МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

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

_____
(подпись)

«_____» _____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Леонович Дарьяны Александровны
(фамилия, имя, отчество студента – автора работы)

Разработка и программная реализация вычислительной схемы численного моделирования многофазных течений в задачах нефтедобычи с использованием горизонтальных скважин
(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

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

Руководитель от НГТУ	**Автор выпускной квалификационной работы**
Персова М.Г. (фамилия, имя, отчество)	Леонович Д.А. (фамилия, И.О.)
д.т.н., профессор (ученая степень, ученое звание)	ФПМИ, ПМ-83 (факультет, группа)
(подпись, дата)	(подпись, дата)

Новосибирск, 2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

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

«21» марта 2022 г..

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Леонович Дарьяне Александровне***
(фамилия, имя, отчество студента)

Направление подготовки *01.03.02. Прикладная математика и информатика*

Факультет Прикладной математики и информатики

Тема ***Разработка и программная реализация вычислительной схемы численного моделирования многофазных течений в задачах нефтедобычи с использованием горизонтальных скважин***

Исходные данные (или цель работы):

Разработка и программная реализация вычислительной схемы численного моделирования многофазных течений в задачах нефтедобычи с использованием горизонтальных скважин

Структурные части работы:

1. Рассмотрение математической модели многофазной фильтрации с учетом зависимости вязкости флюида от температуры.

2. Разработка алгоритма встройки горизонтальных скважин в конечноэлементную сетку.

3. Рассмотрение вычислительной схемы расчета давления и обновления фазового состояния ячеек конечноэлементной сетки.

4. Программная реализация рассмотренных алгоритмов.

5. Верификация разработанной программы.

6. Исследование разных методов встройки горизонтальных скважин в конечноэлементную сетку.

6. Исследование влияния температуры закачиваемой смеси на отбор нефти при численном моделировании многофазной фильтрации с использованием технологии горизонтальных скважин.

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Персова М.Г.	*Леонович Д.А.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
д.т.н., профессор	*ФПМИ, ПМ-83*
(ученая степень, ученое звание)	(факультет, группа)
21.03.2022 г.	*21.03.2022 г.*
(подпись, дата)	(подпись, дата)

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

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

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

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

АННОТАЦИЯ

Отчет 90 с., 18 рис., 6 табл., 14 источников, 1 прил.

МЕТОД КОНЕЧНЫХ ЭЛЕМЕНТОВ, МНОГОФАЗНАЯ ФИЛЬТРАЦИЯ, ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ, ТЕПЛОВЫЕ МЕТОДЫ УВЕЛИЧЕНИЯ НЕФТЕОТДАЧИ

Целью работы является разработка и программная реализация вычислительной схемы численного моделирования многофазных течений в задачах нефтедобычи с использованием горизонтальных скважин.

Для решения задачи фильтрации используется метод конечных элементов (трилинейные базисные функции на шестигранниках). Для моделирования применения технологий горизонтальных скважин при разработке нефтяных месторождений реализованы и сравнены два способа встройки скважин в конечноэлементную сетку. Проведены исследования влияния тепловых методов увеличения нефтеотдачи на пласт.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Высокие объемы добычи нефти на протяжении последних десятилетий приводят к тому, что первичные и вторичные методы нефтедобычи на ряде месторождений становятся неэффективны. Это может происходить, например, из-за снижения давления в коллекторе, вымывания большей части подвижной нефти или слишком высокой вязкости нефти, оставшейся в коллекторе. Поэтому разрабатываются и развиваются различные третичные методы нефтедобычи. А вследствие высокой стоимости и продолжительности во времени разработки месторождений для анализа и сравнения эффективности разных методов увеличения нефтеотдачи (МУН) часто применяется численное моделирование, позволяющее сократить затраты на эксперименты или проанализировать пользу того или иного МУН для конкретного месторождения [1, 2, 3].

Одними из самых распространенных третичных методов нефтедобычи являются тепловые: закачка горячего пара или воды, технология парогравитационного дренажа (или SAGD) и другие методы [4]. Повышение температуры в коллекторе приводит к снижению вязкости нефти и, следовательно, к увеличению доли нефти в добываемой смеси.

В связи с исследованием и применением такого метода увеличения нефтеотдачи расширяется и область применения технологии горизонтальных скважин при разработке углеводородных месторождений. Такая конфигурация, благодаря большей площади соприкосновения скважин с нефтеносным слоем, позволяет значительно увеличить размер зоны перфорации, а значит увеличить объемы добываемой или закачиваемой смеси. На практике технология горизонтальных скважин особенно хорошо проявила себя в паре с тепловыми методами увеличения нефтеотдачи. Становится актуальным моделирование процессов нефтедобычи с использованием технологии горизонтальных скважин.

1. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ И ЧИСЛЕННЫЙ МЕТОД

1.1 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ МНОГОФАЗНЫХ ТЕЧЕНИЙ В ЗАДАЧАХ НЕФТЕДОБЫЧИ

Расчетная область в поставленной задаче является пористой средой, характеризующейся структурной пористостью породы Ф и структурной проницаемостью породы **К**. Пористость – доля пространства в объёме породы, доступного для движения смеси. Структурная проницаемость в общем случае тензор, характеризующий способность породы пропускать смесь в определенном направлении, измеряется в мкм^2. В данной работе будем считать пористость и проницаемость постоянными во всей расчетной области. Так же будем полагать, что в любой момент времени все пространство среды заполнено смесью.

Под фазой будем понимать жидкость, входящую в состав фильтруемой смеси и обладающую своими фильтрационными свойствами. К свойствам фазы относятся: плотность фазы (ρ, кг/м^3), динамическая вязкость (η, Па*с) и множитель структурной проницаемости (k, зависит от насыщенности фазы в среде, безразмерная величина). Насыщенность фазы в среде S характеризует долю фазы в смеси, распределенной по расчетной области.

Для моделирования фильтрации производится расчет движения смеси в среде из-за перепада давления. Скорость фильтрации многофазного потока описывается законом Дарси:

$$\vec{u} = -\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P + P_c^{ph}\right), \qquad (1)$$

где $\vec{u}$ – скорость фильтрации, PH – количество фаз, P – давление и P_c^{ph} – капиллярное давление фазы ph. Записав для каждой фазы закон сохранения массы, получим систему уравнений:

$$-\mathrm{div}\left(\rho^{ph}\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P+P_c^{ph}\right)\right)=\frac{\partial}{\partial t}(\Phi\rho^{ph}S^{ph})+f,\qquad ph=\overline{1,PH}$$

где f – отбор или закачка фазы в области.

С учетом предположения, что фазы несжимаемые (т.е. ρ^{ph} – постоянные величины), правые и левые части системы уравнений можно разделить на плотность соответствующих фаз и сложить. Тогда $\frac{\partial}{\partial t}\left(\sum_{ph=1}^{PH}S^{ph}\right)=0$, т.к. сумма насыщенностей всех фаз равна 1. В итоге получаем эллиптическую краевую задачу для давления:

$$-\mathrm{div}\left(\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P+P_c^{ph}\right)\right)=0, \tag{2}$$

$$P|_{S_1}=P_g, \tag{3}$$

$$\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P+P_c^{ph}\right)\Big|_{S_2}=\theta, \tag{4}$$

где S_1 – удаленные границы, где сохраняется пластовое давление, и S_2 – непроницаемые границы с нулевым потоком и границы скважин, через которые осуществляется отбор или закачка смеси.

Решив задачу и получив распределение поля давления, мы сможем вычислить объемы смеси и каждой фазы в отдельности, перетекающие через грани сетки и рассчитать новые объемы каждой фазы на элементах. Зная их, вычисляем новые насыщенности и температуры фаз. В следствие изменения температуры на элементе может измениться и значение вязкости фаз, так как оно зависит от температуры.

После пересчета состояния ячеек, изменим множитель структурной проницаемости фазы и вязкость согласно заданным зависимостям k^{ph} от насыщенности фазы и η^{ph} от температуры смеси.

Теперь рассмотрим подробнее каждый из алгоритмов.

1.2 АЛГОРИТМ ПОСТРОЕНИЯ КОНЕЧНОЭЛЕМЕНТНОЙ СЕТКИ С МОДЕЛИРОВАНИЕМ ИСПОЛЬЗОВАНИЯ ГОРИЗОНТАЛЬНЫХ СКВАЖИН

Для численного моделирования процесса многофазной фильтрации будет использоваться метод конечных элементов, для применения которого потребуется построить конечноэлементную сетку, аппроксимирующую расчетную область. В данной работе будем использовать согласованные трехмерные сетки из шестигранников. Для моделирования горизонтальных скважин будем модифицировать базовую сетку, чтобы встроить объект, аппроксимирующий скважину, и тем самым сгустить сетку рядом со скважинами. Более подробная сетка вблизи скважины позволит точнее учесть резкое изменение поля рядом с источником при решении краевой задачи для поиска давления. Для встройки скважин были рассмотрены два алгоритма.

Алгоритм заключается в следующем. Собирается базовая сетка на всю расчетную область (без учета скважин). Для генерации используются введенные пользователем данные о координатах начала и конца сетки по каждой из осей, размер бака (расстояние от границы интересующей области до удаленной границы, где считаем давление неизменяемым или поток нулевым), шаг в интересующей области, шаг и коэффициент разрядки в баке (коэффициент равный 1 для регулярных сеток).

Для построения скважин задаются параметры скважины: радиус, начало и конец скважины, границы зон перфорации. Также задаются следующие параметры сетки: коэффициент разрядки (для сгущения сетки вблизи скважины; может быть разным для горизонтальной и вертикальной координаты), количество дроблений между скважиной и базовой сеткой, коэффициент, определяющий во сколько раз область встройки скважины должна быть больше радиуса скважины (может быть разным для горизонтальной и вертикальной координаты). После чего скважина встраивается в базовую сетку.

Далее в зависимости от способа встройки скважин действия различаются. Во варианте с явной встройкой скважины в базовой сетке находятся все элементы, центры которых попадают в область встройки скважины, и удаляются из сетки. После чего строятся новые элементы: от каждого узла по краю вырезанной области строится прямая до центральной оси скважины. На этих прямых откладываются узлы для разбиения встраиваемой области на более мелкие ячейки. После чего эти узлы последовательно обходятся и объединяются в конечные элементы сетки. Параметры пористой среды (проницаемость, пористость и насыщенности фаз) присваиваются новому элементу в зависимости от того, на какой удаленный элемент из базовой сетки приходится центр нового конечного элемента. При этом в область встраивается вся скважина, часть граней которой образует заданные зоны перфорации.

Пример этого способа встройки скважины параллельно оси X представлен на рисунках 1-2. На рисунке 1 представлено поперечное сечение скважины в плоскости YZ, красным выделена область встройки скважины. На рисунке 2 представлено продольное сечение по центру скважины в плоскости XZ, красным выделена область встройки скважины в районе зоны перфорации. Правее и левее зоны перфорации вся скважины встроена аналогичным образом.

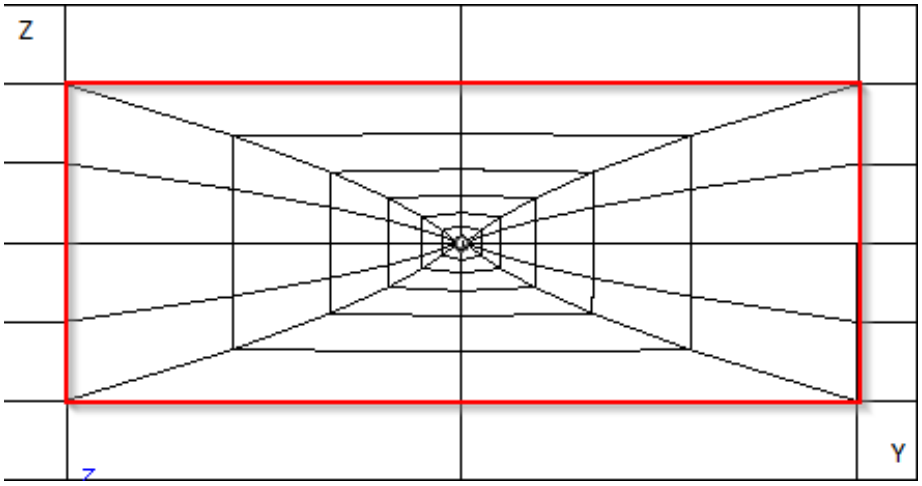

Рисунок 1 – Пример построения конечноэлементной сетки с помощью явной встройки скважины (плоскость YZ)

Рисунок 2 – Пример построения конечноэлементной сетки с помощью явной встройки скважины (плоскость XZ)

Во втором способе, при неявной встройке горизонтальных скважин в конечноэлементную сетку, зоны перфорации скважин задаются с помощью маленьких (в соответствии с радиусом скважин) ячеек. При этом чтобы избежать избыточного дробления всей сетки, используется технология несогласованных сеток. При численном решении задачи фильтрации элементы, приходящиеся непосредственно на «внутреннюю» часть скважин, считаются непроницаемым, т.е. их фазовое состояние не пересчитывается на каждом временном шаге.

Алгоритм заключается в том, что в области встройки скважины только для зон перфорации, а не для всей скважины, строится сетка из параллелепипедов – в центре элемент, равный по размеру диаметру скважины, вокруг элементы, построенные в соответствии с заданным для области встройки скважины коэффициентом разрядки. При этом существовавшие до встройки узлы и ребра не удаляются из базовой сетки.

Пример этого способа встройки скважины параллельно оси X представлен на рисунках 3-4. На рисунке 3 представлено поперечное сечение скважины в плоскости YZ, красным выделена область встройки скважины. На рисунке 4 представлено продольное сечение по центру скважины в плоскости XZ, красным выделена область встройки скважины в районе зоны перфорации. Можно увидеть, что левее и правее зоны перфорации встраивается несколько элементов, но не по всей длине скважины.

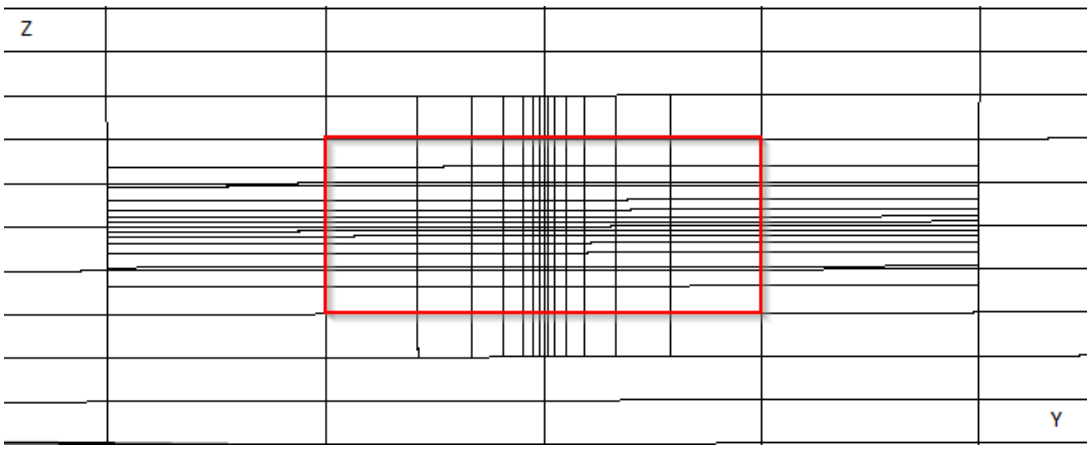

Рисунок 3 – Пример построения конечноэлементной сетки с помощью неявной встройки скважины (плоскость YZ)

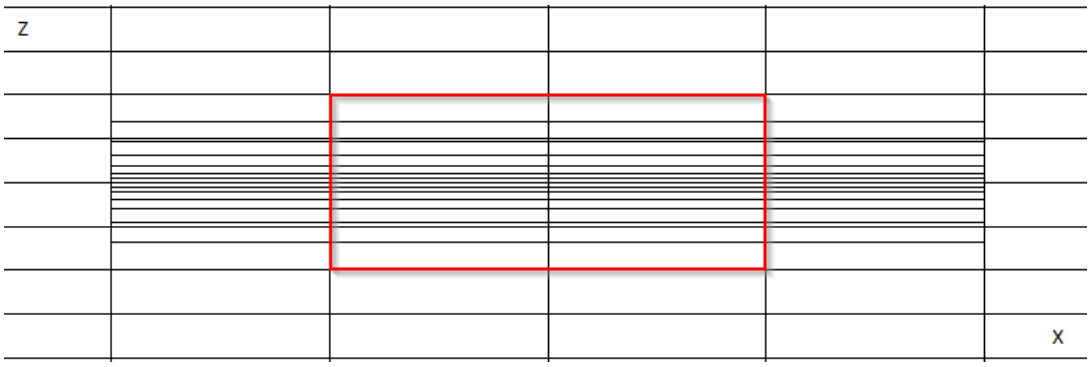

Рисунок 4 – Пример построения конечноэлементной сетки с помощью неявной встройки скважины (плоскость XZ)

1.3 ВЫЧИСЛИТЕЛЬНАЯ СХЕМА РАСЧЕТА ПОЛЯ ДАВЛЕНИЯ

Для расчета поля давления решается эллиптическая краевая задача (2)-(4) методом конечных элементов.

Запишем эквивалентную исходной задаче вариационную формулировку в форме Галеркина [5].

Для этого обозначим невязку исходного уравнения:

$$R(P) = -\mathrm{div}\left(\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P + P_c^{ph}\right)\right).$$

Потребуем, чтобы невязка была ортогональна пространству пробных функций Ф, т.е.

$$\int_{\Omega}\left(-\mathrm{div}\left(\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P+P_c^{ph}\right)\right)\right)\psi\, d\Omega=0,\qquad \forall\psi\epsilon\Phi.$$

Преобразуем интеграл с использованием формулы Грина, распишем интеграл по границе с учетом краевых условий и, для исключения из суммы интеграла по S_1, потребуем, чтобы $\Phi = H_0$, т.е. чтобы пробные функции были из пространства функций, имеющих суммируемые с квадратом производные, и равных нулю на границе S_1. Решение задачи P будет принадлежать пространству H_g . Перепишем получившееся выражение:

$$\int_{\Omega}\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}P\ \mathrm{grad}\,\psi_0\, d\Omega=\int_{\Omega}\left(\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\left(\mathrm{grad}\left(P_c^{ph}\right)\right)\right)\psi_0 d\Omega+$$

$$+\int_{S_2}\theta\psi_0 dS_2\,,\qquad \forall\psi_0\epsilon H_0\ .$$

Получим аппроксимацию уравнения Галеркина. Для этого возьмем конечноэлементные пространства V_0 , V_g , которые аппроксимируют H_0 и H_g соответственно. Для этого заменим $P\epsilon H_g$ на аппроксимирующую $P^h\,\epsilon V_g$ и $v_0\epsilon H_0$ на $v_0^h\,\epsilon V_g$:

$$\int_{\Omega}\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}P^h\ \mathrm{grad}\,\psi_0^h\, d\Omega=\int_{\Omega}\left(\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\left(\mathrm{grad}\left(P_c^{ph}\right)\right)\right)\psi_0^h d\Omega+$$

$$+\int_{S_2}\theta\psi_0^h dS_2\,,\qquad \forall\psi_0^h\epsilon V_0\ .$$

Пусть $\{\psi_i\}$ – базис V (пространство, аппроксимирующее H). Тогда $\psi_0^h \epsilon V_0$ и $P^h \epsilon V_g$ может быть представлено в виде:

$$\psi_0^h = \sum_{i \epsilon N_0} q_i^v \psi_i,$$

$$P^h = \sum_{j=1}^{n} q_j \psi_i,$$

где N_0 – множество индексов i таких, что ψ_i являются базисными функциями пространств V_0 , V_g .

И уравнение Галеркина эквивалентно следующей СЛАУ для компонент вектора q с индексами $j \epsilon N_0$:

$$\sum_{j=1}^{n} \left(\int_{\Omega} \mathbf{K} \sum_{ph=1}^{PH} \frac{k^{ph}}{\eta^{ph}} \operatorname{grad} \psi_j \operatorname{grad} \psi_i \, d\Omega \right) q_i = \int_{\Omega} \mathbf{K} \sum_{ph=1}^{PH} \frac{k^{ph}}{\eta^{ph}} \left(\operatorname{grad}\left(P_c^{ph}\right) \right) \psi_i \, d\Omega +$$

$$+ \int_{S_2} \theta \psi_i dS_2 \, , i \epsilon N_0.$$

Недостающие n-n_0 уравнений для компонент вектора q с индексами $i \notin N_0$ могут быть получены из условия (3):

$$\sum_{j=1}^{n} q_j \psi_i \, |_{S_1} = P_g.$$

Будем решать трехмерную задачу в декартовой системе координат с использованием базисных функций первого порядка на шестигранниках, поэтому перейдем от шестигранников к единичному кубу.

Шаблонный элемент: $\Omega^E = \{(\xi, \eta, \zeta) | -1 \leq \xi, \eta, \zeta \leq 1, \}$.

Введем на нем базисные функции:

$$\hat{\varphi}_i = W_{\mu(i)}(\xi) W_{\nu(i)}(\eta) W_{\vartheta(i)}(\zeta), \qquad i = \overline{1,8}$$

$$W_1(\alpha) = \frac{1-\alpha}{2}, \qquad W_2(\alpha) = \frac{\alpha-(-1)}{2};$$

$$\mu(i) = ((i-1)\mathrm{mod}2) + 1; \qquad \nu(i) = \left(\frac{(i-1)}{2}\mathrm{mod}2\right) + 1;$$

$$\vartheta(i) = \left(\frac{(i-1)}{4}\right) + 1.$$

Тогда переход от единичного куба к шестиграннику с вершинами $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$ будет иметь вид:

$$x = \sum_{i=1}^{8} \hat{x}_i \hat{\varphi}_i(\xi,\eta,\zeta)\,;\; y = \sum_{i=1}^{8} \hat{y}_i \hat{\varphi}_i(\xi,\eta,\zeta)\,;\; z = \sum_{i=1}^{8} \hat{z}_i \hat{\varphi}_i(\xi,\eta,\zeta),$$

а базисные функции на шестиграннике определятся в виде:

$$\psi_i(x,y,z) = \hat{\varphi}_i\big(\xi(x,y,z), \eta(x,y,z), \zeta(x,y,z)\big).$$

Компоненты локальных матриц жесткости имеют вид:

$$\hat{G}_{ij} = \int_{\Omega_k} \lambda \mathrm{grad}\,\psi_j\ \mathrm{grad}\,\psi_i d\Omega.$$

Или, с учетом использования шаблонного элемента:

$$\hat{G}_{ij} = \bar{\lambda} \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \big(J^{-1}\mathrm{grad}\hat{\varphi}_i(\xi,\eta,\zeta)\big)^T J^{-1}\mathrm{grad}\hat{\varphi}_i(\xi,\eta,\zeta)|J|\, d\xi d\eta d\zeta.$$

При этом $\bar{\lambda}$ – осредненное значение $\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}$ на элементе. J – матрица Якоби, связывающая единичный куб и шестигранный элемент:

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{pmatrix}.$$

Компоненты вектора b в рамках поставленной задачи нулевые, т.к. влияние капиллярного давления на задачах с большими размерами незначительно.

Для учета ненулевых вторых краевых условий определим вклад в вектор правой части СЛАУ b от узлов грани с этими краевыми условиями как

$$\hat{b}_{S_2} = \hat{C}_{S_2}\hat{f}_{S_2},$$

где, если меняющиеся на грани координаты шаблонного элемента обозначить за μ и ν, а преобразование между шаблонной гранью и гранью элемента обозначить за J^{2D}, компоненты матрицы $\hat{C}_{S_2}$ будут иметь вид:

$$\hat{C}_{S_2\,ij} = \int_{-1}^{1}\int_{-1}^{1} \hat{\varphi}_i(\mu,\nu)\hat{\varphi}_j(\mu,\nu)|J^{2D}|d\mu d\nu,$$

а $\hat{f}_{S_2}$ – вектор из значений потока через грань в узлах сетки.

