.get_error() << endl;
// else
// {
//     if(!p.compile()) // compile is optional step
//         cerr << p.get_error() << endl;
//     p.set_var("x", 0.4);
//     p.set_var("z", 0.8);
//     double result;
//     if(!p.calculate(result))
//         cerr << p.get_error() << endl;
//     else
//         cout << result << endl;
// }
*/
#include "evaluator_operations.h"
#include "evaluator_internal/evaluator_object.h"
#include "evaluator_internal/var_container.h"
#include "evaluator_internal/transition_table.h"
#include "evaluator_internal/jit/common.h"
#include "evaluator_internal/jit/opcodes.h"
#include "evaluator_internal/jit/func_templates.h"
#include "evaluator_internal/jit/oper_templates.h"

#include <vector>
#include <map>
#include <string>
#include <utility>
#include <algorithm>

// Evaluator, main classs
template<typename T> class evaluator
{
public:

    // Function type in evaluator_object
    typedef T(* func_type)(const T &);

    // Operator type in evaluator_object
    typedef T(* oper_type)(const T &, const T &);

protected:

    // State transition table
    std::vector<evaluator_internal::transition_table_record> m_transition_table;
    // Expression, Reverse Polish notation
    std::vector<evaluator_internal::evaluator_object<T>> m_expression;
    // Container: [function name]->function pointer
    std::map<std::string, func_type> m_functions;
    // Container: [variable name]->pointer to var_container
    std::map<std::string, evaluator_internal::var_container<T>> m_variables;
    // Container: [constant name]->constant value
    std::map<std::string, T> m_constants;
    // Container: [operator name]->pair(priority, operator pointer)
    std::map<char, std::pair<unsigned short int, oper_type>> m_operators;
    // Current parsing status: true is good, false is bad
    bool m_status;
    // Error description if m_status == false
    std::string m_error_string;
```

```

// Current compiling status: true is compiled, false is not compiled
bool m_is_compiled;
#if !defined(EVALUATOR_JIT_DISABLE)
// Executable memory for bytecode
char * volatile m_jit_code;
// Function pointer to same memory
void(EVALUATOR_JIT_CALL * m_jit_func)();
// Size of allocated executable memory
size_t m_jit_code_size;
// Memory for stack, used in compiled code
T * volatile m_jit_stack;
// Size of allocated memory for stack
size_t m_jit_stack_size;
#endif

// Return incorrect big number (uninitialized variable)
template<typename U> T incorrect_number(const std::complex<U> &) const;
template<typename U> T incorrect_number(const U &) const;
// Check whether a number is incorrect (uninitialized variable)
template<typename U> bool is_incorrect(const std::complex<U> & val) const;
template<typename U> bool is_incorrect(const U & val) const;

// Primary initialization
void init();
// Copying from another evaluator
void copy_from_other(const evaluator & other);

public:

// Constructors and destructor
evaluator(const evaluator & other);
evaluator();
evaluator(const std::string & str);
~evaluator();
// Copying from another evaluator
const evaluator & operator = (const evaluator & other);

// Get error description
const std::string & get_error() const;

// Get current parsing status
bool is_parsed() const;

// Set new value 'value' for variable with name 'name'
void set_var(const std::string & name, const T & value);

// Reset all variables
void reset_vars();
// Parse string 'str'
bool parse(const std::string & str);
// Simplify current expression
bool simplify();
// Calculate current expression and write result to 'result'
bool calculate(T & result);

// Get current compiling status
bool is_compiled() const;

// Compile expression, all functions will be inlined
bool compile_inline();
// Compile expression, all functions will be called from 'functions' and 'operators' containers
bool compile_extcall();
// Compile expression, default
bool compile();
};

#include "evaluator_internal/misc.h"
#include "evaluator_internal/parse.h"
#include "evaluator_internal/simplify.h"
#include "evaluator_internal/calculate.h"
#include "evaluator_internal/jit/compile_inline.h"
#include "evaluator_internal/jit/compile_extcall.h"

#endif // EVALUATOR_H

```

Файл sources/src/core/evaluator/evaluator_operations.h