Для учета первых краевых будем находить максимальное число на диагонали, умножать его на 10^9 и считать результат «большим числом». После чего для каждого узла с первыми краевыми на соответствующей строке матрицы на диагональ поставим большое число, а в векторе b заменим соответствующую компоненту на значение искомой функции в узле, умноженное на выбранное большое число.

Вычислять компоненты матрицы и правой части СЛАУ Gq = b будем численно, с помощью метода Гаусса-3. Саму систему уравнений будем решать с использованием прямого решателя Pardiso из библиотеки Intel MKL.

1.4 РАСЧЕТ ПОТОКА СМЕСИ И ПЕРЕСЧЕТ СОСТОЯНИЙ ЯЧЕЕК КОНЕЧНОЭЛЕМЕНТНОЙ СЕТКИ

Решив задачу и получив распределение поля давления, с учетом формулы (1) мы сможем вычислить объемы смеси, перетекающие через грань Γ_i элемента Ω_e за единицу времени:

$$V'_{\Gamma_i\Omega_e} = -\int\limits_{\Gamma_i}\left[\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\mathrm{grad}\left(P+P_c^{ph}\right)\right]*\vec{n}_{\Gamma_i}d\Gamma, \qquad (5)$$

где $\vec{n}_{\Gamma_i}$ – вектор нормали к грани с фиксированным направлением для этой грани. Для каждого элемента хранится флажок соответствия внешней нормали его локальной грани и фиксированного направления нормали этой грани: если вектора совпадают, то флаг равен 1, иначе -1.

С учетом найденного поля давления, для вычисления $V'_{\Gamma_i\Omega_e}$ будем использовать следующее соотношение:

$$V'_{\Gamma_i\Omega_e} = -\mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}\int\limits_{-1}^{1}\int\limits_{-1}^{1}[J^{-1}\mathrm{grad}(\varphi_\mathrm{i})]*\vec{n}_{\Gamma_i}|J^{2D}|\,d\Gamma,$$

где $J^{2D}-$ матрица Якоби, отвечающая за преобразование между шаблонной гранью и гранью элемента. Интеграл вычисляется численно, методом Гаусса-3.

Для граней внутри расчетной области объем смеси, перетекающий за единицу времени, определяется как взвешенная сумма объемов, вычисленных для каждого из элементов $\Omega_e,\ \Omega_k$:

$$V'_{\Gamma_i} = \omega_{\Gamma_i}V'_{\Gamma_i\Omega_e} + \left(1-\omega_{\Gamma_i}\right)V'_{\Gamma_i\Omega_k}, \qquad 0 \le \omega_{\Gamma_i} \le 1.$$

Для внешних граней, соответствующих зонам перфорации с заданным потоком, корректируем перетекающий объем так, чтобы выполнялось условие:

$$V' = \theta\sum_{\mu\in I_h,\nu}\mathrm{mes}(\Gamma_{\mu,\nu}),$$

где I_h - множество граней зоны перфорации, ν – номера элементов у скважины.

Для этого распределим заданный на скважине поток на все грани, формирующие зону перфорации этой скважины. Тогда V'_{Γ_i} будет вычисляться следующим образом:

$$V'_{\Gamma_i} = \frac{|V'|}{\sum_{\Gamma_i \in \{\Gamma_{\mu,\nu}, \mu \in I_h\}} \mathrm{mes}(\Gamma_{\mu,\nu})} * \mathrm{mes}(\Gamma_i).$$

Так как мы считаем эффект гравитации несущественно малым (что допустимо для месторождений с небольшой толщиной и без газа), то объем перетекающих фаз $V_{\Gamma_i}^{ph\prime}$ в единицу времени могут быть вычислены как:

$$V_{\Gamma_i}^{ph\prime} = V'_{\Gamma_i} \frac{k^{ph}}{\eta^{ph} \sum_{n=1}^{NP} \frac{k^n}{\eta^n}}, \tag{6}$$

где коэффициенты вязкости и фазовых проницаемостей берутся с того элемента, из которого вытекает смесь.

Использование узлового МКЭ с базисными функциями для аппроксимации задачи расчета давления по методу Галеркина приводит к решению, не гарантирующему выполнение закона сохранения масс. Для решения этой проблемы предлагается использовать реализованный в программном комплексе [6-8] метод балансировки потоков. В его основе лежит следующая идея: для потока через каждую грань (с нефиксированным потоком) определяем соответствующую минимальную корректирующую добавку, обеспечивающую более высокий уровень сохранения массы смеси. И с учетом этой корректировки определяются сбалансированные потоки смеси через грань, которые и используются дальше. Более подробно этот метод описан, например, в работах [9, 10, 11].

За время Δt объем будет вычисляться как $V_{\Gamma_i}^{ph} = V_{\Gamma_i}^{ph\prime} * \Delta t$. Интервал времени для пересчета выбирается из условия, что с ячейки не может вытечь больше фазы вещества, чем ее имеется на начало интервала времени.

$$\Delta t \leq \frac{\mathrm{mes}(\Omega_e) \Phi S^{ph}}{\sum_{i \in I_{\Omega_e}^{out,ph}} \left| V_{\Gamma_i}^{ph\prime} \right|}, \tag{7}$$

для всех фаз на всех элементах. В формуле (7) $I_{\Omega_e}^{out,ph}$– множество локальных номеров граней элемента Ω_e, через которые вытекает фаза ph, S^{ph} – насыщенность ph фазы в ячейке Ω_e.

Так как в процессе моделирования на элементе может остаться слишком мало какой-либо фазы, что приведет к слишком маленькому шагу по времени, введем пороговое значение Δt_{min}, определяющее минимальное допустимое значение Δt. Если при $\Delta t = \Delta t_{min}$ какой-то фазы не хватает на элементе, Δt выбирается равным Δt_{min} и увеличивается объем других вытекающих фаз так, чтобы для всех элементов выполнилось:

$$\Delta t \leq \frac{\mathrm{mes}(\Omega_e)\Phi}{\sum_{i \in I_{\Omega_e}^{out,ph}} \left|V_{\Gamma_i}'\right|}. \tag{8}$$

Как правило, для части ячеек (например, для маленьких по объему или высокопроницаемых ячеек) необходимый шаг по времени очень мал по сравнению с допустимым шагом по времени для большинства ячеек. Поэтому для экономии вычислительных ресурсов используется разработанный и реализованный в програмном комплексе [6-8] алгоритм группирования ячеек. Его суть заключается в выделении групп ячеек, для каждой из которых используется свой шаг по времени: от заданного пользователем Δt^{max} для первой группы до $\frac{\Delta t^{max}}{2^{n_g-1}}$ для группы с номером n_g. После чего для каждого элемента выбирается группа n_g так, чтобы необходимый шаг Δt был не меньше $\frac{\Delta t^{max}}{2^{n_g-1}}$, но с минимальным номером группы. После чего состояния ячеек в каждой группе обновляются в соответствии со своим шагом по времени. Подробно данный метод рассмотрен, например, в работе [12].

Обозначим новые объемы фаз на элементе как:

$$\tilde{V}_{\Omega_e}^{ph} = \mathrm{mes}(\Omega_e)\Phi S^{ph} + \sum_{i \in I_{\Omega_e}^{in,ph}} |V_{\Gamma_i}^{ph}| - \sum_{i \in I_{\Omega_e}^{out,ph}} \left|V_{\Gamma_i}^{ph}\right|,$$

где $I_{\Omega_e}^{in,ph}$– множество локальных номеров граней элемента Ω_e, через которые втекает фаза ph.

Определив перетекающие за интервал времени объемы фаз $V_{\Gamma_i}^{ph}$, новые значения насыщенностей можем получить по следующей формуле:

$$\tilde{S}_{\Omega_e}^{ph} = \frac{\tilde{V}_{\Omega_e}^{ph}}{\mathrm{mes}(\Omega_e)\Phi}.$$

Новые температуры фаз вычислим из условия сохранения тепловой энергии каждой из фаз. С учетом, что тепловая энергия фазы вычисляется как

$$Q^{ph} = c_{\Omega_e}^{ph} m_{\Omega_e}^{ph} \Delta T_{ph}^{\Omega_e} =$$

$$= c_{\Omega_e}^{ph} \rho_{\Omega_e}^{ph} \left(\left[\mathrm{mes}(\Omega_e)\Phi \tilde{S}_{\Omega_e}^{ph} - \sum_{i \in I_{\Omega_e}^{out,ph}} \left| V_{\Gamma_i}^{ph} \right| \right] + \sum_{i \in I_{\Omega_e}^{in,ph}} |V_{\Gamma_i}^{ph}| \right) \left(\tilde{T}_{\Omega_e}^{ph} - T_{\Omega_e}^{ph} \right),$$

где в квадратных скобках объем, оставшийся на элементе (далее обозначим его $\hat{V}_{\Omega_e}^{ph}$). Так же заметим, что не рассматривается моделирование источников тепла в результате химических реакций или фазовых переходов. Тогда новая температура фазы может быть вычислена как:

$$\tilde{T}_{\Omega_e}^{ph} = \frac{c_{\Omega_e}^{ph} \rho_{\Omega_e}^{ph} \hat{V}_{\Omega_e}^{ph} T_{\Omega_e}^{ph} + \sum_{k \in I_{\Omega_e}^{in,ph}} c_{\Omega_k}^{ph} \rho_{\Omega_k}^{ph} \left| V_{\Gamma_i}^{ph} \right| T_{\Omega_e}^{ph}}{c_{\Omega_e}^{ph} \rho_{\Omega_e}^{ph} \hat{V}_{\Omega_e}^{ph} + \sum_{k \in I_{\Omega_e}^{in,ph}} c_{\Omega_k}^{ph} \rho_{\Omega_k}^{ph} \left| V_{\Gamma_i}^{ph} \right|}.$$

Для моделирования теплового взаимодействия породы и флюида воспользуемся предложенным в [13] подходом. В задачах нефтедобычи как правило влияние на температуру смеси и породы в основном оказывает теплоперенос за счет движения флюида, а воздействие теплопроводности на температурное поле низко. Поэтому вместо системы дифференциальных уравнений для расчета поля температуры будем использовать коэффициент β для характеристики скорости теплообмены фильтрующейся смеси и породы.

Новую температуру смеси $\tilde{T}_{mix}$ и породы $\tilde{T}_{base}$ после теплообмена между ними определим из системы уравнений:

$$\begin{cases} \dfrac{c^{mix}\rho^{mix}\Phi}{\Delta t}\left(\tilde{T}_{mix} - T_{mix}\right) = -\beta\left(\tilde{T}_{mix} - \tilde{T}_{base}\right), \\ \dfrac{c^{base}\rho^{base}(1-\Phi)}{\Delta t}\left(\tilde{T}_{base} - T_{base}\right) = \beta\left(\tilde{T}_{mix} - \tilde{T}_{base}\right) \end{cases}, \tag{9}$$

где c^{mix}, ρ^{mix} – эффективные теплоемкость и плотность смеси фаз, c^{base}, ρ^{base} – эффективные теплоемкость и плотность породы, T_{mix}, T_{base} – эффективные температуры смеси фаз и породы до теплообмена.

Эффективные теплоемкость, плотность и температура смеси вычисляется как средневзвешенное значение входящих в состав смеси фаз пропорционально их массам:

$$\rho^{mix} = \frac{\sum_{ph=1}^{PH}\rho^{ph}m^{ph}}{\sum_{ph=1}^{PH}m^{ph}} = \frac{\sum_{ph=1}^{PH}\rho^{ph}\rho^{ph}\tilde{V}_{\Omega_e}^{ph}}{\sum_{ph=1}^{PH}\rho^{ph}\tilde{V}_{\Omega_e}^{ph}},$$

$$c^{mix} = \frac{\sum_{ph=1}^{PH}c^{ph}\rho^{ph}\tilde{V}_{\Omega_e}^{ph}}{\sum_{ph=1}^{PH}\rho^{ph}\tilde{V}_{\Omega_e}^{ph}},$$

$$T_{mix} = \frac{\sum_{ph}^{PH}\rho^{ph}\tilde{V}_{\Omega_e}^{ph}T_{\Omega_e}^{ph}}{\sum_{ph=1}^{PH}\rho^{ph}\tilde{V}_{\Omega_e}^{ph}}.$$

Чтобы выписать решение системы уравнений (9) в явном виде, введем обозначения:

$$C^{mix} = \frac{c^{mix}\rho^{mix}\Phi}{\Delta t},$$

$$C^{base} = \frac{c^{base}\rho^{base}(1-\Phi)}{\Delta t}.$$

И тогда итоговые формулы для вычисления новой температуры смеси и породы:

$$\tilde{T}_{base} = T_{base} + \frac{\beta C^{mix}(T_{mix} - T_{base})}{\beta(C^{mix} + C^{base}) + C^{mix}C^{base}}, \quad (10)$$

$$\tilde{T}_{mix} = T_{mix} + \frac{C^{mix}T_{mix} + \beta\tilde{T}_{base}}{C^{mix} + \beta}. \quad (11)$$

Вследствие изменения температуры на элементе может измениться и значение вязкости фаз, так как оно зависит от температуры. В рассматриваемой задаче будем считать, что эта зависимость задана таблично и промежуточные значения могут быть определены из построенного по табличным данным линейного сплайна.

2. ОПИСАНИЕ ПРОГРАММЫ

Программа написана на языке C++ и предназначена для моделирования процесса многофазной фильтрации с использованием горизонтальных скважин с учетом теплового воздействия на слой.

Решение СЛАУ реализовано с использованием прямого решателя Pardiso из библиотеки Intel MKL.

Входными данными программы являются:

1. Базовая сетка в виде:
    а) списка координат узлов сетки;
    б) списка конечных элементов как набора глобальных номеров узлов и как набора глобальных номеров граней;
    в) списка соответствия каждому элементу номера материала;
    г) списка элементов и локальных номеров граней, где заданы краевые условия первого рода, а также заданное на этих гранях давление.
2. Информация о скважинах:
    а) местоположение скважин и зон перфорации;
    б) размер области встройки скважин, количество элементов в области встройки скважин и коэффициент разрядки для этой области;
    в) режим работы скважин (заданное давление или поток на скважине и время работы при каждом режиме);
    г) закачиваемая смесь (фазовый состав, температура).
3. Физические свойства:
    а) пористость, проницаемость, температура, теплопроводность породы;
    б) плотность, вязкость, возможно таблично зависящая от температуры, теплопроводность фаз;
    в) первоначальное распределение фаз в области (в виде значения насыщенностей каждой фазы на всех элементах).

4. Настройки решения задачи (количество дней для моделирования, шаг по времени).

Выходными данными являются:

1. Поля распределения насыщенностей фаз, давления, температуры.

2. Графики суммарного объема отбора смеси и каждой фазы по скважинам, мгновенной доли фазы в отбираемой смеси, мгновенного объема отбора фазы.

2.1 СТРУКТУРЫ ДАННЫХ И МОДУЛИ

В разработанной программе присутствуют следующие модули:

1. Модуль встройки горизонтальных скважин.
2. Модуль расчета давления.
3. Модуль численного интегрирования.
4. Модуль пересчета состояния ячеек конечноэлементной сетки при заданном давлении.

В программе к ключевым относятся структуры, хранящие данные о:

1. Узлах сетки в виде трех координат узла в пространстве.
2. Конечных элементах в виде массивов глобальных номеров узлов элемента, глобальных номеров граней и глобальных номеров элементов-соседей по каждой грани, а так же хранится номер материала элемента.
3. Состоянии фазы на элементе в виде значений плотности, вязкости, структурной проницаемости и температуры фазы.
4. Скважинах в виде списка зон перфорации, где указаны грани, описывающие зону перфорации, тип краевого условия и значение, заданное пользователем (давление или поток).
5. Сетке в виде списка элементов, узлов, скважин.
6. Насыщенности фаз в каждом элементе в виде двумерного массива.
7. Давлении в каждом из узлов (результат решения краевой задачи для определения поля давления).

8. Характеристиках фаз на каждом элементе в виде двумерного массива состояний фаз.
9. Перетекающих через каждую грань потоках смеси.
10. Перетекающих через каждую грань потоках каждой фазы в виде двумерного массива.
11. Перетекающих через каждую грань объемах каждой фазы в виде двумерного массива.

2.2 МОДУЛЬ ПОСТРОЕНИЯ КОНЕЧНОЭЛЕМЕНТНЫХ СЕТОК

Модуль разработан в рамках данной работы. На основе базовой сетки, информации о местоположении скважин и свойствах сетки вблизи скважины формируется новая сетка со встроенными горизонтальными скважинами.

Входные данные:

1. Базовая сетка.

2. Информация о скважинах (в том числе способ встройки скважин).

В зависимости от способа встройки скважин по описанному в пункте 1.2 алгоритму производится явная или неявная встройка.

При явном способе в базовой сетке удаляются элементы, попадающие в область встройки скважины. После чего для каждой скважины от ее центра к граничным узлам области встройки формируются новые узлы, элементы и грани. Затем производится перенумерация старых узлов и элементов с учетом удаления фрагментов сетки. Новая сетка записывается обратно и создаются файлы со списком элементов и граней, аппроксимирующих зону перфорации.

При неявном способе по информации о скважинах формируются координаты дополнительных ячеек, и эта информация записывается в специальный файл, откуда реализованный в программном комплексе [6-8] модуль получает данные о новых объектах в сетке и перестраивает сетку на несогласованную.

После чего еще один программный блок формирует список граней и элементов, описывающих зону перфорации.

Выходные данные:

1. Конечноэлементная сетка (с горизонтальными скважинами).

2. Список элементов и граней, аппроксимирующих зону перфорации.

2.3 МОДУЛЬ РАСЧЕТА ДАВЛЕНИЯ

Данный модуль предназначен для решения краевой задачи при известных насыщенностях фаз и свойствах породы. В рамках данной работы реализовано решение трехмерной краевой задачи на шестигранных сетках с использованием трилинейного базиса.

Входные данные:

1. Конечноэлементная сетка.

2. Свойства породы (проницаемость, пористость).

3. Свойства фазы в ячейках (плотность, вязкость, структурная проницаемость).

4. Массивы граней с краевыми условиями (их тип и заданное значение давления или потока).

Согласно алгоритму пункта 1.3, в этом модуле собираются локальные матрицы и вектора правой части (т.к. элементы шестигранные, применяется численное интегрирование для взятия интегралов). После чего формируется глобальная матрица и глобальный вектор правой части, где последовательно учитываются вторые и первые краевые условия. Из полученной СЛАУ с использованием прямого решателя Pardiso из библиотеки Intel MKL получается вектор из значений давления во всех узлах конечноэлементной сетки.

Выходные данные:

1. Массив значений давления во всех узлах конечноэлементной сетки.

2.4 МОДУЛЬ ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

Модуль численного интегрирования реализован в рамках данной работы. При решении краевой задачи для нахождения поля давления используются шестигранные элементы с переводом в шаблонные координаты, как отмечено в пункте 1.3. Поэтому при сборке СЛАУ численно находятся объемные интегралы (для матрицы жесткости) и поверхностные интегралы (для учета вторых краевых условий). Для этого будем использовать трехточечный метод Гаусса. То есть, нам надо суммировать подынтегральные функции в определенных точках интегрирования и с определенными весами. Например, для объемного интеграла:

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta,\zeta)d\xi d\eta d\zeta = \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3} \alpha_i\alpha_j\alpha_k f\left(w_i w_j w_k\right),$$

$$\alpha_1 = \frac{8}{9}, \alpha_2 = \frac{5}{9}, \alpha_3 = \frac{5}{9}, w_1 = 0, w_2 = \sqrt{0.6}, w_3 = -\sqrt{0.6}.$$

Так как используется шаблонный элемент от -1 до 1 по каждой из осей, переходить к новым координатам не надо; и так как на шаблонном элементе значения базисных функций и их производных в точках интегрирования одинаковы для всех элементов сетки, их можно вычислить заранее всего один раз.

Входные данные:

1. Координаты узлов элемента.

Для пришедшего на вход элемента сетки расчитывается его матрица жесткости (без домножения на коэффициент λ) или матрица масс для грани без коэффициента γ (для вычисления вклада в вектор правой части при учете вторых краевых условий). Для этого определяется якобиан и, при необходимости, обратная матрица Якоби. После чего по трехточечному методу Гаусса последовательно определяется каждый элемент требуемой локальной матрицы.

Выходные данные:

1. Локальная матрица жесткости или масс (без домножения на коэффициент) пришедшего на вход конечного элемента.

2.5 МОДУЛЬ ПЕРЕСЧЕТА СОСТОЯНИЯ ЯЧЕЕК КОНЕЧНОЭЛЕМЕНТНОЙ СЕТКИ ПРИ ЗАДАННОМ ДАВЛЕНИИ

При помощи этого модуля после решения краевой задачи по полученному полю давления расчитываются потоки смеси через грани и, в зависимости от этого, новые насыщенность, температура и вязкость фаз, температура породы на каждом элементе. Бóльшая часть данного модуля реализована в программном комплексе [6-8].

Входные данные:

1. Конечноэлементная сетка.

2. Давление в узлах сетки.

3. Массив граней с заданным потоком (грани зон перфорации, где заданы вторые краевые условия).

4. Состояние ячеек на начало шага по времени.

5. Состав смеси для закачки (насыщенность и температура фаз в смесях, закачиваемых через скважины, или смеси, втекающей через удаленную границу).

6. Δt, максимальный шаг по времени, заданный пользователем.

Как описано в пункте 1.4, в этом модуле сперва по формуле (5) с использованием давления, полученного в результате решения краевой задачи, расчитываются потоки смеси через грань для каждого элемента. После чего эти потоки осредняются или корректируются в соответствии с фиксированным значением на определенных гранях.

Далее применяется подмодуль балансировки потоков смеси. В результате вычисляются корректирующие добавки к потокам смеси. Если необходимая точность выполнения закона сохранения масс не достигнута, изменяются параметры регуляризации и вновь решается СЛАУ. Иначе вычисляются сбаланси-

рованные потоки с учетом найденных корректирующих добавок. Подробнее этот алгоритм рассмотрен в [10].

Далее определяются потоки фаз по соотношению (6). Для этого в цикле по всем элементам для каждой грани, через которую вытекает смесь, расчитывается поток фаз. Для граней, через которые смесь втекает в элемент с границы расчетной области или через зоны перфорации, поток фаз расчитывается с использованием свойств смеси для закачки.

После определения перетекающих потоков фаз определим необходимый шаг по времени на каждом элементе по формуле (7). Если полученный шаг слишком маленький, помечаем элемент как требующий «подмены фаз» и считаем шаг по времени по формуле (8). После этого распределяем элементы по группам в соответствии с допустимым шагом по времени (подробнее о процедуре группирования написано в работе [12]) и расчитываем перетекающие объемы фаз и смеси.

Теперь можно обновить состояния ячеек. Для этого вычисляется новый объем фазы в элементе, обновляется насыщенность фазы. Затем вычисляется новая температура смеси и породы по соотношениям (10), (11). Исходя из новых значений температуры, обновляется вязкость фаз.

Выходные данные:

Состояние ячеек на конец шага по времени.

3. ИССЛЕДОВАНИЯ

3.1 ПОРЯДОК СХОДИМОСТИ И АППРОКСИМАЦИИ ПРИ РЕШЕНИИ ЭЛЛИПТИЧЕСКОЙ ЗАДАЧИ ДЛЯ РАСЧЕТА ДАВЛЕНИЯ

В первую очередь проверим правильность решения краевой задачи для расчета поля давления. Для этого, с учетом использования трилинейных базисных функций, проверим, что порядок аппроксимации и сходимости соответствует теоретически ожидаемому.

Теоретически ожидаемый порядок аппроксимации трилинейного базиса – первый (смешанные произведения тоже верно аппроксимируются). В целях проверки зададим функцию P(x,y,z) в виде полинома, вычислим правую часть, удовлетворяющую

$$-\mathrm{div}(\lambda \mathrm{grad} P) = f.$$

Расчетную область зададим как куб от 0 до 5 по каждой из осей, с шагом 1. По границе зададим первые краевые условия, λ выберем равным 1. Погрешность будем считать как

$$\delta = \frac{\sum_{i=1}^{N} |P^*(x_i, y_i, z_i) - P(x_i, y_i, z_i)|}{N},$$

где N – количество узлов сетки, x_i, y_i, z_i – координаты i-ого узла сетки, $P^*(x_i, y_i, z_i)$ – аналитическиое значение функции, $P(x_i, y_i, z_i)$ – численное значение функции.

Результаты приведены в таблице 1. Как видно из результатов, на полиноме первой степени и со смешанным произведением погрешность близка к 0 (существующую ошибку можно отнести к вычислительной погрешности), а для 2 степени уже есть значительная погрешность. Следовательно, теоретически ожидаемый порядок аппроксимации подтвержден.

Таблица 1 – Исследование на порядок аппроксимации

$P^*(x, y, z)$, вид заданной функции	δ, погрешность полученного решения
$x + y + z$	1,3021E-13
$2xy + z$	6,2804E-10
$x^2 + y^2 + z^2$	1,4021E+00

Для тестирования порядка сходимости зададим $P^*(x, y, z) = \cos(x + z)$. С дроблением сетки в 2 раза по одной из осей x или z погрешность должна падать в 2 раза (у линейного базиса порядок сходимости первый). Соответственно, при дроблении сетки сразу по двум этим осям погрешность должна падать в 4 раза.

Для исследования будем использовать ту же расчетную область, за первоначальный шаг возьмем 1, и этот шаг будем уменьшать в 2 и 4 раза. Для вычисления погрешности будем брать 100 случайных точек в области и в них считать

$$\delta = \frac{\sum_{i=1}^{N} |P^*(x_i, y_i, z_i) - P(x_i, y_i, z_i)|}{N},$$

где N равно 100, x_i, y_i, z_i – координаты i-ой контрольной точки, $P^*(x_i, y_i, z_i)$ – аналитическиое значение функции, $P(x_i, y_i, z_i)$ – численное значение функции.

Порядок сходимости:

$$R = \log_2\left(\frac{\delta_h}{\delta_{h/2}}\right).$$

Результаты приведены в таблице 2.

Таблица 2 – Исследования на порядок сходимости

Шаг по сетке	δ, погрешность полученного решения	Отношение погрешностей $\frac{\delta_h}{\delta_{h/2}}$	Порядок сходимости R
1	3,6517E-01	-	-
0,5	1,2207E-01	2,9914E+00	1,58
0,25	3,3176E-02	3,6796E+00	1,88
0,125	8,4214E-03	3,9395E+00	1,98
0,0625	2,2092E-03	3,8119E+00	1,93

Из таблиц видно, что порядок аппроксимации и сходимости соответствуют теоретически ожидаемому.

3.2 СРАВНЕНИЕ С ЗАДАЧЕЙ, ИМЕЮЩЕЙ АНАЛИТИЧЕСКОЕ РЕШЕНИЕ

Возьмем задачу, имеющую аналитическое решение. Расчетную область зададим кубом с ребром 200м (R = 100м). Скважина радиусом $r_w = 1$м расположена в центре расчетной области вдоль оси x и закачивает 0,79м^3/сут. На границе задано пластовое давление $P_g = 130$ атм. Коэффициент, связывающий поток смеси с gradP $\lambda = \mathbf{K}\sum_{ph=1}^{PH}\frac{k^{ph}}{\eta^{ph}}$, возьмем равным 500 $\frac{\text{мД}}{\text{Па}*\text{с}}$. Сравнивать будем давление вдоль линии от центра скважины до края расчетной области.

Аналитическое решение (в цилиндрических координатах) имеет вид:

$$P(r) = \frac{\theta r_w}{\lambda}\ln\left(\frac{R}{r}\right) + P_g.$$

Для численного решения данной задачи использовались базисные функции первого порядка. На рисунке 5 показана погрешность вычисления давления на трех вложенных сетках относительно аналитического решения. Черным цветом показана погрешность для самой грубой сетки (размер первой ячейки от сква-

жины составляет 0,3м, количество шагов в сетке – 56), зеленым – на вложенной сетке, красным – на дважды вложенной сетке.

На рисунке 5 видно, что максимальное отклонение получилось в окрестности скважины и составляет 0,035% на самой грубой сетке, 0,009% на более подробной и 0,002% на сетке с шагом, уменьшенным в 4 раза относительно грубой сетки. Погрешность уменьшается примерно в 4 раза с уменьшением шага в 2 раза, что соответствует теоретически ожидаемому порядку сходимости.

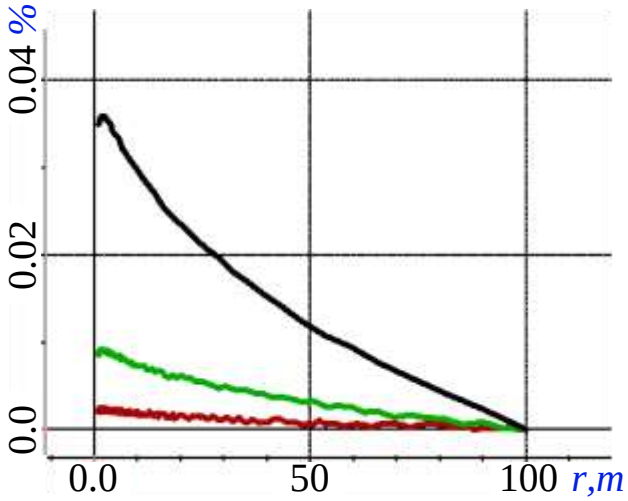

Рисунок 5 – Отклонение численного решения от аналитического

3.3 СХОДИМОСТЬ НА МОДЕЛЬНОЙ ЗАДАЧЕ ПРИ ДРОБЛЕНИИ СЕТКИ

Рассмотрим следующую модельную задачу. Расчетная область задана от -250м до 250м по осям x, y и от 0 до 100м по z. Стартовая насыщенность нефти в слое 0,85. Влияние температуры не учитываем, считая всю смесь однородной температуры.

Две скважины расположены вдоль оси x, в координате 0 по y. Сверху, на глубине 30м, находится нагнетательная скважина; под ней на 40м ниже – добывающая. Радиус задан 0,5м, протяженность зон перфорации у двух скважин по 100м. Нагнетательная скважина закачивает 100м^3/сут воды, добывающая работает с мощностью 100м^3/сут.

Сетку зададим с шагом 50м по x, y и 5м по оси z.

Попробуем уменьшать шаг по z и увеличивать количество элементов в области встройки скважины. Результирующий суммарный отбор нефти представлен на рисунке 6. Так как при дроблении по z суммарный отбор практически не меняется, будем уменьшать шаг по осям x, y в 2 раза (а так же увеличивать количество дроблений в области встройки скважины в 2 раза).

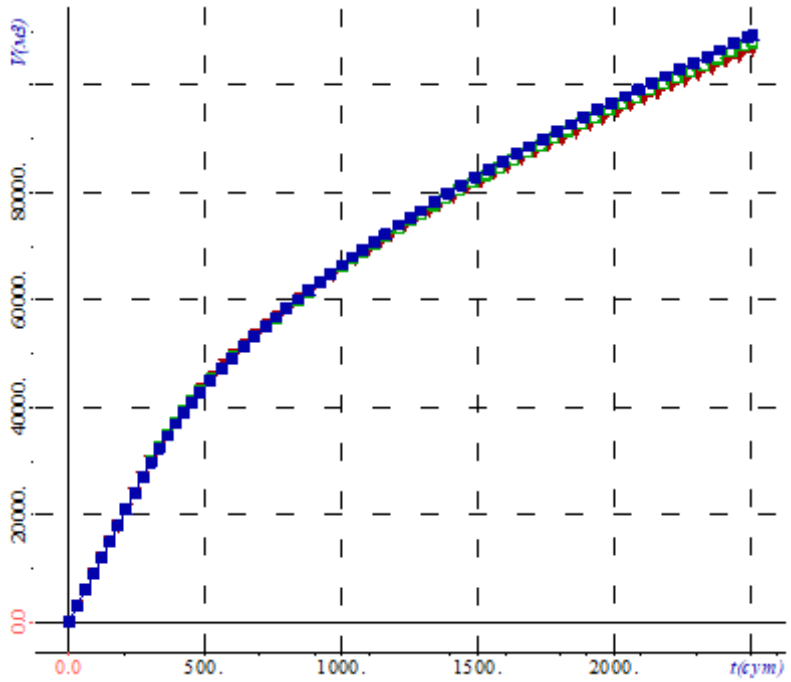

Рисунок 6 – Суммарный отбор нефти; синий – шаг по z 5м, зеленый – шаг по z 2,5м, красный – шаг по z 1,25м

На рисунке 7 показана получившаяся сетка в разрезе поперек оси x.

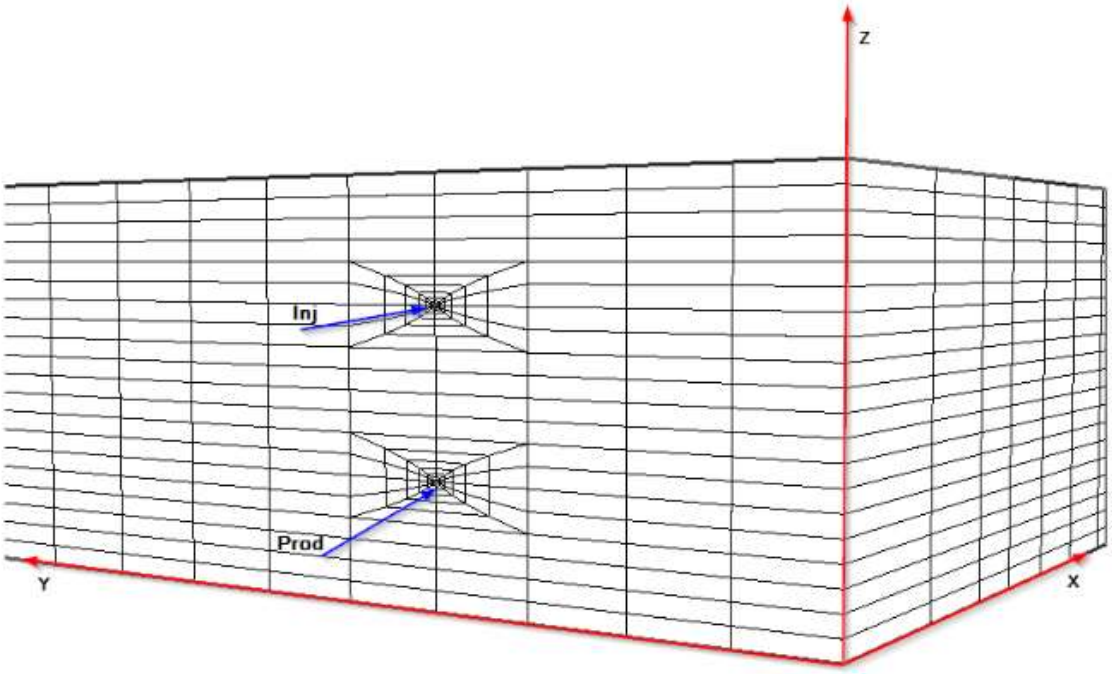

Рисунок 7 – Вид на расчетную область в разрезе поперек оси x;
шаг по x, y равен 25м

На рисунке 8 представлены полученные на разных сетках графики суммарного отбора нефти. С дроблением сетки результат все меньше отличается от предыдущего, что говорит о сходимости решения.

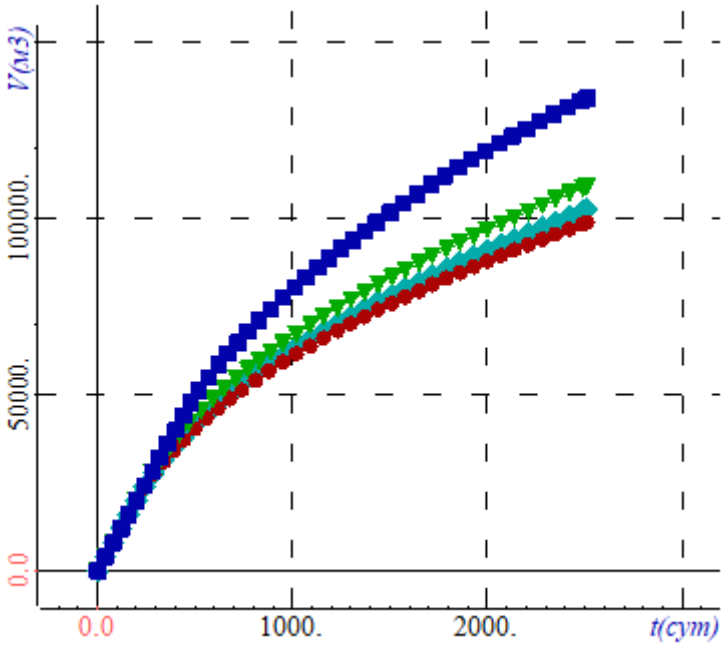

Рисунок 8 – Суммарный объем нефти на разных сетках; голубой – шаг 50, синий – шаг 25, зеленый – шаг 12,5, красный – шаг 6,25

Чтобы оценить скорость сходимости, определим разницу между решениями, полученными на сетках с разным шагом, как

$$\delta_h = V_{sum}^{h}(2500) - V_{sum}^{h/2}(2500).$$

В таком случае падение разницы получаемых решений можно оценить в виде таблицы 3.

Таблица 3 – Изменение решения с уменьшением шага по сетке.

Шаг по осям x, y	$V_{sum}^{h}(2500)$, м^3	δ_h, м^3	$\delta_h/\delta_{h/2}$
50м	133945	-	-
25м	109024	24921	-
12,5м	102554	6470	3,851777
6,25м	98794,3	3759,7	1,720882

С дроблением сетки разница между δ_h все меньше, что говорит о сходимости к некоторому решению. Полученный порядок сходимости ниже порядка сходимости вычислительной схемы расчета давления, так как тут сравниваем суммарный объем отобранной нефти, который получен после интегрирования найденного поля давления, что снижает порядок сходимости.

3.4 СРАВНЕНИЕ СПОСОБОВ ВСТРОЙКИ ГОРИЗОНТАЛЬНЫХ СКВАЖИН

Для анализа возьмем сетку с шагом сетки по оси x и y 25м, по оси z 5м. Встраивается горизонтальная скважина, параллельная оси x, длинной 500м, радиусом 0,5м и с зоной перфорации длинной 100м.

На рисунке 9 представлены визуальные отличия сеток. Здесь показано, что явная встройка геометрически более точно учитывает наличие скважины в расчетной области, а так же более точно моделирует круглую форму. В то время как неявная встройка геометрически условно описывает область, где расположена зона перфорации скважины, заменяя цилиндр параллелепипедом.

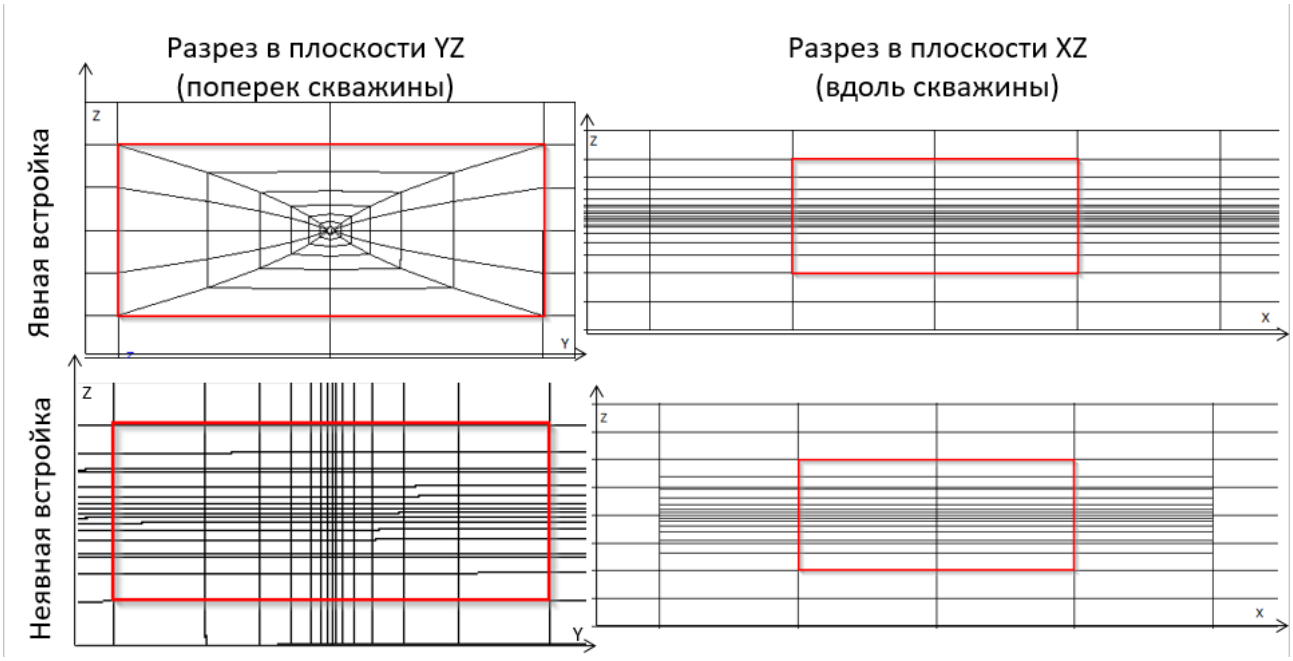

Рисунок 9 – Сравнение вида сеток при встройке скважины разными способами

Помимо точности учета геометрической формы, сравним количество дополнительных элементов и узлов при разных методах встройки. Для этого помимо имеющейся сетки рассмотрим еще одну, размером 2000м по x и y, но с прежним шагом. Параметры скважины оставим прежними.

Параметры получившихся сеток приведены в таблице 4. Как видно из приведенных данных, при неявной встройке получаем малое число узлов на больших моделях с небольшими зонами перфорации, в то время как явная встройка из-за задания скважины целиком на задачах с большими размерами расчетной области дает значительно больше новых узлов и элементов. Большое количество узлов приводит к увеличению вычислительных затрат на поиск поля давления, а рост числа элементов и граней – к повышению затрат на расчет потоков через грани и на обновление состояний ячеек.

Таблица 4 – Сравнение параметров сетки при встройке скважины разными способами

Размер, м	Встройка	Кол-во узлов	Кол-во элементов	Кол-во граней
x, y = 500, z = 100	Базовая сетка	7581	6480	20484
	Явная	10518	9024	28552
	Неявная	11421	9456	30452
x, y = 2000, z = 100	Базовая сетка	78141	72000	222000
	Явная	86874	79920	246700
	Неявная	81981	74976	231968

Так же для сравнения способов встройки скважин сравним решение задачи фильтрации на двух сетках с явной и неявной встройкой. Возьмем модельную задачу из пункта 3.3.

На рисунке 10 показано распределение насыщенности в пласте вблизи скважин при разных типах встройки на момент времени 450сут с начала моделирования. Здесь можно заметить, что вблизи нагнетательной скважины поле давления немного отличается, что связано с разной аппроксимацией источника. Но на сравнительно небольшом расстоянии от скважины поле давления одинаково.

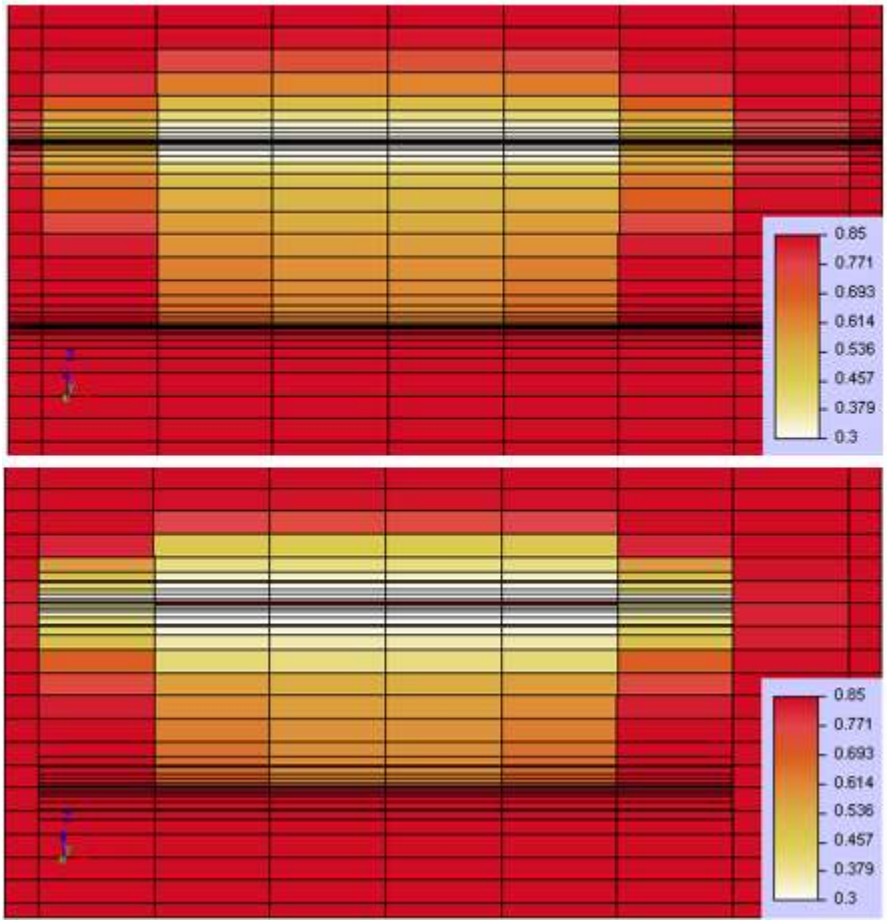

Рисунок 10 – Распределение насыщенности нефти в разрезе вдоль скважин; сверху явная встройка, снизу – неявная встройка скважин

Для оценки сопоставимости решений при разных способах встройки скважин так же приведем сравнение графиков доли нефти в отборе смеси в каждый момент времени (рисунок 11). Первое время вся добываемая смесь является нефтью (т.к. 0.15 воды в слое являются остаточной насыщенностью фазы в породе), позже доля нефти в отборе снижается в связи с обводненностью скважины (вода от нагнетательной скважины перемещается ближе к добывающей). На нижнем графике можно увидеть, что в среднем отличие составляет менее двух процентов.