```
#if !defined(EVALUATOR_OPERATIONS_H)
#define EVALUATOR_OPERATIONS_H

#include <cmath>
#include <complex>
#include <utility>
#include <map>
#include <string>
#include "evaluator_internal/var_container.h"
#include "evaluator_internal/type_detection.h"

namespace evaluator_internal
{
    // All fuctions (must NOT be inline).
    template<typename T> T eval_sin(const T & ar);
    template<typename T> T eval_cos(const T & ar);
    template<typename T> T eval_tan(const T & ar);
    template<typename T> T eval_sinh(const T & ar);
    template<typename T> T eval_cosh(const T & ar);
    template<typename T> T eval_tanh(const T & ar);
    template<typename T> T eval_log(const T & ar);
    template<typename T> T eval_log10(const T & ar);
    template<typename T> T eval_exp(const T & ar);
    template<typename T> T eval_sqrt(const T & ar);
    template<typename T> T eval_log2(const T & ar);
    template<typename T> T eval_asinh(const T & ar);
    template<typename T> T eval_acosh(const T & ar);
    template<typename T> T eval_atanh(const T & ar);
    template<typename T> std::complex<T> eval_abs(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_asin(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_acos(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_atan(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_imag(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_real(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_conj(const std::complex<T> & ar);
    template<typename T> std::complex<T> eval_arg(const std::complex<T> & ar);
    template<typename T> T eval_abs(const T & ar);
    template<typename T> T eval_asin(const T & ar);
    template<typename T> T eval_acos(const T & ar);
    template<typename T> T eval_atan(const T & ar);
    template<typename T> T eval_imag(const T & ar);
    template<typename T> T eval_real(const T & ar);
    template<typename T> T eval_conj(const T & ar);
    template<typename T> T eval_arg(const T & ar);

    // Add function pointers into container.
    template<typename T>
    void init_functions(std::map<std::string, T(*)> & funcs_map);

    // All operators (must NOT be inline).
    template<typename T> T eval_plus(const T & larg, const T & rarg);
    template<typename T> T eval_minus(const T & larg, const T & rarg);
    template<typename T> T eval_mult(const T & larg, const T & rarg);
    template<typename T> T eval_div(const T & larg, const T & rarg);
    template<typename T> T eval_pow(const T & larg, const T & rarg);

    // Add operators pointers into container.
    template<typename T>
    void init_operators(std::map<char, std::pair<unsigned short int,
                                                T(*)>> & oper_map);

    // Init default constant values.
    template<typename T>
    void init_constants(std::map<std::string, T> & consts_map);

} // namespace evaluator_internal
#endif // EVALUATOR_OPERATIONS_H
```

Файл sources/src/core/evaluator/evaluator_xyz.h

```
#if !defined(EVALUATOR_XYZ_H)
#define EVALUATOR_XYZ_H

/*
// Usage example:
//
// #include "evaluator/evaluator_xyz.h"
// ...
// evaluator_xyz<double> e;
// if(!e.parse("exp(-(0.5-x)*(0.5-z)*(0.5-z))"))
//     cerr << e.get_error() << endl;
// else if(!e.simplify()) // simplify is optional step
//     cerr << e.get_error() << endl;
// else
// {
//     if(!e.compile()) // compile is optional step
//         cerr << e.get_error() << endl;
//     e.set_x(0.4);
//     e.set_z(0.8);
//     double result;
//     if(!e.calculate(result))
//         cerr << e.get_error() << endl;
//     else
//         cout << result << endl;
// }
*/
#include "evaluator.h"

// Reference example: Evaluator(x,y,z), takes O(1) time for setting x, y or z
template<typename T> class evaluator_xyz : public evaluator<T>
{
public:

    // Constructors
    evaluator_xyz();
    evaluator_xyz(const std::string & str);
    evaluator_xyz(const evaluator_xyz & other);

    // Copying from another evaluator
    const evaluator_xyz & operator = (const evaluator_xyz & other);

    // Set new value 'x' for variable with name 'x'
    void set_x(const T & x);

    // Set new value 'y' for variable with name 'y'
    void set_y(const T & y);

    // Set new value 'z' for variable with name 'z'
    void set_z(const T & z);

private:

    // Cached pointers for x, y and z
    T * m_x, * m_y, * m_z;

    // Update cached pointers
    void update_cache();
};

#endif // EVALUATOR_XYZ_H
```