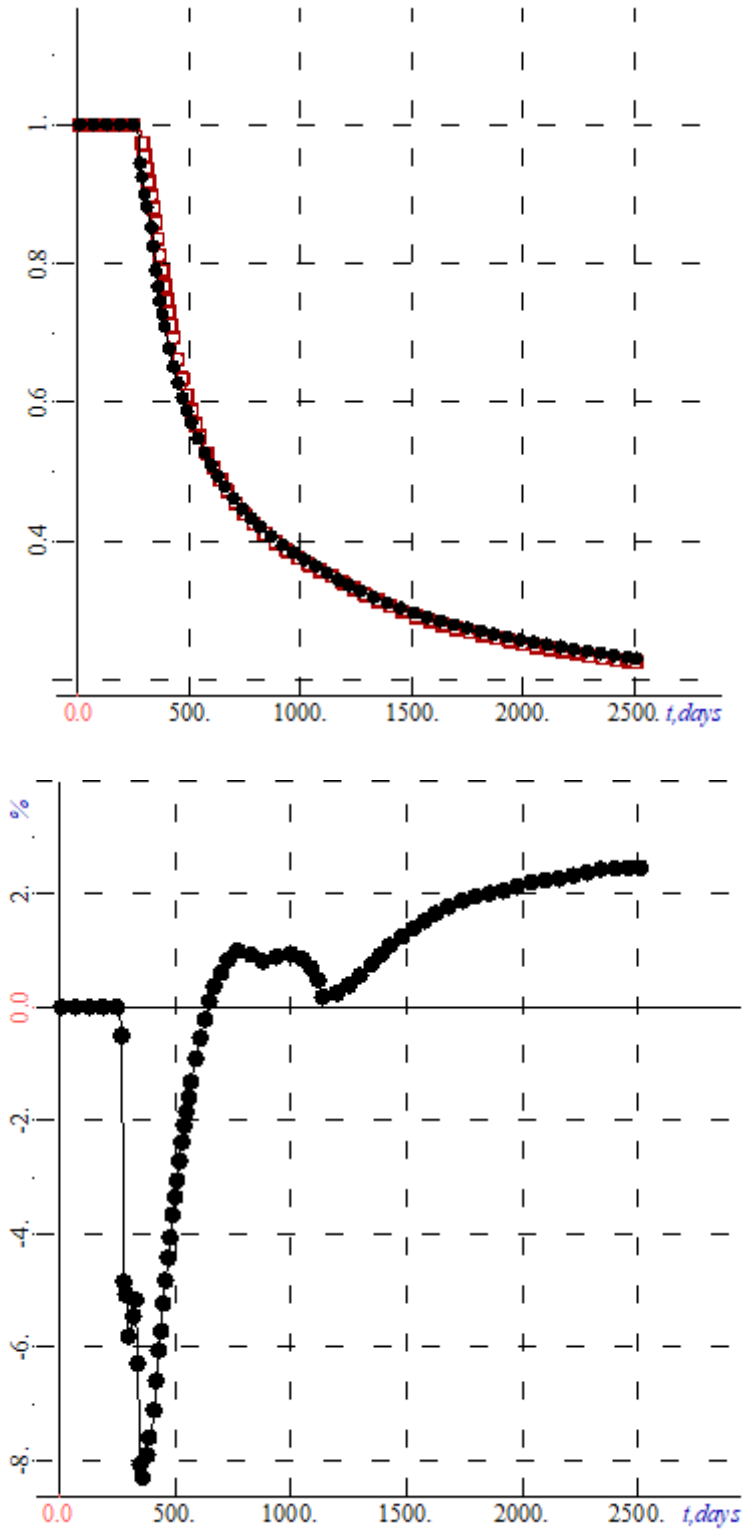

Рисунок 11 – График доли нефти в отбираемой смеси (сверху абсолютное значение, снизу – процент отличия); черный – явная встройка, красный – неявная встройка

3.5 МОДЕЛИРОВАНИЕ ТЕПЛОВЫХ МЕТОДОВ УВЕЛИЧЕНИЯ НЕФТЕОТДАЧИ ПЛАСТА

Рассмотрим следующую модельную задачу. Расчетная область задана в метрах, от -200 до 300 по оси x, от -100 до 100 по оси y и от 0 до 40 по оси z (ноль по оси z задан на глубине 100м). Пластовое давление считаем равным 16 атм, температуру слоя – 20℃. В пласте насыщенность нефти зададим 0.65, остальное – вода. Сетку зададим с шагом 30м по x и y, 5м по z. Общее время моделирования 3240сут, шаг по времени 30 дней. Зависимость вязкости нефти от температуры приведена в таблице 5.

Таблица 5 – Зависимость вязкости нефти от температуры

Температура, ℃	20	37,78	65,56	93,33	121,11	148,89	176,67	260
Вязкость, Па*с	5,78	1,38	0,187	0,047	0,0174	0,0085	0,0052	0,0025

На глубине 115м зададим нагнетательную скважину, на глубине 130м – добывающую. Скважины параллельны оси x, расположены в 0 по оси y и находятся друг под другом. Зоны перфорации в двух скважинах от -120 до 220 по оси x. Радиусы скважин зададим 1м. Закачка и отбор работают с мощностью 100м^3/сут, закачивается в пласт вода.

Температуру закачки «холодной» воды возьмем равной пластовой температуре, «горячей» – 50℃. На рисунке 12 показан общий вид поля давления в области через 30 дней после начала расчета, верхняя скважина нагнетательная, нижняя – добывающая, значения давления приведены в атмосферах.

На рисунке 13 приведен график суммарного отбора нефти при пластовой температуре закачиваемой жидкости (синий) и при нагревании закачиваемой воды до температуры 50℃ (зеленый).

Как видно из графика, использование воды с температурой выше пластовой способствует увеличению нефтеотдачи за счет уменьшения вязкости нефти в результате нагрева пласта.

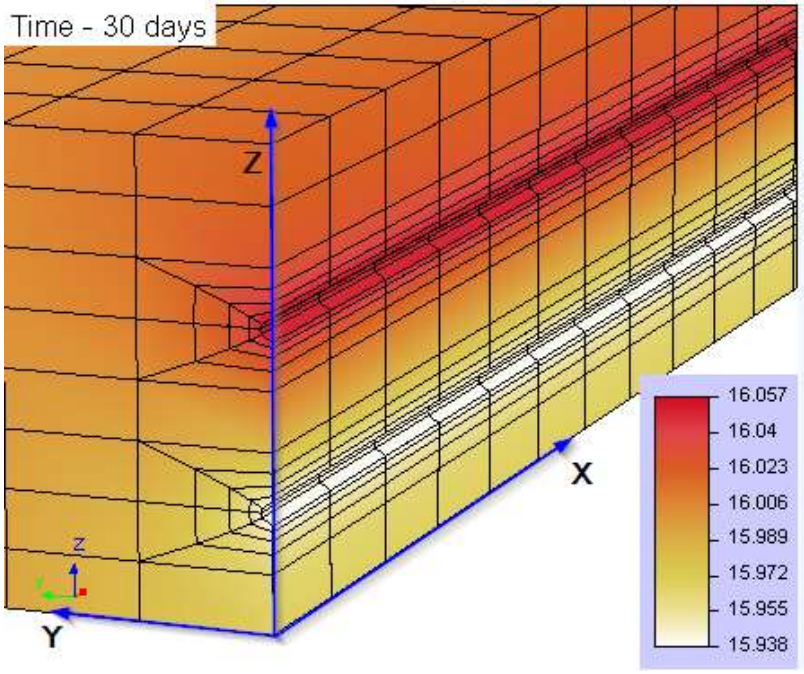

Рисунок 12 – Вид слоя в разрезе поперек оси x и вдоль скважины

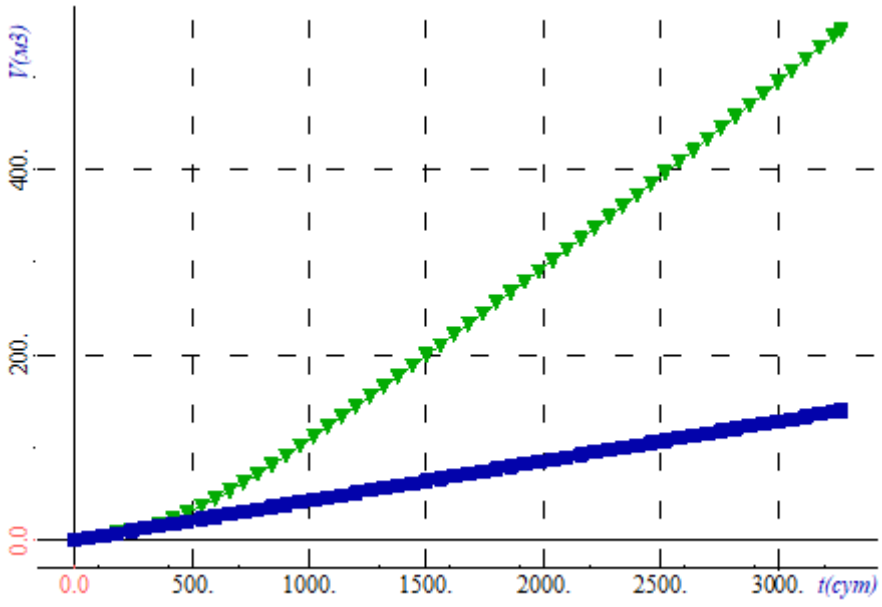

Рисунок 13 – Суммарный отбор нефти; синий – холодная вода, зеленый – горячая вода

На рисунках 14-16 представлены поля вязкости, насыщенности и температуры нефти на момент времени моделирования 1020 дней в области зоны пер-

форации на сетке, разрезанной вдоль скважин. Левое изображение соответствует ситуации, когда закачивается холодная вода (поэтому поля вязкости и температуры не меняются). Справа ситуация с закачкой горячей воды – можно увидеть, как прогревается пласт (а значит и повышается температура нефти). Так же повышение температуры снижает вязкость нефти. Как следствие, мы наблюдаем различное распределение насыщенности нефти в области при разных температурах закачиваемой смеси.

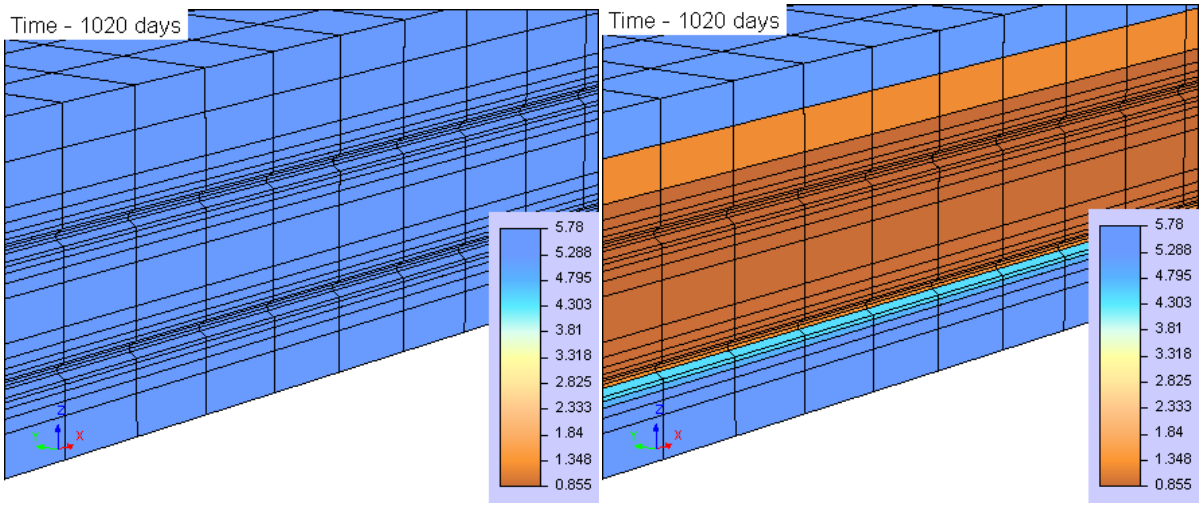

Рисунок 14 – Поле вязкости нефти; слева – при T = 20°C, справа – при T = 50°C

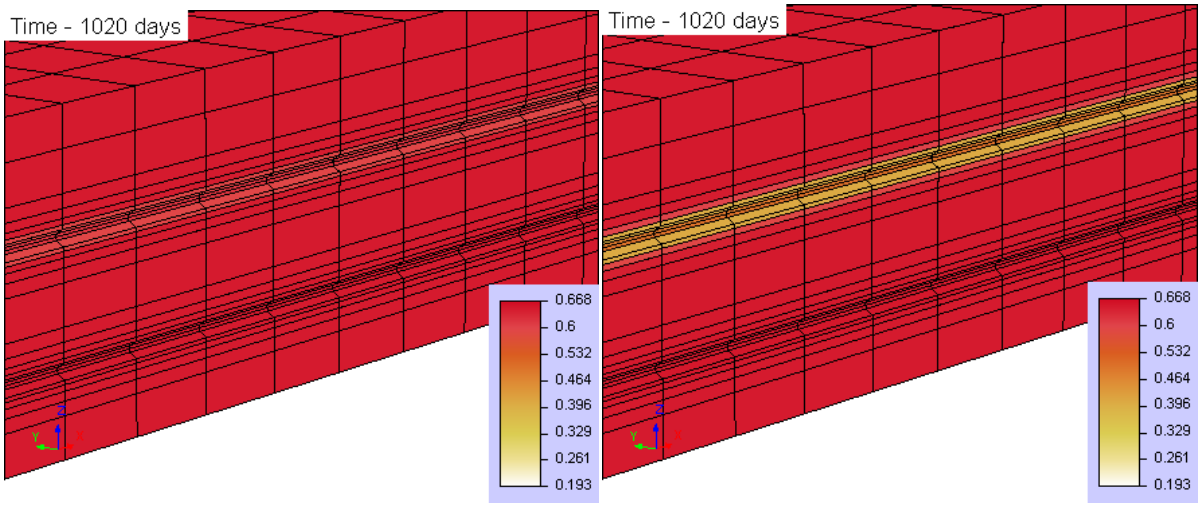

Рисунок 15 – Поле насыщенности нефти; слева – при T = 20°C, справа – при T = 50°C

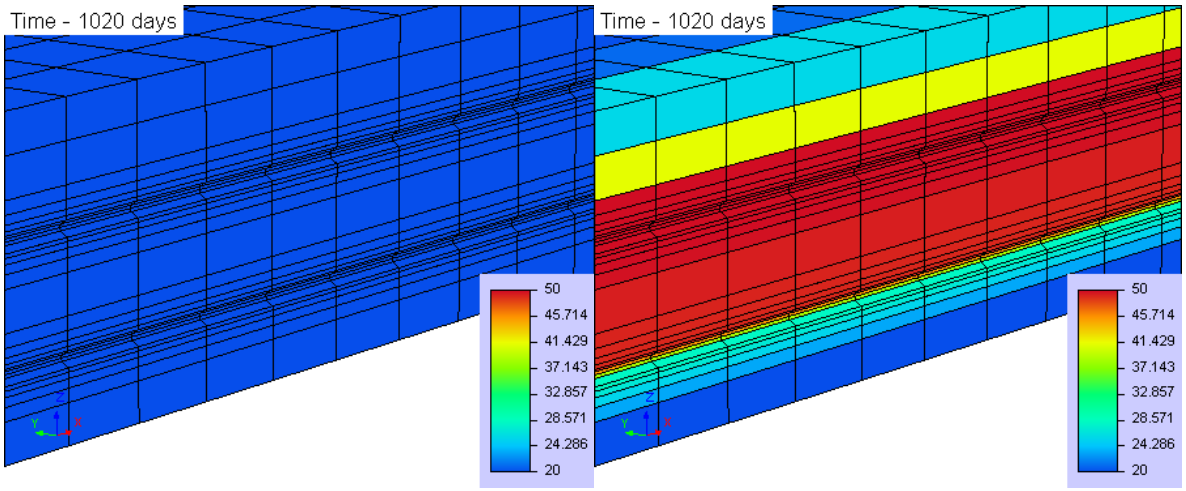

Рисунок 16 – Поле температуры нефти; слева – при T = 20℃, справа – при T = 50℃

3.6 ИСПОЛЬЗОВАНИЕ ГОРЯЧЕЙ ВОДЫ РАЗНОЙ ТЕМПЕРАТУРЫ ДЛЯ УВЕЛИЧЕНИЯ НЕФТЕОТДАЧИ

Возьмем модельную задачу из пункта 3.5 и будем менять температуру закачиваемой смеси – 50, 100, 150 и 200℃. Можно моделировать процесс с температурой воды 150 и 200 градусов, так как при давлении 16 атм температура кипения воды составляет 200,4℃.

На рисунке 17 представлен график суммарного отбора нефти при разных температурах закачиваемой жидкости.

В таблице 6 приведены суммарные объемы добытой нефти на конец моделируемого периода. Можно заметить, что изменение температуры на 30 градусов увеличивает результирующую добычу почти в 4 раза, нагрев еще на 50 градусов приводит к увеличению в 16 раз. А вот дальнейший нагрев на 50 градусов увеличивает добычу в примерно в 3 раза, а затем и в 1,33.

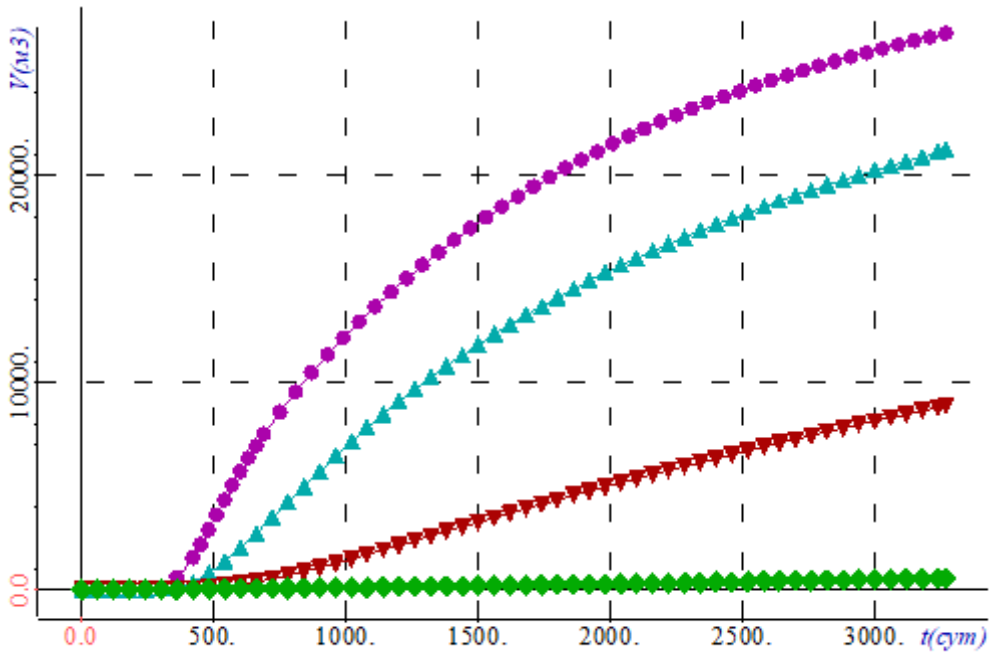

Рисунок 17 – Суммарный отбор нефти при разных температурах закачиваемой жидкости; зеленый – 50°С, красный – 100°С, голубой – 150°С и розовый – 200°С

Такое неравномерное увеличение отбора связано с нелинейной зависимостью вязкости нефти от температуры. Как можно увидеть из таблицы 5, от 20 до 100 °С вязкость изменяется примерно в 120 раз, а от 100 до 260 – примерно в 20 раз.

Таблица 6 – Сравнение суммарных добытых объемов нефти при разной температуре закачиваемой жидкости

Температура закачиваемой жидкости (T_i), °С	Суммарный добытый объем нефти ($V_{T_i}^{oil\ prod}$), м^3	$\frac{V_{T_i}^{oil\ prod}}{V_{T_{i-1}}^{oil\ prod}}$	$\frac{T_i}{T_{i-1}}$
20	139,942	-	-
50	558,232	3,9890	2,5
100	9373,89	16,7921	2
150	26569,3	2,8344	1,5
200	35563,4	1,3385	1,3333

При применении теплового метода на практике важно учитывать соотношение затрат на нагрев закачиваемой смеси и соответствующее увеличение отбора, что не рассмотрено в модельной задаче.

На рисунке 18 представлено процентное содержание нефти в отбираемой смеси при разных температурах закачиваемой воды.

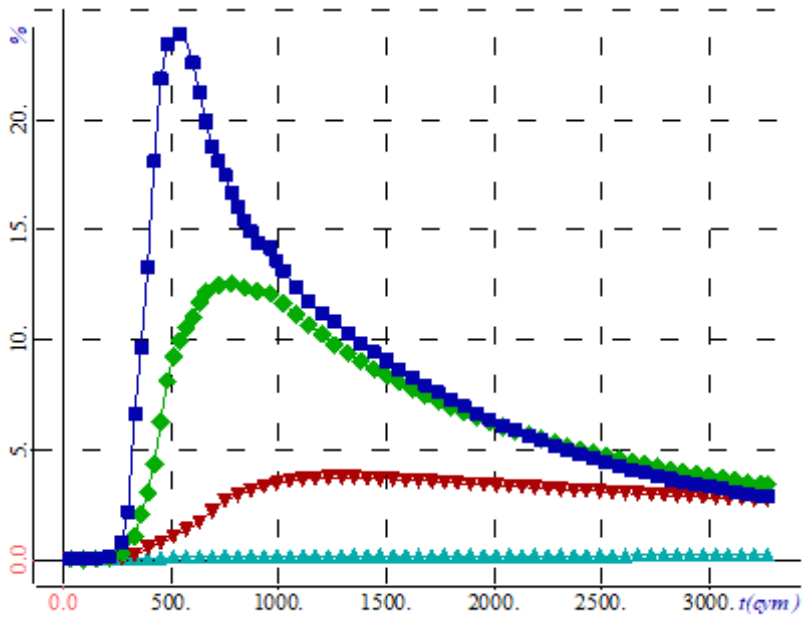

Рисунок 18 – Процент нефти в отбираемой смеси

Так как объем отбираемой смеси одинаков для всех запусков, соотношение процентов нефти в смеси равно соотношению мгновенного отбора нефти. На графике видны максимумы доли нефти в смеси – это отражает тот факт, что нефть около добывающей скважины нагрелась и имеет более низкую вязкость. Спад после пика связан с тем, что горячая вода от нагнетательной скважины сдвинулась к добывающей. Из графика видно, что для температуры закачки 50℃ доля нефти в отборе менее 2 процентов. Такую долю можно считать слишком малой, чтобы вести добычу.

ЗАКЛЮЧЕНИЕ

В рамках данной работы:

1. Для моделирования процесса многофазной фильтрации с использованием технологии горизонтальных скважин была рассмотрена математическая модель многофазной фильтрации с учетом зависимости вязкости флюида от температуры.
2. Был реализован программный модуль встройки горизонтальных скважин явным и неявным способами. В результате проведенных исследований было показано, что разные методы встраивания при задании потока смеси на скважинах дают отличие в результатах в среднем менее 2%, но явная встройка позволяет строить сетку с меньшим количеством элементов в простых случаях, в то время как неявная встройка выгодна при малом размере зон перфорации и в случаях сложных моделей с несогласованными сетками.
3. Был реализован модуль расчета поля давления: решение эллиптической краевой задачи с использованием трилинейного базиса на шестигранниках. Для вычисления интегралов на шестигранниках был реализован модуль интегрирования методом Гаусс-3 с оптимизацией на основе особенностей подынтегральных функций заранее известного вида. В результате исследований для реализованной вычислительной схемы был подтвержден соответствующий порядок сходимости и аппроксимации.
4. Были проведены исследования влияния температуры закачиваемой смеси на объем добываемой нефти и продемонстрирована польза такого метода увеличения нефтеотдачи пласта. В дальнейшем возможно изучение экономической выгоды при различных планах закачивания теплоносителя в пласт.

СПИСОК ЛИТЕРАТУРЫ

1. Керимов В.Ю., Бахтизин Р.Н., Данцова К.И. [и др.] Моделирование месторождений и залежей нефти и газа для решения задач разведки и разработки // Транспорт и хранение нефтепродуктов и углеводородного сырья, 2018. №3. С. 52–56
2. Сургучев М.П., Горбунов А.Т., Забродин Д.И. Методы извлечения остаточной нефти. Москва: Недра, 1991. 347 с.
3. Chaar M., Venetos M., Dargin J., et al. Economics Of Steam Generation For Thermal Enhanced Oil Recovery // Oil and Gas Facilities. 2015. Vol. 4, № 6. P. 42–50.
4. Mokheimer E.M.A., Hamdy M., Abubakar Z., et al. A comprehensive review of thermal enhanced oil recovery: Techniques evaluation // Journal of Energy Resources Technology. 2019. Vol. 141, № 3.
5. Соловейчик Ю.Г., Рояк М.Э., Персова М.Г. Метод конечных элементов для решения скалярных и векторных задач // Новосибирск: НГТУ, 2007. 896 с.
6. Персова М. Г., Соловейчик Ю. Г., Вагин Д. В. [и др.] Программное обеспечение, реализующее работу с информационными массивами данных при решении задач геологоразведки и многофазной фильтрации // Свидетельство о государственной регистрации № 2018619455 от 07.08.2018 М.: Федеральная служба по интеллектуальной собственности (Роспатент). 2018.
7. Персова М. Г., Соловейчик Ю. Г., Овчинникова А. С. [и др.] Программный комплекс для гидродинамического моделирования FlowER // Свидетельство о государственной регистрации № 2019665615 от 26.11.2019 М.: Федеральная служба по интеллектуальной собственности (Роспатент). 2019.
8. Персова М. Г., Соловейчик Ю. Г., Овчинникова А. С. [и др.] HDPoM 2.0 (HydroDynamic in Porous Media) // Свидетельство о государственной ре-

гистрации №2021661751 от 15.07.2021 – М.: Федеральная служба по интеллектуальной собственности (Роспатент). – 2021.

9. M. G. Persova, Y. G. Soloveichik, A. M. Grif, I. I. Patrushev. Flow balancing in FEM modelling of multi-phase flow in porous media // Актуальные проблемы электронного приборостроения (АПЭП–2018): тр. 14 междунар. науч.-техн. конф., Новосибирск, 2–6 окт. 2018 г. : в 8 т. Новосибирск : Изд-во НГТУ, 2018. Т. 1, ч. 4. С. 205–211.
10. Персова М.Г., Соловейчик Ю.Г., Гриф А.М. Балансировка потоков на неконформных конечноэлементных сетках при моделировании многофазной фильтрации // Программная инженерия. – 2021. – Т.12, № 9. – С. 450-458.
11. Овчинникова А.С., Патрушев И.И., Гриф А.М. [и др.] Конечноэлементное моделирование многофазных потоков с их балансировкой при фиксировании рабочего давления на скважинах в процессе нефтедобычи // Вычислительные методы и программирование. 2022, Т. 23, №1. С. 60-74.
12. Персова М.Г., Соловейчик Ю.Г., Патрушев И.И., и др. Применение процедуры группирования конечных элементов для повышения эффективности моделирования нестационарного многофазного потока в высоконеоднородных трехмерных пористых средах // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2021. Т. 57. №4. С. 34-44.
13. Овчинникова А.С. Вычислительная схема для расчета температурного поля при решении задач нефтедобычи // Системы анализа и обработки данных. 2021. Т. 84. № 4. С. 37–48.
14. Хисамов Р.С., Назимов Н.А., Хайруллин М.Х., и др. Оценка профиля притока к стволу горизонтальной скважины по результатам термогидродинамических исследований // Нефтяное хозяйство, 2021. №12. С. 114–116.

ПРИЛОЖЕНИЕ. ФРАГМЕНТЫ ПРОГРАММЫ

Приведены основные функции реализованных модулей.

Модуль встройки горизонтальных скважин (явный способ)

```
int main()
{
	ifstream inf;
	inf.open("horizontal_well.txt");
	if (!inf)
	{
		cout << "No horizontal wells in model" << endl;
		return 0;
	}
	inf.close();
	inf.clear();

	MoveCoutToFile();
	cout << "Program start" << endl;

	MeshXYZ mesh;
	mesh.MeshRead("mesh");
	// чтение базовой сетки
	cout << "Mesh read" << endl;
	mesh.PropRead("properties");
	cout << "Properties read" << endl;

	// чтение параметров для горизонтальных скважин
	WellsRead("", mesh.hor_well);
	cout << "Horizontal wells read" << endl;

	// удаление
	if (mesh.DeleteElems() != 0)
	{
		cout << "Wells has intersection. Please use smaller phi_z or h_z" << endl;
		return 1;
	}
	cout << "Elements deleted" << endl;

// построение новых элементов
	if (mesh.BuildNewElems() != 0)
	{
		cout << " BuildNewElems on well " << 0 << " return 1" << endl;
		return 1;
	}
	else
	{
		// перенумерация
		mesh.RenumWithoutDeletedElements();
		cout << "Elements renumerated" << endl;

		// запись сетки обратно в файл
		mesh.MeshWrite("mesh");
		mesh.PropWrite("properties");
		mesh.PropWrite("output3D");

		// переписывание краевых
		mesh.RestartBC();
```

```
		mesh.NewBCWrite("mesh");

		cout << "All done" << endl;
	}
	return 0;
}
int MeshXYZ::DeleteElems()
{
	int el = 0;
	while (el < Elements.size())
	{
		pair<double, double> distanceToCenter_prev;
		Point3D centerOfElement_prev;
		for (int well_id = 0; well_id < hor_well.size(); well_id++)
		{
			Point3D centerOfElement;
			GetElementCenter(el, centerOfElement);

			pair<double, double> distanceToCenter;
			distanceToCenter = GetDistanceInPlain(hor_well[well_id],
centerOfElement);

			// удаляем ли элемент
			if (distanceToCenter.first < hor_well[well_id].wide_area &&
distanceToCenter.second < hor_well[well_id].wide_area_z)
			{
				if (well_id == 0)
				{
					int jjjj = 0;
				}
				elDel++;
				if (deletedElements.count(el))
				{
					cout << "Warring: areas have intersection\n";
					return 1;
				}
				deletedElements.insert(el);

				if (deletedElemNeibVec.count(el) == 0)
				{
					deletedElemNeibVec[el] = ElemNeibVec[el];
				}

				// удалить у граней связь с удаленным элементом
				for (int i = 0; i < 6; i++)
				{
					auto cur_face = &Faces[Elements[el].faces[i]];
					if (cur_face->elems.size() == 0)// ???
					{
						cout << "Error: face was deleted\n";
						return 1;
					}
					else if (cur_face->elems[0] == el)
					    cur_face->elems.erase(cur_face->elems.begin());
					else if (cur_face->elems.size() == 1) // ???
						{
							cout << "Error: face not associated with
element\n";

							return 1;
						}
					else
```

```
                                        cur_face->elems.erase(cur_face->elems.begin() +
1);

                                    // убрать связь между удаляемым элементом и его
соседями
                                    for (int j = 0; j < ElemNeibVec[el].neib[i].size();
j++)
                                    {
                                        int el_neib = ElemNeibVec[el].neib[i][j]; //
взять элемент-соседа
                                        int i_neib =
Elements[el_neib].faces2[Elements[el].faces[i]];
                                        if (deletedElemNeibVec.count(el_neib) == 0)
                                        {
                                            deletedElemNeibVec[el_neib] =
ElemNeibVec[el_neib];
                                        }
                                        // удаляемый элемент больше не сосед
                                        for (int k = 0; k <
ElemNeibVec[el_neib].neib[i_neib].size(); k++)
                                        {
                                            if (ElemNeibVec[el_neib].neib[i_neib][k]
== el)
                                            {

        ElemNeibVec[el_neib].neib[i_neib].erase(ElemNeibVec[el_neib].neib[i_neib].begin()
+ k);
                                            }
                                        }
                                    }

                                    // если надо, удалить грань (поменять макс.номер на
удаляемый)
                                    if (cur_face->elems.size() == 0 || (cur_face-
>elems.size() == 1 && cur_face->elems[0] == -1))
                                    {
                                        faceDel++;
                                        auto swap_face = &Faces[nfaces - faceDel];
                                        // удаление из краевых
                                        for (int j = 0; j < swap_face->elems.size(); j++)
                                        {
                                            if (swap_face->elems[j] != -1)
                                            {
                                                int loc_face = Elements[swap_face-
>elems[j]].faces2[nfaces - faceDel];
                                                Elements[swap_face-
>elems[j]].faces[loc_face] = Elements[el].faces[i];
                                                Elements[swap_face-
>elems[j]].faces2[Elements[el].faces[i]] = loc_face;
                                                Elements[swap_face-
>elems[j]].faces2.erase(nfaces - faceDel);

                                                auto tmp = make_pair(el, i);
                                                if (bc1_map.count(tmp))
                                                {
                                                    kolbc1faces--;
                                                    iter_swap(bc1faces.begin() +
bc1_map[tmp], bc1faces.begin() + kolbc1faces);
                                                    bc1_map.erase(tmp);
                                                }
                                                else if (bc2_map.count(tmp))
                                                {
```

```
                                                kolbc2faces--;
                                                iter_swap(bc2faces.begin() +
bc2_map[tmp], bc2faces.begin() + kolbc2faces);
                                                bc2_map.erase(tmp);
                                        }

                                    }
                                }

                            }

                        }

                        // убрать у узлов связь с удаленным элементом
                        for (int i = 0; i < 8; i++)
                        {
                            auto cur_point = &Points[Elements[el].node[i]];
                            for (int k = 0; k < cur_point->elems.size(); k++)
                            {
                                if (cur_point->elems[k] == el)
                                    cur_point->elems.erase(cur_point-
>elems.begin() + k);
                            }
                            if (cur_point->elems.size() == 0)
                            {
                                nodeDel++;
                            }
                        }
                    }
                    distanceToCenter_prev = distanceToCenter;
                    centerOfElement_prev = centerOfElement;
                }
                el++;
            }

            koluz_main -= nodeDel;
            koluz -= nodeDel;
            nfaces -= faceDel;
            ncFaces -= faceDel;

            return 0;
}
int MeshXYZ::BuildNewElems()
{
            for (int well_id = 0; well_id < hor_well.size(); well_id++)
            {
                cout << "start well " << well_id << endl;
                if (deletedElements.size() == 0)
                {
                    cout << "Error: elements wasn't deleted. Please use bigger phi_z or
phi_hor" << endl;
                    return 1;
                }

                bool find_start = false;

                int point_id = 0;  // какую пару точек добавляем
                int face_id = 0;   // по какой грани ищем соседей

                // порядок обхода граней/точек в зависимости от параллельности осям
                if (hor_well[well_id].start.crd[0] != hor_well[well_id].end.crd[0])
                {
```

```
                neib_faces[0] = 0;
                neib_faces[1] = 3;
                neib_faces[2] = 5;
                neib_faces[3] = 2;
                neib_faces[4] = 4;
                neib_faces[5] = 1;

                oder_nodes[0] = 2;
                oder_nodes[1] = 6;
                oder_nodes[2] = 4;
                oder_nodes[3] = 0;

        }
        else if (hor_well[well_id].start.crd[1] != hor_well[well_id].end.crd[1])
        {
                neib_faces[0] = 3;
                neib_faces[1] = 0;
                neib_faces[2] = 5;
                neib_faces[3] = 1;
                neib_faces[4] = 4;
                neib_faces[5] = 2;

                oder_nodes[0] = 2;
                oder_nodes[1] = 6;
                oder_nodes[2] = 7;
                oder_nodes[3] = 3;
        }

        auto iterator = deletedElements.begin();
        // найдем один из элементов 1-ого слоя
        for (int i = 0; i < deletedElements.size() && !find_start; i++)
        {
                Point3D centerOfElement;
                GetElementCenter((*iterator), centerOfElement);

                pair<double, double> distanceToCenter;
                distanceToCenter = GetDistanceInPlain(hor_well[well_id],
centerOfElement);

                if (distanceToCenter.first < hor_well[well_id].wide_area &&
distanceToCenter.second < hor_well[well_id].wide_area_z)
                {
                        if (ElemNeibVec[(*iterator)].neib[neib_faces[face_id]].size()
== 0)
                                find_start = true;
                        for (int j = 0; j <
ElemNeibVec[(*iterator)].neib[neib_faces[face_id]].size(); j++)
                        {
                                if
(deletedElements.count(ElemNeibVec[(*iterator)].neib[neib_faces[face_id]][j]) == 0)
                                        find_start = true;
                        }
                }
                if (!find_start)
                        iterator++;
        }
        if (iterator == deletedElements.end())
        {
                cout << "Error: can't find start elem for well " << well_id << ".
May be no one elem was deleted for this well.\n";
                return 1;
        }
```

```
		// найдем стартовый элемент "слоя"
		face_id = 1;
		bool find_bounder = false;
		int cur_elem = (*iterator);
		int start_el;
		// найдем крайний элемент в одном направлении
		int iteration2 = 0;
		while (!find_bounder || iteration2 > 10000)
		{
			if (ElemNeibVec[cur_elem].neib[neib_faces[face_id]].size() == 0)
				find_bounder = true;
			for (int j = 0; j < 
ElemNeibVec[cur_elem].neib[neib_faces[face_id]].size(); j++)
			{
				if 
(deletedElements.count(ElemNeibVec[cur_elem].neib[neib_faces[face_id]][j]) == 0)
					find_bounder = true;
			}
			if (!find_bounder)
				cur_elem = 
ElemNeibVec[cur_elem].neib[neib_faces[face_id]][0];
			iteration2++;
		}
		if (iteration2 == 10000)
		{
			cout << " can't find start elem " << endl;
			return 1;
		}

		start_el = cur_elem;
		vector<int> bounderPoints1, bounderPoints2;
		vector<pair<int, int>> bounderFaces1, bounderFaces2; // element (glob), 
face(loc)
		vector<Point3D> new_points1, new_points2;
		vector<Point3D> new_points0_1, new_points0_2;

		Point3D centre;

		// 1 слой
		if (GetBounderNodesOnLayer(start_el, bounderPoints1, bounderFaces1))
			return 1;
		cout << " GetBounderNodesOnLayer " << endl;
		GetNewNodesOnLayer(hor_well[well_id], bounderPoints1, new_points1);
		cout << " GetNewNodesOnLayer " << endl;

		PointsNew.insert(PointsNew.end(), new_points1.begin(), new_points1.end());
		new_points0_1.resize(2 * new_points1.size());

		cout << " prepare nodes " << endl;

		int knewuz_onlayer = bounderPoints1.size() * (hor_well[well_id].R_step);

		int layer = 1; // 0 - первый слой
		int newFacesCount = 0, newElemsCount = 0, newPointsCount = 
new_points1.size();


		while (ElemNeibVec[start_el].neib[neib_faces[5]].size() != 0)
		{
			cout << "layer " << layer << endl;
```

```
                    start_el = ElemNeibVec[start_el].neib[neib_faces[5]][0]; // перейти
на слой дальше
                    if (GetBounderNodesOnLayer(start_el, bounderPoints2, bounderFaces2))
                          return 1;

                    GetNewNodesOnLayer(hor_well[well_id], bounderPoints2, new_points2);

                    ElementsNew.reserve(new_points2.size());
                    PointsNew.insert(PointsNew.end(), new_points2.begin(),
new_points2.end()); // копирование Можно через ссылку ???
                    newPointsCount += new_points2.size();

                    // собрать элементы
                    for (int i = 0; i < new_points2.size(); i++)
                    {
                          ElemNeib tmp;
                          ElemNeibVecNew.push_back(tmp);

                          Element3D buf;
                          int R = hor_well[well_id].R_step;

                          /* о нумерации узлов*/
                          //buf.node[6] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i;                                 // new_points1[i]
                          //buf.node[7] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 ;                            // new_points1[i + 1]
                          //buf.node[2] = koluz + koluzNew + layer       *
knewuz_onlayer + i;                                 // new_points2[i]
                          //buf.node[3] = koluz + koluzNew + layer       *
knewuz_onlayer + i + 1;                             // new_points2[i + 1]
                          //buf.node[4] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i     + hor_well[well_id].R_step; // new_points1[i + R_step]
                          //buf.node[5] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points1[i + R_step + 1]
                          //buf.node[0] = koluz + koluzNew + layer       *
knewuz_onlayer + i     + hor_well[well_id].R_step; // new_points2[i + R_step]
                          //buf.node[1] = koluz + koluzNew + layer       *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points2[i + R_step + 1]

                          if ((i + R) >= new_points2.size()) // последняя линия по слою
                          {
                                buf.node[6] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i;
                                buf.node[2] = koluz + koluzNew + layer * knewuz_onlayer
+ i;
                                buf.node[4] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i - (new_points2.size() - hor_well[well_id].R_step);
                                buf.node[0] = koluz + koluzNew + layer * knewuz_onlayer
+ i - (new_points2.size() - hor_well[well_id].R_step);

                                if (i / R != (i + 1) / R) // последний узел по линии

                                {
                                      buf.node[7] = bounderPoints1[i / R];
                                      buf.node[3] = bounderPoints2[i / R];
                                      buf.node[5] = bounderPoints1[0];
                                      buf.node[1] = bounderPoints2[0];

                                      Points[bounderPoints1[i /
R]].elems.push_back(kolel - elDel + ElementsNew.size());
                                      Points[bounderPoints2[i /
R]].elems.push_back(kolel - elDel + ElementsNew.size());
```

```
                                    Points[bounderPoints1[0]].elems.push_back(kolel -
elDel + ElementsNew.size());
                                    Points[bounderPoints2[0]].elems.push_back(kolel -
elDel + ElementsNew.size());
                              }
                              else
                              {
                                    buf.node[7] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1;                        // new_points1[i + 1]
                                    buf.node[3] = koluz + koluzNew + layer *
knewuz_onlayer + i + 1;                         // new_points2[i + 1]
                                    buf.node[5] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 - (new_points2.size() - hor_well[well_id].R_step); //
new_points1[i + R_step + 1]
                                    buf.node[1] = koluz + koluzNew + layer *
knewuz_onlayer + i + 1 - (new_points2.size() - hor_well[well_id].R_step); //
new_points2[i + R_step + 1]
                              }
                         }
                         else
                         {
                                    buf.node[6] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i;                              // new_points1[i]
                              buf.node[2] = koluz + koluzNew + layer * knewuz_onlayer
+ i;                              // new_points2[i]
                              buf.node[4] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + hor_well[well_id].R_step; // new_points1[i + R_step]
                              buf.node[0] = koluz + koluzNew + layer * knewuz_onlayer
+ i + hor_well[well_id].R_step; // new_points2[i + R_step]

                              if (i / R != (i + 1) / R)  // последний узел по линии

                              {
                                    buf.node[7] = bounderPoints1[i / R];
                                    buf.node[3] = bounderPoints2[i / R];
                                    buf.node[5] = bounderPoints1[i / R + 1];
                                    buf.node[1] = bounderPoints2[i / R + 1];

                                    Points[bounderPoints1[i /
R]].elems.push_back(kolel - elDel + ElementsNew.size());
                                    Points[bounderPoints2[i /
R]].elems.push_back(kolel - elDel + ElementsNew.size());
                                    Points[bounderPoints1[i / R +
1]].elems.push_back(kolel - elDel + ElementsNew.size());
                                    Points[bounderPoints2[i / R +
1]].elems.push_back(kolel - elDel + ElementsNew.size());

                              }
                              else
                              {
                                    buf.node[7] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1;                        // new_points1[i + 1]
                                    buf.node[3] = koluz + koluzNew + layer *
knewuz_onlayer + i + 1;                         // new_points2[i + 1]
                                    buf.node[5] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points1[i + R_step + 1]
                                    buf.node[1] = koluz + koluzNew + layer *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points2[i + R_step + 1]

                              }
                         }
```

```
                        // in which element
                        GetElementCenter(buf, centre);
                        bool find_elem = false;
                        auto iterator2 = deletedElements.begin();
                        for (; iterator2 != deletedElements.end() && !find_elem; )
                        {
                              if (PointInElem(Elements[(*iterator2)], centre))
                                    find_elem = true;
                              else
                                    iterator2++;

                        }
                        if (iterator2 == deletedElements.end())
                        {
                              cout << "Something get wrong" << endl;
                              return 1;
                        }
                        auto cur_elem = &Elements[(*iterator2)];
                        buf.nmat = cur_elem->nmat;
                        buf.layer = cur_elem->layer;
                        layers[cur_elem->layer].push_back(ElementsNew.size());
                        buf.phasecomp_hi = cur_elem->phasecomp_hi;
                        buf.S0 = cur_elem->S0;
                        buf.bfelem = cur_elem->bfelem;
                        buf.Tbase = cur_elem->Tbase;
                        buf.Tph0 = cur_elem->Tph0;
                        buf.well_id = cur_elem->well_id;

                        /* Points0 */
                        double x0min = Points0[cur_elem->node[0]].crd[0];
                        double y0min = Points0[cur_elem->node[0]].crd[1];
                        double z0min = Points0[cur_elem->node[0]].crd[2];
                        double x0max = Points0[cur_elem->node[1]].crd[0];
                        double y0max = Points0[cur_elem->node[2]].crd[1];
                        double z0max = Points0[cur_elem->node[4]].crd[2];
                        double xmin = Points[cur_elem->node[0]].crd[0];
                        double ymin = Points[cur_elem->node[0]].crd[1];
                        double zmin = Points[cur_elem->node[0]].crd[2];
                        double xmax = Points[cur_elem->node[1]].crd[0];
                        double ymax = Points[cur_elem->node[2]].crd[1];
                        double zmax = Points[cur_elem->node[4]].crd[2];

                        double koef_x = (x0max - x0min) / (xmax - xmin);
                        double koef_y = (y0max - y0min) / (ymax - ymin);
                        double koef_z = (z0max - z0min) / (zmax - zmin);

                        double start_x = x0max + koef_x * xmin;
                        double start_y = y0max + koef_y * ymin;
                        double start_z = z0max + koef_z * zmin;

                        if (layer == 1)
                        {
                              Points0New.reserve(knewuz_onlayer);
                              for (int i = 0; i < knewuz_onlayer; i++)
                              {
                                    double xp = start_x - koef_x *
new_points1[i].crd[0];
                                    double yp = start_y - koef_y *
new_points1[i].crd[1];
                                    double zp = start_z - koef_z *
new_points1[i].crd[2];
                                    Points0New.push_back(Point3D(xp, yp, zp));
```

```
                        }

                    }

                    Points0New.reserve(knewuz_onlayer);
                    for (int i = 0; i < new_points2.size(); i++)

                    {
                        double xp = start_x - koef_x * new_points2[i].crd[0];
                        double yp = start_y - koef_y * new_points2[i].crd[1];
                        double zp = start_z - koef_z * new_points2[i].crd[2];
                        Points0New.push_back(Point3D(xp, yp, zp));
                    }

                    /* о нумерации граней */
                    // buf.faces[0] = ; // или новое (если на row нет элемента
предыдущего), или 1 грань предыдущего элем.
                    // buf.faces[1] =;  // или новое, или из Faces
                    // buf.faces[2] =;  //новое или 5 грань (row + 1) элемента
                    // buf.faces[3] =;  // или новое или 4 грань (row - 1)
элемента
                    // buf.faces[4] =;  // новое
                    // buf.faces[5] =;  //или новое, или 3 грань из (layer - 1)
элемента

                    // 0
                    if (i % R == 0)
                    {
                        buf.faces[0] = nfaces + nfacesNew + newFacesCount; //
или новое (если на row нет элемента предыдущего), или 1 грань предыдущего элем.

                        // проверка, из holes ли грань
                        // если точка между 0 - 2 входит в начало-конец holes,
то грань оттуда
                        double cent_point;
                        if (hor_well[well_id].start.crd[0] !=
hor_well[well_id].end.crd[0])
                            cent_point = (new_points2[0].crd[0] +
new_points1[0].crd[0]) / 2;
                        else if (hor_well[well_id].start.crd[0] !=
hor_well[well_id].end.crd[0])
                            cent_point = (new_points2[0].crd[1] -
new_points1[0].crd[1]) / 2;
                        else if (hor_well[well_id].start.crd[0] !=
hor_well[well_id].end.crd[0])
                            cent_point = (new_points2[0].crd[2] -
new_points1[0].crd[2]) / 2;
                        for (int ih = 0; ih < hor_well[well_id].kol_holes;
ih++)
                        {
                            if (hor_well[well_id].holes[ih].hole_start <
cent_point && hor_well[well_id].holes[ih].hole_end > cent_point)
                            {
                                hor_well[well_id].holes[ih].faces[kolel -
elDel + ElementsNew.size()] = 0;
                                hor_well[well_id].holes[ih].kol_faces++;
                            }
                        }

                        newFacesCount++;
                    }
                    else
```

```
                        {
                                buf.faces[0] = ElementsNew[ElementsNew.size() -
1].faces[1];

                                ElemNeibVecNew[ElementsNew.size() -
1].neib[1].resize(1);
                                ElemNeibVecNew[ElementsNew.size() - 1].neib[1][0] =
kolel - elDel + ElementsNew.size();

                                ElemNeibVecNew[ElementsNew.size()].neib[0].resize(1);
                                ElemNeibVecNew[ElementsNew.size()].neib[0][0] = kolel
- elDel + ElementsNew.size() - 1;
                        }


                        // 1
                        if ((i + 1) % R == 0)
                        {
                                int old_face_glob = Elements[bounderFaces1[i /
R].first].faces[bounderFaces1[i / R].second];
                                buf.faces[1] = old_face_glob;
                                // лок.номер грани удаленного элемента
                                if (deletedElemNeibVec[bounderFaces1[i /
R].first].neib[bounderFaces1[i / R].second].size() != 0)
                                {
                                        int uu = deletedElemNeibVec.count(5048);
                                        for (int y = 0; y < bounderFaces1.size(); y++)
                                        {
                                                if (bounderFaces1[y].first == 5048)
                                                        cout << endl;
                                        }
                                        int neib_elem =
deletedElemNeibVec[bounderFaces1[i / R].first].neib[bounderFaces1[i / R].second][0];
                                        int neib_face =
Elements[neib_elem].faces2[old_face_glob];


        //ElemNeibVec[neib_elem].neib[neib_face].resize(1);
                                        //ElemNeibVec[neib_elem].neib[neib_face][0] =
kolel - elDel + ElementsNew.size();

        ElemNeibVec[neib_elem].neib[neib_face].reserve(1);

        ElemNeibVec[neib_elem].neib[neib_face].push_back(kolel - elDel +
ElementsNew.size());


        ElemNeibVecNew[ElementsNew.size()].neib[1].resize(1);
                                        ElemNeibVecNew[ElementsNew.size()].neib[1][0] =
neib_elem;
                                }
                        }
                        else
                        {
                                buf.faces[1] = nfaces + nfacesNew + newFacesCount;
                                newFacesCount++;
                        }

                        // 2
                        if ((i + R) >= new_points2.size())
                        {
```

```
                                    buf.faces[2] = ElementsNew[ElementsNew.size() - R *
(bounderPoints1.size() - 1)].faces[3];

                                    ElemNeibVecNew[ElementsNew.size() - R *
(bounderPoints1.size() - 1)].neib[3].resize(1);
                                    ElemNeibVecNew[ElementsNew.size() - R *
(bounderPoints1.size() - 1)].neib[3][0] = kolel - elDel + ElementsNew.size();

                                    ElemNeibVecNew[ElementsNew.size()].neib[2].resize(1);
                                    ElemNeibVecNew[ElementsNew.size()].neib[2][0] = kolel -
elDel + ElementsNew.size() - R * (bounderPoints1.size() - 1);
                              }
                              else
                              {
                                    buf.faces[2] = nfaces + nfacesNew + newFacesCount;
                                    newFacesCount++;
                              }

                              // 3
                              if (i / R == 0)
                              {
                                    buf.faces[3] = nfaces + nfacesNew + newFacesCount; //
новое
                                    newFacesCount++;
                              }
                              else
                              {
                                    buf.faces[3] = ElementsNew[ElementsNew.size() -
R].faces[2];

                                    ElemNeibVecNew[ElementsNew.size() -
R].neib[2].resize(1);
                                    ElemNeibVecNew[ElementsNew.size() - R].neib[2][0] =
kolel - elDel + ElementsNew.size();

                                    ElemNeibVecNew[ElementsNew.size()].neib[3].resize(1);
                                    ElemNeibVecNew[ElementsNew.size()].neib[3][0] = kolel -
elDel + ElementsNew.size() - R;

                              }

                              // 4
                              buf.faces[4] = nfaces + nfacesNew + newFacesCount; // новое
                              newFacesCount++;

                              // 5
                              if (layer < 2)
                              {
                                    if
(ElemNeibVec[(*iterator2)].neib[neib_faces[0]].size() != 0)
                                    {
                                          cout << "Horizontal well " << well_id << " isn't
in whole layer. Change coordinates start and end" << endl;
                                          return 1;

                                          int neib_elem =
ElemNeibVec[(*iterator2)].neib[neib_faces[0]][0];


      ElemNeibVec[neib_elem].neib[neib_faces[5]].reserve(1);
```

```
	ElemNeibVec[neib_elem].neib[neib_faces[5]].push_back(kolel - elDel +
ElementsNew.size());


	ElemNeibVecNew[ElementsNew.size()].neib[5].resize(1);
					ElemNeibVecNew[ElementsNew.size()].neib[5][0] =
neib_elem;

					buf.faces[5] =
Elements[neib_elem].faces[neib_faces[5]];
				}
				else
				{
					buf.faces[5] = nfaces + nfacesNew +
newFacesCount;
					newFacesCount++;
				}
			}
			else
			{
				buf.faces[5] = ElementsNew[ElementsNew.size() - R *
bounderPoints1.size()].faces[4];

				ElemNeibVecNew[ElementsNew.size() - R *
bounderPoints1.size()].neib[4].resize(1);
				ElemNeibVecNew[ElementsNew.size() - R *
bounderPoints1.size()].neib[4][0] = kolel - elDel + ElementsNew.size();

				ElemNeibVecNew[ElementsNew.size()].neib[5].resize(1);
				ElemNeibVecNew[ElementsNew.size()].neib[5][0] = kolel -
elDel + ElementsNew.size() - R * bounderPoints1.size();
			}

			ElementsNew.push_back(buf);
			newElemsCount++;
		}

		bounderPoints1.swap(bounderPoints2);
		bounderFaces1.swap(bounderFaces2);
		new_points1.swap(new_points2);

		layer++;
	}

	// порядок обхода граней/точек в зависимости от параллельности осям для
последнего элемента
	if (hor_well[well_id].start.crd[0] != hor_well[well_id].end.crd[0])
	{
		oder_nodes[0] = 3;
		oder_nodes[1] = 7;
		oder_nodes[2] = 5;
		oder_nodes[3] = 1;

	}
	else if (hor_well[well_id].start.crd[1] != hor_well[well_id].end.crd[1])
	{
		oder_nodes[0] = 0;
		oder_nodes[1] = 4;
		oder_nodes[2] = 5;
		oder_nodes[3] = 1;
	}
```

```
		if (GetBounderNodesOnLayer(start_el, bounderPoints2, bounderFaces2))
			return 1;
		GetNewNodesOnLayer(hor_well[well_id], bounderPoints2, new_points2);
		PointsNew.insert(PointsNew.end(), new_points2.begin(), new_points2.end());

		ElementsNew.reserve(new_points2.size());
		newPointsCount += new_points2.size();

		for (int i = 0; i < new_points2.size(); i++)
		{
			ElemNeib tmp;
			ElemNeibVecNew.push_back(tmp);

			Element3D buf;
			int R = hor_well[well_id].R_step;

			/* о нумерации узлов*/
				//buf.node[6] = koluz  + koluzNew + (layer - 1) *
knewuz_onlayer + i;                                 // new_points1[i]
				//buf.node[7] = koluz  + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 ;                            // new_points1[i + 1]
				//buf.node[2] = koluz  + koluzNew + layer       *
knewuz_onlayer + i;                                 // new_points2[i]
				//buf.node[3] = koluz  + koluzNew + layer       *
knewuz_onlayer + i + 1;                             // new_points2[i + 1]
				//buf.node[4] = koluz  + koluzNew + (layer - 1) *
knewuz_onlayer + i     + hor_well[well_id].R_step; // new_points1[i + R_step]
				//buf.node[5] = koluz  + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points1[i + R_step + 1]
				//buf.node[0] = koluz  + koluzNew + layer       *
knewuz_onlayer + i     + hor_well[well_id].R_step; // new_points2[i + R_step]
				//buf.node[1] = koluz  + koluzNew + layer       *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points2[i + R_step + 1]

			if ((i + R) >= new_points2.size()) // последняя линия по слою
			{
				buf.node[6] = koluz + koluzNew + (layer - 1) * knewuz_onlayer
+ i;
				buf.node[2] = koluz + koluzNew + layer * knewuz_onlayer + i;
				buf.node[4] = koluz + koluzNew + (layer - 1) * knewuz_onlayer
+ i - (new_points2.size() - hor_well[well_id].R_step);
				buf.node[0] = koluz + koluzNew + layer * knewuz_onlayer + i -
(new_points2.size() - hor_well[well_id].R_step);

				if (i / R != (i + 1) / R)  // последний узел по линии
				{
					buf.node[7] = bounderPoints1[i / R];
					buf.node[3] = bounderPoints2[i / R];
					buf.node[5] = bounderPoints1[0];
					buf.node[1] = bounderPoints2[0];

					Points[bounderPoints1[i / R]].elems.push_back(kolel -
elDel + ElementsNew.size());
					Points[bounderPoints2[i / R]].elems.push_back(kolel -
elDel + ElementsNew.size());
					Points[bounderPoints1[0]].elems.push_back(kolel - elDel
+ ElementsNew.size());
					Points[bounderPoints2[0]].elems.push_back(kolel - elDel
+ ElementsNew.size());
				}
				else
```

```
                        {
                              buf.node[7] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1;                         // new_points1[i + 1]
                              buf.node[3] = koluz + koluzNew + layer * knewuz_onlayer
+ i + 1;                          // new_points2[i + 1]
                              buf.node[5] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 - (new_points2.size() - hor_well[well_id].R_step); //
new_points1[i + R_step + 1]
                              buf.node[1] = koluz + koluzNew + layer * knewuz_onlayer
+ i + 1 - (new_points2.size() - hor_well[well_id].R_step); // new_points2[i + R_step +
1]
                        }
                  }
                  else
                  {
                        buf.node[6] = koluz + koluzNew + (layer - 1) * knewuz_onlayer
+ i;                              // new_points1[i]
                        buf.node[2] = koluz + koluzNew + layer * knewuz_onlayer + i;
// new_points2[i]
                        buf.node[4] = koluz + koluzNew + (layer - 1) * knewuz_onlayer
+ i + hor_well[well_id].R_step; // new_points1[i + R_step]
                        buf.node[0] = koluz + koluzNew + layer * knewuz_onlayer + i +
hor_well[well_id].R_step; // new_points2[i + R_step]

                        if (i / R != (i + 1) / R)  // последний узел по линии
                        {
                              buf.node[7] = bounderPoints1[i / R];
                              buf.node[3] = bounderPoints2[i / R];
                              buf.node[5] = bounderPoints1[i / R + 1];
                              buf.node[1] = bounderPoints2[i / R + 1];

                              Points[bounderPoints1[i / R]].elems.push_back(kolel -
elDel + ElementsNew.size());
                              Points[bounderPoints2[i / R]].elems.push_back(kolel -
elDel + ElementsNew.size());
                              Points[bounderPoints1[i / R + 1]].elems.push_back(kolel
- elDel + ElementsNew.size());
                              Points[bounderPoints2[i / R + 1]].elems.push_back(kolel
- elDel + ElementsNew.size());

                        }
                        else
                        {
                              buf.node[7] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1;                         // new_points1[i + 1]
                              buf.node[3] = koluz + koluzNew + layer * knewuz_onlayer
+ i + 1;                          // new_points2[i + 1]
                              buf.node[5] = koluz + koluzNew + (layer - 1) *
knewuz_onlayer + i + 1 + hor_well[well_id].R_step; // new_points1[i + R_step + 1]
                              buf.node[1] = koluz + koluzNew + layer * knewuz_onlayer
+ i + 1 + hor_well[well_id].R_step; // new_points2[i + R_step + 1]

                        }
                  }

                  // in which element
                  GetElementCenter(buf, centre);
                  bool find_elem = false;
                  auto iterator3 = deletedElements.begin();
                  for (; iterator3 != deletedElements.end() && !find_elem; )
                  {
                        if (PointInElem(Elements[(*iterator3)], centre))
```

```
                        find_elem = true;
                    else
                        iterator3++;

                }
                if (iterator3 == deletedElements.end())
                {
                    cout << "Something get wrong" << endl;
                    return 1;
                }
                auto cur_elem = &Elements[(*iterator3)];
                buf.nmat = cur_elem->nmat;
                buf.layer = cur_elem->layer;
                layers[cur_elem->layer].push_back(ElementsNew.size());
                buf.phasecomp_hi = cur_elem->phasecomp_hi;
                buf.S0 = cur_elem->S0;
                buf.bfelem = cur_elem->bfelem;
                buf.Tbase = cur_elem->Tbase;
                buf.Tph0 = cur_elem->Tph0;
                buf.well_id = cur_elem->well_id;

                if (layer == 1)
                {
                    Point3D_t p;
                    p[0] = new_points1[i].crd[0];
                    p[1] = new_points1[i].crd[1];
                    p[2] = new_points1[i].crd[2];

                    auto rez = ComputeLocalPointFromGlobalPoint(p, *this,
*iterator3);

                    Points0New.push_back(Point3D(rez[0], rez[1], rez[2]));

                }
                Points0New.reserve(knewuz_onlayer);
                for (int i = 0; i < knewuz_onlayer; i++)
                {
                    Point3D_t p;
                    p[0] = new_points2[i].crd[0];
                    p[1] = new_points2[i].crd[1];
                    p[2] = new_points2[i].crd[2];

                    auto rez = ComputeLocalPointFromGlobalPoint(p, *this,
*iterator3);

                    Points0New.push_back(Point3D(rez[0], rez[1], rez[2]));
                }

                /* о нумерации граней */
                    // buf.faces[0] = ; // или новое (если на row нет элемента
предыдущего), или 1 грань предыдущего элем.
                    // buf.faces[1] =;  // или новое, или из Faces
                    // buf.faces[2] =;  //новое или 5 грань (row + 1) элемента
                    // buf.faces[3] =;  // или новое или 4 грань (row - 1)
элемента
                    // buf.faces[4] =;  // новое
                    // buf.faces[5] =;  //или новое, или 3 грань из (layer - 1)
элемента

                    // 0
                if (i % R == 0)
                {
```

```
                        buf.faces[0] = nfaces + nfacesNew + newFacesCount; // или
новое (если на row нет элемента предыдущего), или 1 грань предыдущего элем.

                        // проверка, из holes ли грань
                        // если точка между 0 - 2 входит в начало-конец holes, то
грань оттуда
                        double cent_point;
                        if (hor_well[well_id].start.crd[0] !=
hor_well[well_id].end.crd[0])
                              cent_point = (new_points2[0].crd[0] +
new_points1[0].crd[0]) / 2;
                        else if (hor_well[well_id].start.crd[0] !=
hor_well[well_id].end.crd[0])
                              cent_point = (new_points2[0].crd[1] -
new_points1[0].crd[1]) / 2;
                        else if (hor_well[well_id].start.crd[0] !=
hor_well[well_id].end.crd[0])
                              cent_point = (new_points2[0].crd[2] -
new_points1[0].crd[2]) / 2;
                        for (int ih = 0; ih < hor_well[well_id].kol_holes; ih++)
                        {
                              if (hor_well[well_id].holes[ih].hole_start < cent_point
&& hor_well[well_id].holes[ih].hole_end > cent_point)
                              {
                                    hor_well[well_id].holes[ih].faces[kolel - elDel +
ElementsNew.size()] = 0;
                                    hor_well[well_id].holes[ih].kol_faces++;
                              }
                        }

                        newFacesCount++;
                  }
                  else
                  {
                        buf.faces[0] = ElementsNew[ElementsNew.size() - 1].faces[1];

                        ElemNeibVecNew[ElementsNew.size() - 1].neib[1].resize(1);
                        ElemNeibVecNew[ElementsNew.size() - 1].neib[1][0] = kolel -
elDel + ElementsNew.size();

                        ElemNeibVecNew[ElementsNew.size()].neib[0].resize(1);
                        ElemNeibVecNew[ElementsNew.size()].neib[0][0] = kolel - elDel
+ ElementsNew.size() - 1;
                  }


                  // 1
                  if ((i + 1) % R == 0)
                  {
                        int old_face_glob = Elements[bounderFaces1[i /
R].first].faces[bounderFaces1[i / R].second];
                        buf.faces[1] = old_face_glob;
                        // лок.номер грани удаленного элемента
                        if (deletedElemNeibVec[bounderFaces1[i /
R].first].neib[bounderFaces1[i / R].second].size() != 0)
                        {
                              int neib_elem = deletedElemNeibVec[bounderFaces1[i /
R].first].neib[bounderFaces1[i / R].second][0];
                              int neib_face =
Elements[neib_elem].faces2[old_face_glob];

                              ElemNeibVec[neib_elem].neib[neib_face].reserve(1);
```

```
                                ElemNeibVec[neib_elem].neib[neib_face].push_back(kolel
- elDel + ElementsNew.size());

                                ElemNeibVecNew[ElementsNew.size()].neib[1].resize(1);
                                ElemNeibVecNew[ElementsNew.size()].neib[1][0] =
neib_elem;
                        }
                }
                else
                {
                        buf.faces[1] = nfaces + nfacesNew + newFacesCount;
                        newFacesCount++;
                }

                // 2
                if ((i + R) >= new_points2.size())
                {
                        buf.faces[2] = ElementsNew[ElementsNew.size() - R *
(bounderPoints1.size() - 1)].faces[3];

                        ElemNeibVecNew[ElementsNew.size() - R *
(bounderPoints1.size() - 1)].neib[3].resize(1);
                        ElemNeibVecNew[ElementsNew.size() - R *
(bounderPoints1.size() - 1)].neib[3][0] = kolel - elDel + ElementsNew.size();

                        ElemNeibVecNew[ElementsNew.size()].neib[2].resize(1);
                        ElemNeibVecNew[ElementsNew.size()].neib[2][0] = kolel - elDel
+ ElementsNew.size() - R * (bounderPoints1.size() - 1);
                }
                else
                {
                        buf.faces[2] = nfaces + nfacesNew + newFacesCount;
                        newFacesCount++;
                }

                // 3
                if (i / R == 0)
                {
                        buf.faces[3] = nfaces + nfacesNew + newFacesCount; // новое
                        newFacesCount++;
                }
                else
                {
                        buf.faces[3] = ElementsNew[ElementsNew.size() - R].faces[2];

                        ElemNeibVecNew[ElementsNew.size() - R].neib[2].resize(1);
                        ElemNeibVecNew[ElementsNew.size() - R].neib[2][0] = kolel -
elDel + ElementsNew.size();

                        ElemNeibVecNew[ElementsNew.size()].neib[3].resize(1);
                        ElemNeibVecNew[ElementsNew.size()].neib[3][0] = kolel - elDel
+ ElementsNew.size() - R;

                }

                // 4
                buf.faces[4] = nfaces + nfacesNew + newFacesCount; // новое
                newFacesCount++;

                // 5
                if (layer < 2)
                {
```

```
                        if (ElemNeibVec[(*iterator3)].neib[neib_faces[0]].size() !=
0)
                        {
                                cout << "Horizontal well " << well_id << " isn't in
whole layer. Change coordinates start and end" << endl;
                                return 1;

                                int neib_elem =
ElemNeibVec[(*iterator3)].neib[neib_faces[0]][0];

                                ElemNeibVec[neib_elem].neib[neib_faces[5]].reserve(1);

      ElemNeibVec[neib_elem].neib[neib_faces[5]].push_back(kolel - elDel +
ElementsNew.size());

                                ElemNeibVecNew[ElementsNew.size()].neib[5].resize(1);
                                ElemNeibVecNew[ElementsNew.size()].neib[5][0] =
neib_elem;

                                buf.faces[5] =
Elements[neib_elem].faces[neib_faces[5]];
                        }
                        else
                        {
                                buf.faces[5] = nfaces + nfacesNew + newFacesCount;
                                newFacesCount++;
                        }
                }
                else
                {
                        buf.faces[5] = ElementsNew[ElementsNew.size() - R *
bounderPoints1.size()].faces[4];

                        ElemNeibVecNew[ElementsNew.size() - R *
bounderPoints1.size()].neib[4].resize(1);
                        ElemNeibVecNew[ElementsNew.size() - R *
bounderPoints1.size()].neib[4][0] = kolel - elDel + ElementsNew.size();

                        ElemNeibVecNew[ElementsNew.size()].neib[5].resize(1);
                        ElemNeibVecNew[ElementsNew.size()].neib[5][0] = kolel - elDel
+ ElementsNew.size() - R * bounderPoints1.size();
                }

                // если после вырезанного участка еще есть элементы
                if (ElemNeibVec[(*iterator3)].neib[neib_faces[5]].size() != 0)
                {
                        int neib_elem =
ElemNeibVec[(*iterator3)].neib[neib_faces[5]][0];

                        ElemNeibVec[neib_elem].neib[neib_faces[0]].resize(1);
                        ElemNeibVec[neib_elem].neib[neib_faces[0]][0] = kolel - elDel
+ ElementsNew.size();

                        ElemNeibVecNew[ElementsNew.size()].neib[4].resize(1);
                        ElemNeibVecNew[ElementsNew.size()].neib[4][0] =
ElemNeibVec[(*iterator3)].neib[neib_faces[0]][0];
                }

                ElementsNew.push_back(buf);
                newElemsCount++;
        }
```

```
			kolelNew += newElemsCount;
			koluzNew += newPointsCount;
			nfacesNew += newFacesCount;
		}

		return 0;
}
int MeshXYZ::RenumWithoutDeletedElements()
{
		RenumNodes();

		elDel = 0;
		int old_kel = kolel - deletedElements.size();
		int all_delelems = deletedElements.size();
		for (int el = 0; el < old_kel; el++)
		{
			// удаляем ли элемент
			if (deletedElements.count(el))
			{
				// первый - элемент, который убираем
				// второй - элемент, который в конце сетки, с которым "меняем"
				// у соседей второго элемента сменим связь на ассоциацию с первым
элементом
				for (int face_id = 0; face_id < 6; face_id++)
				{
					auto tmp = make_pair(old_kel + elDel, face_id);
					if (bc1_map.count(tmp))
					{
						bc1faces[bc1_map[tmp]].el = el;
					}
					else if (bc2_map.count(tmp))
					{
						bc2faces[bc2_map[tmp]].el = el;
					}

					if (Elements[old_kel + elDel].well_id != make_pair(-1, -1))
					{
						auto id = Elements[old_kel + elDel].well_id;
						auto cur_hole = &old_wells[id.first].holes[id.second];
						cur_hole->faces[el] = cur_hole->faces[old_kel + elDel];
						cur_hole->faces.erase(old_kel + elDel);
						for (int gr_id = 0; gr_id <
old_wells[id.first].holes[id.second].groups_count; gr_id++)
						{
							if (cur_hole->groups[gr_id].faces.count(old_kel +
elDel) != 0)
							{
								cur_hole->groups[gr_id].faces[el] =
cur_hole->groups[gr_id].faces[old_kel + elDel];
								cur_hole-
>groups[gr_id].faces.erase(old_kel + elDel);
							}
						}
					}

					for (int j = 0; j < ElemNeibVec[old_kel +
elDel].neib[face_id].size(); j++)
					{
						int el_neib = ElemNeibVec[old_kel +
elDel].neib[face_id][j]; // взять элемент-соседа
						if (el_neib < kolel)
```

```
                                {
                                        int i_neib =
Elements[el_neib].faces2[Elements[old_kel + elDel].faces[face_id]];
                                    // перемещаемый элемент теперь сосед
                                        for (int j = 0; j <
ElemNeibVec[el_neib].neib[i_neib].size(); j++)
                                        {

                                                if (ElemNeibVec[el_neib].neib[i_neib][j]
== old_kel + elDel)
                                                {
                                                        ElemNeibVec[el_neib].neib[i_neib][j]
= el;
                                                }
                                        }
                                }
                                else
                                {
                                        el_neib -= old_kel;
                                        int i_neib = 1;//delface_to_neibface[i];
                                    // перемещаемый элемент теперь сосед
                                        for (int j = 0; j <
ElemNeibVecNew[el_neib].neib[i_neib].size(); j++)
                                        {

                                                if
(ElemNeibVecNew[el_neib].neib[i_neib][j] == old_kel + elDel)
                                                {

        ElemNeibVecNew[el_neib].neib[i_neib][j] = el;
                                                }
                                        }

                                }

                        }
                }

                // меняем местами
                std::iter_swap(Elements.begin() + el, Elements.begin() + (old_kel +
elDel));
                std::swap(ElemNeibVec[el], ElemNeibVec[old_kel + elDel]);

                deletedElements.erase(el);
                elDel++;
        }
    }

    kolel_main -= elDel;
    kolel -= elDel;

    for (int i = 0; i < layersCount; i++)
    {
        layers[i].reserve(kolel - deletedElements.size());
    }
    for (int el = 0; el < kolel - deletedElements.size(); el++)
    {
        layers[Elements[el].layer].push_back(el);
    }

    return 0;
}
```

```
int MeshXYZ::RenumNodes()
{
	for (int el = 0; el < kolel; el++)
	{
		for (int i = 0; i < 8; i++)
		{
			auto cur_point = &Points[Elements[el].node[i]];
			if (cur_point->elems.size() == 0)
			{
				nodeDel--;
				auto swap_node = &Points[koluz + nodeDel];
				for (int j = 0; j < swap_node->elems.size(); j++)
				{
					int loc_node = Elements[swap_node-
>elems[j]].nodes2[koluz + nodeDel];
					Elements[swap_node->elems[j]].node[loc_node] =
Elements[el].node[i];
					swap_node->elems[j] = el;
				}
				iter_swap(Points.begin() + (koluz + nodeDel), Points.begin()
+ (Elements[el].node[i]));
				iter_swap(Points0.begin() + (koluz + nodeDel),
Points0.begin() + (Elements[el].node[i]));
			}
		}
	}
	return 0;
}
```

Модуль встройки горизонтальных скважин (неявный способ)

```
// блок, формирующий дополнительные узлы для построения несогласованной сетки
int main()
{
	ifstream inf;
	 inf.open("horizontal_well.txt");
	if (!inf)
	{
		cout << "No horizontal wells in model"<< endl;
		inf.close();
		inf.clear();
		inf.open("add_area.txt");
		if (!inf)
		{
			cout << "No areas in model" << endl;
			return 0;
		}
	}
	inf.close();
	inf.clear();

	MoveCoutToFile();
	cout << "Program start" << endl;

	MeshXYZ mesh;
	mesh.MeshRead("mesh\\");
	cout << "Mesh read" << endl;

	mesh.WellsRead("");
	cout << "Horizontal wells read" << endl;
```

```
    // запись сетки обратно в файл
      mesh.MeshWrite("mesh\\");

      cout << "All done" << endl;
      return 0;
}
int MeshXYZ::WellsRead(string pathInput)
{
      ifstream inf;
      int size, count_holes;
      if (pathInput != "") inf.open(pathInput + "/horizontal_well.txt");
      else                 inf.open("horizontal_well.txt");
      if (!inf)
      {
            cout << "Can't open " << ("horizontal_well.txt") << endl;
            return 0;
      }
      inf >> size;
      hor_well.resize(size);
      double phi, phi_z;
      for (int i = 0; i < size; i++)
      {
            inf >> hor_well[i].start.crd[0] >> hor_well[i].start.crd[1] >>
hor_well[i].start.crd[2]
                  >> hor_well[i].end.crd[0] >> hor_well[i].end.crd[1] >>
hor_well[i].end.crd[2]
                  >> hor_well[i].radius
                  >> hor_well[i].R_step >> hor_well[i].R_koef >> phi
                  >> hor_well[i].R_koef_z >> phi_z;
            hor_well[i].wide_area = 2 * hor_well[i].radius + hor_well[i].radius * phi;
            hor_well[i].wide_area_z = 2 * hor_well[i].radius + hor_well[i].radius *
phi_z;

            inf >> count_holes;

            hor_well[i].holes.resize(count_holes);
            hor_well[i].kol_holes = count_holes;

            if (hor_well[i].start.crd[0] != hor_well[i].end.crd[0])
            {
                  for (int j = 0; j < count_holes; j++)
                  {
                        auto cur_hole = &hor_well[i].holes[j];
                        inf >> cur_hole->hole_start >> cur_hole->hole_end;

                        cur_hole->start_area.resize(hor_well[i].R_step+1);
                        cur_hole->end_area.resize(hor_well[i].R_step+1);

                        double h, h_z;
                        if (hor_well[i].R_koef == 1)
                        {
                              h = (hor_well[i].wide_area - hor_well[i].radius) /
hor_well[i].R_step;
                              h_z = (hor_well[i].wide_area_z - hor_well[i].radius) /
hor_well[i].R_step;
                              cur_hole->start_area[0].crd[0] = cur_hole->hole_start;
                              cur_hole->start_area[0].crd[1] =
hor_well[i].start.crd[1] - hor_well[i].radius;
                              cur_hole->start_area[0].crd[2] =
hor_well[i].start.crd[2] - hor_well[i].radius;

                              cur_hole->end_area[0].crd[0] = cur_hole->hole_end;
```

```
                                cur_hole->end_area[0].crd[1] = hor_well[i].end.crd[1] +
hor_well[i].radius;
                                cur_hole->end_area[0].crd[2] = hor_well[i].end.crd[2] +
hor_well[i].radius;

                                for (int p = 1; p < hor_well[i].R_step; p++)
                                {
                                        cur_hole->start_area[p].crd[0] = cur_hole-
>start_area[0].crd[0];
                                        cur_hole->start_area[p].crd[1] = cur_hole-
>start_area[0].crd[1] - h * p;
                                        cur_hole->start_area[p].crd[2] = cur_hole-
>start_area[0].crd[2] - h_z * p;

                                        cur_hole->end_area[p].crd[0] = cur_hole-
>end_area[0].crd[0];
                                        cur_hole->end_area[p].crd[1] = cur_hole-
>end_area[0].crd[1] + h * p;
                                        cur_hole->end_area[p].crd[2] = cur_hole-
>end_area[0].crd[2] + h_z * p;
                                }
                        }
                        else
                        {
                                h = (hor_well[i].wide_area - hor_well[i].radius) *
(hor_well[i].R_koef - 1) / (pow(hor_well[i].R_koef, hor_well[i].R_step) - 1);
                                h_z = (hor_well[i].wide_area_z - hor_well[i].radius) *
(hor_well[i].R_koef_z - 1) / (pow(hor_well[i].R_koef_z, hor_well[i].R_step) - 1);

                                cur_hole->start_area[0].crd[0] = cur_hole->hole_start;
                                cur_hole->start_area[0].crd[1] =
hor_well[i].start.crd[1] - hor_well[i].radius;
                                cur_hole->start_area[0].crd[2] =
hor_well[i].start.crd[2] - hor_well[i].radius;

                                cur_hole->end_area[0].crd[0] = cur_hole->hole_end;
                                cur_hole->end_area[0].crd[1] = hor_well[i].end.crd[1] +
hor_well[i].radius;
                                cur_hole->end_area[0].crd[2] = hor_well[i].end.crd[2] +
hor_well[i].radius;

                                for (int p = 0; p < hor_well[i].R_step - 1; p++)
                                {
                                        cur_hole->start_area[p+1].crd[0] = cur_hole-
>start_area[0].crd[0];
                                        cur_hole->start_area[p+1].crd[1] = cur_hole-
>start_area[p].crd[1] - h * pow(hor_well[i].R_koef, p);
                                        cur_hole->start_area[p+1].crd[2] = cur_hole-
>start_area[p].crd[2] - h_z * pow(hor_well[i].R_koef_z, p);

                                        cur_hole->end_area[p+1].crd[0] = cur_hole-
>end_area[0].crd[0];
                                        cur_hole->end_area[p+1].crd[1] = cur_hole-
>end_area[p].crd[1] + h * pow(hor_well[i].R_koef, p);
                                        cur_hole->end_area[p+1].crd[2] = cur_hole-
>end_area[p].crd[2] + h_z * pow(hor_well[i].R_koef_z, p);
                                }
                        }
                        cur_hole->start_area[hor_well[i].R_step].crd[0] =
hor_well[i].start.crd[0];
                        cur_hole->start_area[hor_well[i].R_step].crd[1] =
hor_well[i].start.crd[1] - hor_well[i].wide_area;
```

```
                        cur_hole->start_area[hor_well[i].R_step].crd[2] =
hor_well[i].start.crd[2] - hor_well[i].wide_area_z;

                        cur_hole->end_area[hor_well[i].R_step].crd[0] =
hor_well[i].end.crd[0];
                        cur_hole->end_area[hor_well[i].R_step].crd[1] =
hor_well[i].end.crd[1] + hor_well[i].wide_area;
                        cur_hole->end_area[hor_well[i].R_step].crd[2] =
hor_well[i].end.crd[2] + hor_well[i].wide_area_z;
                  }
                  FormNodes(hor_well[i]);
            }

            if (hor_well[i].start.crd[1] != hor_well[i].end.crd[1])
            {
                  for (int j = 0; j < count_holes; j++)
                  {
                        auto cur_hole = &hor_well[i].holes[j];
                        inf >> cur_hole->hole_start >> cur_hole->hole_end;

                        cur_hole->start_area.resize(hor_well[i].R_step + 1);
                        cur_hole->end_area.resize(hor_well[i].R_step + 1);

                        double h, h_z;
                        if (hor_well[i].R_koef == 1)
                        {
                              h = (hor_well[i].wide_area - hor_well[i].radius) /
hor_well[i].R_step;
                              h_z = (hor_well[i].wide_area_z - hor_well[i].radius) /
hor_well[i].R_step;
                              cur_hole->start_area[0].crd[0] =
hor_well[i].start.crd[0] - hor_well[i].radius;
                              cur_hole->start_area[0].crd[1] = cur_hole->hole_start;
                              cur_hole->start_area[0].crd[2] =
hor_well[i].start.crd[2] - hor_well[i].radius;

                              cur_hole->end_area[0].crd[0] = hor_well[i].end.crd[0] +
hor_well[i].radius;
                              cur_hole->end_area[0].crd[1] = cur_hole->hole_end;
                              cur_hole->end_area[0].crd[2] = hor_well[i].end.crd[2] +
hor_well[i].radius;

                              for (int p = 1; p < hor_well[i].R_step; p++)
                              {
                                    cur_hole->start_area[p].crd[0] = cur_hole-
>start_area[0].crd[0] - h * p;
                                    cur_hole->start_area[p].crd[1] = cur_hole-
>start_area[0].crd[1];
                                    cur_hole->start_area[p].crd[2] = cur_hole-
>start_area[0].crd[2] - h_z * p;

                                    cur_hole->end_area[p].crd[0] = cur_hole-
>end_area[0].crd[0] + h * p;
                                    cur_hole->end_area[p].crd[1] = cur_hole-
>end_area[0].crd[1];
                                    cur_hole->end_area[p].crd[2] = cur_hole-
>end_area[0].crd[2] + h_z * p;
                              }
                        }
                        else
                        {
```

```
                                h = (hor_well[i].wide_area - hor_well[i].radius) *
(hor_well[i].R_koef - 1) / (pow(hor_well[i].R_koef, hor_well[i].R_step) - 1);
                                h_z = (hor_well[i].wide_area_z - hor_well[i].radius) *
(hor_well[i].R_koef_z - 1) / (pow(hor_well[i].R_koef_z, hor_well[i].R_step) - 1);

                                cur_hole->start_area[0].crd[0] =
hor_well[i].start.crd[0] - hor_well[i].radius;
                                cur_hole->start_area[0].crd[1] = cur_hole->hole_start;
                                cur_hole->start_area[0].crd[2] =
hor_well[i].start.crd[2] - hor_well[i].radius;

                                cur_hole->end_area[0].crd[0] = hor_well[i].end.crd[0] +
hor_well[i].radius;
                                cur_hole->end_area[0].crd[1] = cur_hole->hole_end;
                                cur_hole->end_area[0].crd[2] = hor_well[i].end.crd[2] +
hor_well[i].radius;

                                for (int p = 0; p < hor_well[i].R_step - 1; p++)
                                {
                                        cur_hole->start_area[p + 1].crd[0] = cur_hole-
>start_area[0].crd[0] - h * pow(hor_well[i].R_koef, p);
                                        cur_hole->start_area[p + 1].crd[1] = cur_hole-
>start_area[p].crd[1];
                                        cur_hole->start_area[p + 1].crd[2] = cur_hole-
>start_area[p].crd[2] - h_z * pow(hor_well[i].R_koef_z, p);

                                        cur_hole->end_area[p + 1].crd[0] = cur_hole-
>end_area[0].crd[0] + h * pow(hor_well[i].R_koef, p);
                                        cur_hole->end_area[p + 1].crd[1] = cur_hole-
>end_area[p].crd[1];
                                        cur_hole->end_area[p + 1].crd[2] = cur_hole-
>end_area[p].crd[2] + h_z * pow(hor_well[i].R_koef_z, p);
                                }
                        }
                        cur_hole->start_area[hor_well[i].R_step].crd[0] =
hor_well[i].start.crd[0] - hor_well[i].wide_area;
                        cur_hole->start_area[hor_well[i].R_step].crd[1] = cur_hole-
>hole_start;
                        cur_hole->start_area[hor_well[i].R_step].crd[2] =
hor_well[i].start.crd[2] - hor_well[i].wide_area_z;

                        cur_hole->end_area[hor_well[i].R_step].crd[0] =
hor_well[i].end.crd[0] + hor_well[i].wide_area;
                        cur_hole->end_area[hor_well[i].R_step].crd[1] = cur_hole-
>hole_end;
                        cur_hole->end_area[hor_well[i].R_step].crd[2] =
hor_well[i].end.crd[2] + hor_well[i].wide_area_z;
                }
                FormNodes(hor_well[i]);
        }
    }
    inf.close();
    inf.clear();
    return 0;
}
int MeshXYZ::FormNodes(Well &well)
{
    if (well.holes.size() == 0)
        return 0;

    int layer_x1, layer_y1, layer_z1;
    int layer_x2, layer_y2, layer_z2;
```

```
	double x0, y0, z0;

	if (well.start.crd[0] != well.end.crd[0])
	{
		for (int i_h = 0; i_h < well.kol_holes; i_h++)
		{
			well.holes[i_h].nx.resize(2);
			well.holes[i_h].ny.resize(2 * (well.R_step + 1));
			well.holes[i_h].nz.resize(2 * (well.R_step + 1));
		}

		layer_x1 = GetLayer_x(well.holes[0].start_area[0].crd[0]);
		x0 = GetShablon(well.holes[0].start_area[0].crd[0], basenodes_x[layer_x1],
basenodes_x[layer_x1 + 1]);
		nodes_x[layer_x1].push_back(well.holes[0].start_area[0].crd[0]);
		nodes_x0[layer_x1].push_back(x0);
		kn_x[layer_x1] += 1;
		for (int i_h = 0; i_h < well.kol_holes; i_h++)
		{
			well.holes[i_h].nx[0] = make_pair(kn_x[layer_x1], layer_x1);
		}

		layer_x2 = GetLayer_x(well.holes[0].end_area[0].crd[0]);
		x0 = GetShablon(well.holes[0].end_area[0].crd[0], basenodes_x[layer_x2],
basenodes_x[layer_x2 + 1]);
		nodes_x[layer_x2].push_back(well.holes[0].end_area[0].crd[0]);
		nodes_x0[layer_x2].push_back(x0);
		kn_x[layer_x2] += 1;
		for (int i_h = 0; i_h < well.kol_holes; i_h++)
		{
			well.holes[i_h].nx[1] = make_pair(kn_x[layer_x2], layer_x2);
		}

		for (int i_h = 0; i_h < well.kol_holes; i_h++)
		{
			auto cur_hole = &well.holes[i_h];

			for (int i_r = well.R_step; i_r >= 0; i_r--)
			{
				layer_y1 = GetLayer_y(cur_hole->start_area[i_r].crd[1]);
				y0 = GetShablon(cur_hole->start_area[i_r].crd[1],
basenodes_y[layer_y1], basenodes_y[layer_y1 + 1]);
				nodes_y[layer_y1].push_back(cur_hole-
>start_area[i_r].crd[1]);
				nodes_y0[layer_y1].push_back(y0);
				kn_y[layer_y1] += 1;
				cur_hole->ny[well.R_step - i_r] = make_pair(kn_y[layer_y1],
layer_y1);
			}

			for (int i_r = 0; i_r <= well.R_step; i_r++)
			{
				layer_y2 = GetLayer_y(cur_hole->end_area[i_r].crd[1]);
				y0 = GetShablon(cur_hole->end_area[i_r].crd[1],
basenodes_y[layer_y2], basenodes_y[layer_y2 + 1]);
				nodes_y[layer_y2].push_back(cur_hole->end_area[i_r].crd[1]);
				nodes_y0[layer_y2].push_back(y0);
				kn_y[layer_y2] += 1;
				cur_hole->ny[well.R_step + i_r + 1] =
make_pair(kn_y[layer_y2], layer_y2);
			}
```

```
                  for (int i_r = well.R_step; i_r >= 0; i_r--)
                  {
                        layer_z1 = GetLayer_z(cur_hole->start_area[i_r].crd[2]);
                        z0 = GetShablon(cur_hole->start_area[i_r].crd[2],
basenodes_z[layer_z1], basenodes_z[layer_z1 + 1]);
                        nodes_z[layer_z1].push_back(cur_hole-
>start_area[i_r].crd[2]);
                        nodes_z0[layer_z1].push_back(z0);
                        kn_z[layer_z1] += 1;
                        cur_hole->nz[well.R_step - i_r] = make_pair(kn_z[layer_z1],
layer_z1);
                  }

                  for (int i_r = 0; i_r <= well.R_step; i_r++)
                  {
                        layer_z2 = GetLayer_z(cur_hole->end_area[i_r].crd[2]);
                        z0 = GetShablon(cur_hole->end_area[i_r].crd[2],
basenodes_z[layer_z2], basenodes_z[layer_z2 + 1]);
                        nodes_z[layer_z2].push_back(cur_hole->end_area[i_r].crd[2]);
                        nodes_z0[layer_z2].push_back(z0);
                        kn_z[layer_z2] += 1;
                        cur_hole->nz[well.R_step + i_r + 1] =
make_pair(kn_z[layer_z2], layer_z2);
                  }
            }
      }

      if (well.start.crd[1] != well.end.crd[1])
      {
            for (int i_h = 0; i_h < well.kol_holes; i_h++)
            {
                  well.holes[i_h].nx.resize(2 * (well.R_step + 1));
                  well.holes[i_h].ny.resize(2);
                  well.holes[i_h].nz.resize(2 * (well.R_step + 1));
            }

            layer_y1 = GetLayer_y(well.holes[0].start_area[0].crd[1]);
            y0 = GetShablon(well.holes[0].start_area[0].crd[1], basenodes_y[layer_y1],
basenodes_y[layer_y1 + 1]);
            nodes_y[layer_y1].push_back(well.holes[0].start_area[0].crd[1]);
            nodes_y0[layer_y1].push_back(y0);
            kn_y[layer_y1] += 1;
            for (int i_h = 0; i_h < well.kol_holes; i_h++)
            {
                  well.holes[i_h].ny[0] = make_pair(kn_y[layer_y1], layer_y1);
            }

            layer_y2 = GetLayer_y(well.holes[0].end_area[0].crd[1]);
            y0 = GetShablon(well.holes[0].end_area[0].crd[1], basenodes_y[layer_y2],
basenodes_y[layer_y2 + 1]);
            nodes_y[layer_y2].push_back(well.holes[0].end_area[0].crd[1]);
            nodes_y0[layer_y2].push_back(y0);
            kn_y[layer_y2] += 1;
            for (int i_h = 0; i_h < well.kol_holes; i_h++)
            {
                  well.holes[i_h].ny[1] = make_pair(kn_y[layer_y2], layer_y2);
            }

            for (int i_h = 0; i_h < well.kol_holes; i_h++)
            {
                  auto cur_hole = &well.holes[i_h];
```

```
                    for (int i_r = well.R_step; i_r >= 0; i_r--)
                    {
                          layer_x1 = GetLayer_x(cur_hole->start_area[i_r].crd[0]);
                          x0 = GetShablon(cur_hole->start_area[i_r].crd[0],
basenodes_x[layer_x1], basenodes_x[layer_x1 + 1]);
                          nodes_x[layer_x1].push_back(cur_hole-
>start_area[i_r].crd[0]);
                          nodes_x0[layer_x1].push_back(x0);
                          kn_x[layer_x1] += 1;
                          cur_hole->nx[well.R_step - i_r] = make_pair(kn_x[layer_x1],
layer_x1);
                    }

                    for (int i_r = 0; i_r <= well.R_step; i_r++)
                    {
                          layer_x2 = GetLayer_x(cur_hole->end_area[i_r].crd[0]);
                          x0 = GetShablon(cur_hole->end_area[i_r].crd[0],
basenodes_x[layer_x2], basenodes_x[layer_x2 + 1]);
                          nodes_x[layer_x2].push_back(cur_hole->end_area[i_r].crd[0]);
                          nodes_x0[layer_x2].push_back(x0);
                          kn_x[layer_x2] += 1;
                          cur_hole->nx[well.R_step + i_r + 1] =
make_pair(kn_x[layer_x2], layer_x2);
                    }

                    for (int i_r = well.R_step; i_r >= 0; i_r--)
                    {
                          layer_z1 = GetLayer_z(cur_hole->start_area[i_r].crd[2]);
                          z0 = GetShablon(cur_hole->start_area[i_r].crd[2],
basenodes_z[layer_z1], basenodes_z[layer_z1 + 1]);
                          nodes_z[layer_z1].push_back(cur_hole-
>start_area[i_r].crd[2]);
                          nodes_z0[layer_z1].push_back(z0);
                          kn_z[layer_z1] += 1;
                          cur_hole->nz[well.R_step - i_r] = make_pair(kn_z[layer_z1],
layer_z1);
                    }

                    for (int i_r = 0; i_r <= well.R_step; i_r++)
                    {
                          layer_z2 = GetLayer_z(cur_hole->end_area[i_r].crd[2]);
                          z0 = GetShablon(cur_hole->end_area[i_r].crd[2],
basenodes_z[layer_z2], basenodes_z[layer_z2 + 1]);
                          nodes_z[layer_z2].push_back(cur_hole->end_area[i_r].crd[2]);
                          nodes_z0[layer_z2].push_back(z0);
                          kn_z[layer_z2] += 1;
                          cur_hole->nz[well.R_step + i_r + 1] =
make_pair(kn_z[layer_z2], layer_z2);
                    }
              }
        }

        return 0;
}

// блок, помечающий нужные грани несогласованной как грани зоны перфорации
int main()
{
        MoveCoutToFile();
        cout << "Program start" << endl;

        ifstream inf;
```

```
	inf.open("horizontal_well.txt");
	if (!inf)
	{
		cout << "No horizontal wells in model" << endl;
		return 0;
	}
	inf.close();
	inf.clear();

	MeshXYZ mesh;
	mesh.MeshRead("mesh");
	cout << "Mesh read" << endl;
	mesh.PropRead("properties");
	cout << "Properties read" << endl;

	mesh.WellsRead("");
	cout << "Horizontal wells read" << endl;

	mesh.func();

    // запись сетки обратно в файл
	mesh.MeshWrite("mesh");

	cout << "All done" << endl;
	return 0;
}
nt MeshXYZ::WellsRead(string pathInput)
{
	ifstream inf;
	int size, count_holes;
	if (pathInput != "") inf.open(pathInput + "/horizontal_well.txt");
	else 		inf.open("horizontal_well.txt");
	if (!inf)
	{
		cout << "Can't open " << ("horizontal_well.txt") << endl;
	}
	inf >> size;
	hor_well.resize(size);
	double phi, phi_z;
	for (int i = 0; i < size; i++)
	{
		inf >> hor_well[i].start.crd[0] >> hor_well[i].start.crd[1] >>
hor_well[i].start.crd[2]
			>> hor_well[i].end.crd[0] >> hor_well[i].end.crd[1] >>
hor_well[i].end.crd[2]
			>> hor_well[i].radius >> hor_well[i].R_step >> hor_well[i].R_koef >>
phi
			>> hor_well[i].R_koef_z >> phi_z;
		hor_well[i].wide_area = 2 * hor_well[i].radius + hor_well[i].radius * phi;
		hor_well[i].wide_area_z = 2 * hor_well[i].radius + hor_well[i].radius *
phi_z;

		inf >> count_holes;
		hor_well[i].holes.resize(count_holes);
		hor_well[i].kol_holes = count_holes;
		for (int j = 0; j < count_holes; j++)
		{
			inf >> hor_well[i].holes[j].hole_start >>
hor_well[i].holes[j].hole_end;
		}
	}
	inf.close();
```

```
	inf.clear();

	vector<int> x_num(basenodes_x.size()), y_num(basenodes_y.size()),
z_num(basenodes_z.size());
	vector<double> nodes_x, nodes_y, nodes_z;
	int num;
	double dbl_tmp;
	inf.open("mesh\\DataIntPG");
	if (!inf)
	{
		cout << "Can't read DataIntP\n";
	}
	else
	{
		for (int i = 0; i < basenodes_x.size(); i++)
			inf >> x_num[i];
		nodes_x.resize(x_num[basenodes_x.size() - 1]);

		for (int i = 0; i < basenodes_x.size() - 1; i++)
		{
			for (int j = x_num[i]; j < x_num[i + 1]; j++)
			{
				inf >> nodes_x[j - 1]>> dbl_tmp;
				nodes_x[j - 1] += i;
			}
		}

		for (int i = 0; i < basenodes_y.size(); i++)
			inf >> y_num[i];
		nodes_y.resize(y_num[basenodes_y.size() - 1]);

		for (int i = 0; i < basenodes_y.size() - 1; i++)
		{
			for (int j = y_num[i]; j < y_num[i + 1]; j++)
			{
				inf >> nodes_y[j - 1] >> dbl_tmp;
				nodes_y[j - 1] += i;
			}
		}

		for (int i = 0; i < basenodes_z.size(); i++)
			inf >> z_num[i];
		nodes_z.resize(z_num[basenodes_z.size() - 1]);

		for (int i = 0; i < basenodes_z.size() - 1; i++)
		{
			for (int j = z_num[i]; j < z_num[i + 1]; j++)
			{
				inf >> nodes_z[j - 1] >> dbl_tmp;
				nodes_z[j - 1] += i;
			}
		}

		inf >> num;
		int x1, x2, y1, y2, z1, z2;
		for (int i_w = 0; i_w < hor_well.size(); i_w++)
		{
			for (int i_h = 0; i_h < hor_well[i_w].holes.size(); i_h++)
			{
				inf >> x1 >> x2 >> y1 >> y2 >> z1 >> z2 >> num;
				hor_well[i_w].holes[i_h].start_area.resize(1);
				hor_well[i_w].holes[i_h].end_area.resize(1);
```

```
                        hor_well[i_w].holes[i_h].start_area[0].crd[0] = nodes_x[x1 -
1];
                        hor_well[i_w].holes[i_h].start_area[0].crd[1] = nodes_y[y1 -
1];
                        hor_well[i_w].holes[i_h].start_area[0].crd[2] = nodes_z[z1 -
1];
                        hor_well[i_w].holes[i_h].end_area[0].crd[0] = nodes_x[x2 -
1];
                        hor_well[i_w].holes[i_h].end_area[0].crd[1] = nodes_y[y2 -
1];
                        hor_well[i_w].holes[i_h].end_area[0].crd[2] = nodes_z[z2 -
1];
                }
            }
      }
      inf.close();
      inf.clear();

      return 0;
}
int MeshXYZ::func()
{
      for (int i = 0; i < kolel; i++)
      {
            Point3D centre0;
            GetElementCenter0(i, centre0);

            pair<int, int> well_hole = FindWell(centre0);
            if (well_hole.first != -1)
            {
                  for (int fc = 0; fc < 6; fc++)
                  {
                        if (ElemNeibVec[i].neib[fc].size() > 1)
                        {
                              cout << "Error: element " << i << " belongs well, but
has more than 1 neib on face " << fc << endl;
                              exit(1);
                        }
                        if (ElemNeibVec[i].neib[fc].size() == 0)
                        {
                              cout << "Waring: element " << i << " belongs well, but
hasn't neib on face " << fc << endl;
                              continue;
                        }
                        int neib = ElemNeibVec[i].neib[fc][0];
                        int neib_fc = Elements[neib].faces2[Elements[i].faces[fc]];

                        Point3D neib_centre0;
                        GetElementCenter0(neib, neib_centre0);

                        pair<int, int> neib_well_hole = FindWell(neib_centre0);
                        if (neib_well_hole.first == -1)
                        {
                              if (hor_well[well_hole.first].start.crd[0] !=
hor_well[well_hole.first].end.crd[0])
                              {
                                    if (neib_fc != 0 && neib_fc != 1)
                                    {

      hor_well[well_hole.first].holes[well_hole.second].faces[neib] = neib_fc;

      hor_well[well_hole.first].holes[well_hole.second].kol_faces++;
```

```
                                        }
                                }

                                if (hor_well[well_hole.first].start.crd[1] !=
hor_well[well_hole.first].end.crd[1])
                                {
                                        if (neib_fc != 2 && neib_fc != 3)
                                        {

        hor_well[well_hole.first].holes[well_hole.second].faces[neib] = neib_fc;

        hor_well[well_hole.first].holes[well_hole.second].kol_faces++;
                                        }
                                }
                        }
                }
        }
}
```

Модуль интегрирования

```
void Integrate_Gauss3Method3D::Init(vector<basis_func>& basis, vector<basis_func>&
derivative_basis)
{
        double ksi_x, ksi_y, ksi_z;

        int p_id = 0;
        for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)
                        for (int k = 0; k < 3; k++)
                        {
                                intagrate_points_3D[p_id].coord[0] = basic_point[i];
                                intagrate_points_3D[p_id].coord[1] = basic_point[j];
                                intagrate_points_3D[p_id].coord[2] = basic_point[k];
                                intagrate_points_3D[p_id].koef = basic_coef[i] *
basic_coef[j] * basic_coef[k];
                                ksi_x = basic_point[i];
                                ksi_y = basic_point[j];
                                ksi_z = basic_point[k];
                                for (int node_id = 0; node_id < 8; node_id++)
                                {
                                        // запомнить значения базисных функций и их производных
в точках шаблонного элемента
                                        intagrate_points_3D[p_id].derivative[node_id][0] =
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);
                                        intagrate_points_3D[p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);
                                        intagrate_points_3D[p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);
                                        intagrate_points_3D[p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
                                }
                                p_id++;
                        }

        double ksi, eta;
```

```
// запомнить вспомогатеьные величины для перевода между гранями и шаблонными гранями
	for (int face_id = 0; face_id < 6; face_id++)
	{
		p_id = 0;
		switch (face_id) // запоминаем направление нормали и меняющиеся координаты
шаблонной грани
		{
		case 0:
			for (int i = 0; i < 3; i++)
				for (int j = 0; j < 3; j++)
				{
					intagrate_points_face[face_id][p_id].coord[0] = -1;
					intagrate_points_face[face_id][p_id].coord[1] =
basic_point[i];
					intagrate_points_face[face_id][p_id].coord[2] =
basic_point[j];
					intagrate_points_face[face_id][p_id].koef =
basic_coef[i] * basic_coef[j];
					ksi_x = -1;
					ksi_y = basic_point[i];
					ksi_z = basic_point[j];
					for (int node_id = 0; node_id < 8; node_id++) // ???
зачем 8
					{

	intagrate_points_face[face_id][p_id].derivative[node_id][0] =
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

	intagrate_points_face[face_id][p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

	intagrate_points_face[face_id][p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);

	intagrate_points_face[face_id][p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
					}
					p_id++;
				}
			break;
		case 1:
			for (int i = 0; i < 3; i++)
				for (int j = 0; j < 3; j++)
				{
					intagrate_points_face[face_id][p_id].coord[0] = 1;
					intagrate_points_face[face_id][p_id].coord[1] =
basic_point[i];
					intagrate_points_face[face_id][p_id].coord[2] =
basic_point[j];
					intagrate_points_face[face_id][p_id].koef =
basic_coef[i] * basic_coef[j];
					ksi_x = 1;
					ksi_y = basic_point[i];
					ksi_z = basic_point[j];
					for (int node_id = 0; node_id < 8; node_id++)
					{

	intagrate_points_face[face_id][p_id].derivative[node_id][0] =
```

```
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
                              }
                              p_id++;
                        }
                  break;
            case 2:
                  for (int i = 0; i < 3; i++)
                        for (int j = 0; j < 3; j++)
                        {
                              intagrate_points_face[face_id][p_id].coord[0] =
basic_point[i];
                              intagrate_points_face[face_id][p_id].coord[1] = -1;
                              intagrate_points_face[face_id][p_id].coord[2] =
basic_point[j];
                              intagrate_points_face[face_id][p_id].koef =
basic_coef[i] * basic_coef[j];
                              ksi_x = basic_point[i];
                              ksi_y = -1;
                              ksi_z = basic_point[j];
                              for (int node_id = 0; node_id < 8; node_id++)
                              {

      intagrate_points_face[face_id][p_id].derivative[node_id][0] =
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
                              }
                              p_id++;
                        }
                  break;
            case 3:
                  for (int i = 0; i < 3; i++)
                        for (int j = 0; j < 3; j++)
                        {
                              intagrate_points_face[face_id][p_id].coord[0] =
basic_point[i];
                              intagrate_points_face[face_id][p_id].coord[1] = 1;
                              intagrate_points_face[face_id][p_id].coord[2] =
basic_point[j];
```

```
                                intagrate_points_face[face_id][p_id].koef =
basic_coef[i] * basic_coef[j];
                                ksi_x = basic_point[i];
                                ksi_y = 1;
                                ksi_z = basic_point[j];
                                for (int node_id = 0; node_id < 8; node_id++)
                                {

      intagrate_points_face[face_id][p_id].derivative[node_id][0] =
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
                                }
                                p_id++;
                          }
                    break;
             case 4:
                    for (int i = 0; i < 3; i++)
                          for (int j = 0; j < 3; j++)
                          {
                                intagrate_points_face[face_id][p_id].coord[0] =
basic_point[i];
                                intagrate_points_face[face_id][p_id].coord[1] =
basic_point[j];
                                intagrate_points_face[face_id][p_id].coord[2] = -1;
                                intagrate_points_face[face_id][p_id].koef =
basic_coef[i] * basic_coef[j];
                                ksi_x = basic_point[i];
                                ksi_y = basic_point[j];
                                ksi_z = -1;
                                for (int node_id = 0; node_id < 8; node_id++)
                                {

      intagrate_points_face[face_id][p_id].derivative[node_id][0] =
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
                                }
                                p_id++;
                          }
                    break;
             case 5:
```

```
                  for (int i = 0; i < 3; i++)
                        for (int j = 0; j < 3; j++)
                        {
                              intagrate_points_face[face_id][p_id].coord[0] =
basic_point[i];
                              intagrate_points_face[face_id][p_id].coord[1] =
basic_point[j];
                              intagrate_points_face[face_id][p_id].coord[2] = 1;
                              intagrate_points_face[face_id][p_id].koef =
basic_coef[i] * basic_coef[j];
                              ksi_x = basic_point[i];
                              ksi_y = basic_point[j];
                              ksi_z = 1;
                              for (int node_id = 0; node_id < 8; node_id++)
                              {

      intagrate_points_face[face_id][p_id].derivative[node_id][0] =
derivative_basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][1] =
basis[mu(node_id)](ksi_x) * derivative_basis[nu(node_id)](ksi_y) *
basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].derivative[node_id][2] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) *
derivative_basis[teta(node_id)](ksi_z);

      intagrate_points_face[face_id][p_id].basis_func[node_id] =
basis[mu(node_id)](ksi_x) * basis[nu(node_id)](ksi_y) * basis[teta(node_id)](ksi_z);
                              }
                              p_id++;
                        }
                  break;
            default:
                  break;
            }
      }
}
double Integrate_Gauss3Method3D::CalcJ_1(vector<vector<double>>& J_1_T, int p_id,
vector<Node3D> &element)
{
      vector<vector<double>> J(3, vector<double>(3));
      for (int i = 0; i < 8; i++)
      {
            J[0][0] += element[i].x * intagrate_points_3D[p_id].derivative[i][0]; //
d_xi[i];
            J[0][1] += element[i].x * intagrate_points_3D[p_id].derivative[i][1]; //
d_eta[i];
            J[0][2] += element[i].x * intagrate_points_3D[p_id].derivative[i][2]; //
d_zeta[i];

            J[1][0] += element[i].y * intagrate_points_3D[p_id].derivative[i][0]; //
d_xi[i];
            J[1][1] += element[i].y * intagrate_points_3D[p_id].derivative[i][1]; //
d_eta[i];
            J[1][2] += element[i].y * intagrate_points_3D[p_id].derivative[i][2]; //
d_zeta[i];

            J[2][0] += element[i].z * intagrate_points_3D[p_id].derivative[i][0]; //
d_xi[i];
```

```
            J[2][1] += element[i].z * intagrate_points_3D[p_id].derivative[i][1]; //
d_eta[i];
            J[2][2] += element[i].z * intagrate_points_3D[p_id].derivative[i][2]; //
d_zeta[i];
      }

      // вычисляем Якобиан (определитель)
      double det_J = J[0][0] * J[1][1] * J[2][2] - J[0][0] * J[1][2] * J[2][1] +
J[1][0] * J[2][1] * J[0][2]
            - J[1][0] * J[0][1] * J[2][2] + J[2][0] * J[0][1] * J[1][2] - J[2][0] *
J[1][1] * J[0][2];

      // матрица, обратная к транспонированной матрице Якоби
      J_1_T[0][0] = (J[1][1] * J[2][2] - J[2][1] * J[1][2]) / det_J;
      J_1_T[1][0] = (J[2][1] * J[0][2] - J[0][1] * J[2][2]) / det_J;
      J_1_T[2][0] = (J[0][1] * J[1][2] - J[1][1] * J[0][2]) / det_J;
      J_1_T[0][1] = (-J[1][0] * J[2][2] + J[2][0] * J[1][2]) / det_J;
      J_1_T[1][1] = (J[0][0] * J[2][2] - J[2][0] * J[0][2]) / det_J;
      J_1_T[2][1] = (-J[0][0] * J[1][2] + J[1][0] * J[0][2]) / det_J;
      J_1_T[0][2] = (J[1][0] * J[2][1] - J[2][0] * J[1][1]) / det_J;
      J_1_T[1][2] = (-J[0][0] * J[2][1] + J[2][0] * J[0][1]) / det_J;
      J_1_T[2][2] = (J[0][0] * J[1][1] - J[1][0] * J[0][1]) / det_J;

      return det_J;
}
int Integrate_Gauss3Method3D::GetLocG(vector<Node3D> &element, vector<vector<double>>
&locG)
{
      for (int i = 0; i < 8; i++)
      {
            for (int j = 0; j < 8; j++)
            {
                  locG[i][j] = 0;
            }
      }
      for (int p_id = 0; p_id < 27; p_id++)
      {
            vector<vector<double>> J_1(3, vector<double>(3));
            double det_J = CalcJ_1(J_1, p_id, element);

            // intagrate_points_3D[p_id].derivative[i] - градиент i-ой базисной
функции в точке p_id
            vector<vector<double>> modifided_grad(8, vector<double>(3));
            double modifided_coef = intagrate_points_3D[p_id].koef * abs(det_J);
            for (int i = 0; i < 8; i++)
            {
                  // modifided_grad[i] = J_1 * intagrate_points_3D[p_id].derivative[i]
                  for (int j = 0; j < 3; j++)
                  {
                        for (int k = 0; k < 3; k++)
                        {
                              modifided_grad[i][j] += J_1[j][k] *
intagrate_points_3D[p_id].derivative[i][k];
                        }
                  }
            }

            for (int i = 0; i < 8; i++)
            {
                  for (int j = 0; j < 8; j++)
                  {
```

```
                        locG[i][j] += mult(modifided_grad[i], modifided_grad[j]) *
modifided_coef;
                  }
            }

      }

      return 0;
}
int Integrate_Gauss3Method3D::GetLocM_face(vector<Node3D>& face,
vector<vector<double>>& locM, int face_id)
{
      if (locM.size() != 4)
      {
            cout << "error. Integrate_Gauss3Method3D::GetLocM_face
vector<vector<double>>& locM must have size [4][4]" << endl;
      }
      for (int i = 0; i < 4; i++)
      {
            if (locM[i].size() != 4)
            {
                  cout << "error. Integrate_Gauss3Method3D::GetLocM_face
vector<vector<double>>& locM must have size [4][4]" << endl;
            }
            for (int j = 0; j < 4; j++)
            {
                  locM[i][j] = 0;
            }
      }
      for (int p_id = 0; p_id < 3*3; p_id++)
      {
            double det_J = CalcdetJ_face(p_id, face, face_id);
            det_J = sqrt(abs(det_J));

            double modifided_coef = intagrate_points_face[face_id][p_id].koef *
abs(det_J);

            int i_3D, j_3D;
            for (int i = 0; i < 4; i++)
            {
                  i_3D = face_to_node[face_id][i];
                  for (int j = 0; j < 4; j++)
                  {
                        j_3D = face_to_node[face_id][j];
                        locM[i][j] +=
intagrate_points_face[face_id][p_id].basis_func[i_3D] *
intagrate_points_face[face_id][p_id].basis_func[j_3D] * modifided_coef;
                  }
            }
      }

      return 0;
}
```

Модуль решения краевой задачи

```
int FEM::SolveTask(string path)
{
      Init(path); // чтение входных данных
      cout << "read files\n";
      for (int i = 0; i < mesh.kol_elements; i++)
```

```
	{
		GetG_Loc(i);
		slau.AddLocal(A_loc, b_loc, i, mesh);
	}
	cout << "get matrix\n";
	slau.SetS2(mesh, gauss3method_3D);
	slau.SetS1(mesh);
	cout << "set BC\n";

	return 0;
}
int FEM::Init(string path)
{
	basis = { linear1, linear2 };
	derivative_basis = { dlinear1_dksi, dlinear2_dksi };
	gauss3method_3D.Init(basis, derivative_basis);
	mesh.ReadMesh(path);
	slau.GeneratePortret(mesh);
	P.resize(mesh.kol_nodes);
	b_loc.resize(8);
	A_loc.resize(8);
	for (int i = 0; i < 8; i++)
	{
		A_loc[i].resize(8);
	}
	return 0;
}
int FEM::GetG_Loc(int el_id, double B) // получение локальной G
{
	// lambda = K * СУМ(k_m / eta_m)
	double lambda = 0;
	for (int i = 0; i < mesh.kol_phase; i++)
	{
		lambda += mesh.phases[i].k / mesh.phases[i].eta;
	}
	lambda *= mesh.materials[mesh.elements[el_id].material].K;

	vector<Node3D> nodes_coord(8);
	for(int i = 0; i < 8; i++)
	{
		nodes_coord[i] = mesh.nodes[mesh.elements[el_id].node_loc[i]];
	}
	gauss3method_3D.GetLocG(nodes_coord, A_loc);

	for (int i = 0; i < 8; i++)
	{
		for (int j = 0; j < 8; j++)
			A_loc[i][j] *= lambda;
	}

	return 0;
}
int SLAU::SetS2(Mesh& mesh, Integrate_Gauss3Method3D &gauss3d) // учет вторых краевых
{
	vector<Node3D> face(4); // координаты 4ех узлов, образующих грань
	vector<vector<double>> M_loc(4, vector<double>(4));
	vector<double> b_S2(4);

	for (auto iterator_S2 = mesh.S2.begin(); iterator_S2 != mesh.S2.end();
iterator_S2++)
	{
		double lambda = mesh.phases[0].k / mesh.phases[0].eta +
```

```
                    (iterator_S2->value < 0 ? mesh.phases[1].k / mesh.phases[1].eta :
0);
              lambda *= mesh.materials[0].K;

              for (int j = 0; j < 4; j++)
              {
                    b_S2[j] = 0;
              }

              int elem_id = iterator_S2->elem;
              int face_id = iterator_S2->loc_fc;
              double f[4];

              for (int node_i = 0; node_i < 4; node_i++)
              {
                    face[node_i] =
mesh.nodes[mesh.elements[elem_id].node_loc[face_to_node[face_id][node_i]]];
                    f[node_i] = iterator_S2->value;
              }

              gauss3d.GetLocM_face(face, M_loc, face_id);

              for (int i = 0; i < 4; i++)
              {
                    for (int j = 0; j < 4; j++)
                    {
                          b_S2[i] += f[j] * M_loc[i][j];
                    }
              }

              for (int node_i = 0; node_i < 4; node_i++)
              {
                    int node_id_glob =
mesh.elements[elem_id].node_loc[face_to_node[face_id][node_i]];
                    b[node_id_glob] += b_S2[node_i];
              }
        }
        return 0;
}
int SLAU::SetS1(Mesh& mesh) // учет первых краевых
{
        int NS1 = mesh.kol_S1faces;
        double max = 0;
        for (int i = 0; i < di.size(); i++)
        {
              if (di[i] > max)
                    max = di[i];
        }
        for (auto iterator_S1 = mesh.S1.begin(); iterator_S1 != mesh.S1.end();
iterator_S1++)
        {
              di[iterator_S1->first] = max * BIG_VALUE;
              b[iterator_S1->first] = max * BIG_VALUE * iterator_S1->second;
        }
        return 0;
}
```