

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра математического и компьютерного моделирования

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ТЕЧЕНИЙ
МЕЛКОЙ ВОДЫ МЕТОДОМ КОНЕЧНЫХ ЭЛЕМЕНТОВ

Бакалаврская работа

студента 4 курса 413 группы

направление 01.03.02 - Прикладная математика и информатика

механико-математического факультета

Шарова Александра Вадимовича

Научный руководитель
старший преподаватель

В.С. Кожанов

Зав. кафедрой
зав.каф., д.ф. – м. н., доцент

Ю.А. Блинков

Саратов 2018

СОДЕРЖАНИЕ

Стр.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	4
ВВЕДЕНИЕ	5
1 Вывод уравнений мелкой воды.....	7
2 Постановка задачи	15
3 Триангуляция двумерной области	17
3.1 Алгоритм Рапперта [Jim Ruppert].....	18
3.2 Алгоритм Чу [L. Paul Chew]	21
4 Метод конечных элементов в двумерной области.....	22
4.1 Метод взвешенных невязок	23
4.2 Метод конечных элементов	26
4.3 МКЭ для рассматриваемой задачи	28
4.3.1 Интегрирование функции двух переменных по произ- вольному треугольнику	28
4.3.2 Уравнения мелкой воды в терминах МКЭ.....	29
5 Решение задачи.....	31
5.1 Построение графиков	31
5.2 Изменение формы водоёма.....	31
5.2.1 Пруд.....	32
5.2.2 Реальный водоём.....	35
5.3 Учёт наличия острова (островов) внутри водоёма	38
5.3.1 Пруд.....	38
5.3.2 Реальный водоём.....	41
5.4 Программная реализация	42
6 Разработка базы данных для хранения информации о про- ведённых экспериментах	44
6.1 Описание базы данных	44
6.2 Формат представления данных в БД	46
6.3 Примеры запросов к БД	49

ЗАКЛЮЧЕНИЕ	50
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	51
ПРИЛОЖЕНИЕ А Исходный код реализации	53
ПРИЛОЖЕНИЕ Б Исходный код Docker контейнера	70
ПРИЛОЖЕНИЕ В Выкладки для МКЭ	71
B.1 Уравнение неразрывности.....	71
B.2 Первое уравнение мелкой воды.....	72
B.3 Второе уравнение мелкой воды	74

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1. МКЭ – метод конечных элементов
2. КЭ – конечный элемент
3. МВ – мелкая вода
4. ГЖС – газожидкостная смесь
5. POLY – тип файла, в котором содержится список вершин и сегментов. Помимо этого, данный файл может содержать в себе информацию о пустотах и вогнутостях, а также некоторую дополнительную информацию [1]
6. СУБД – система управления базами данных
7. БД – база данных
8. NoSQL – термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL
9. Oracle RDBMS – объектно-реляционная система управления базами данных компании Oracle
10. Нода кластера – один компьютер в группе, объединённой высокоскоростными каналами связи и представляющей с точки зрения пользователя единый аппаратный ресурс
11. Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы

ВВЕДЕНИЕ

Для корректного построения математических моделей и численных методов расчета течения жидкости важно знать, каким именно образом описываются основные процессы, происходящие в водоеме. Для этого строятся модели, которые учитывают основные характеристики течения в природной среде.

На данный момент во многих случаях не оправдывается применение более сложных математических моделей для исследования течений в прибрежных водах и озерах, чем модели, полученные путем применения осредненных по вертикали характеристик и основанные на численном решении двумерных уравнений – уравнений мелкой воды.

Уравнения мелкой воды – широко известное приближение, на основе которого проводится численное моделирование течений в реках и водоемах, в прибрежных зонах морей и океанов. Трехмерные решения данных уравнений являются нецелесообразными, так как они требуют намного большего количества исходной информации и машинного времени даже с учетом современных вычислительных мощностей.

При выводе данных уравнений предполагается, что среда представляет собой достаточно тонкий слой, глубина которого много меньше его продольного размера, поэтому вертикальной составляющей скорости можно пренебречь и полагать, что продольные скорости постоянны по толщине слоя. Исходя из этого данное приближение успешно применяется для описания течений, где влияние на поток свободной поверхности и рельефа дна значительно.

Целью представленной бакалаврской квалификационной работы является построение и численная реализация математической модели на основе двумерных уравнений мелкой воды с помощью метода конечных элементов, который уже давно зарекомендовал [2] себя в таких областях, как механика деформируемого тела, электродинамика и, конечно же, гидродинамика. Данный метод является оптимальным для решения поставленной задачи, так как именно он дает возможность применять достаточно гибкую разбивку рассматриваемой области и при его использовании достаточно удовлетворить лишь главным граничным условиям. Для этой задачи требуется составить

и решить систему дифференциальных уравнений с использованием метода конечных элементов. Предполагается, что глубина водоема постоянна, а сам водоем является однородным, то есть не рассматривается случай ГЖС. Необходимо учесть, что озеро подвержено влиянию ветра, а также требуется рассмотреть данную задачу с различными начальными параметрами и проанализировать полученные результаты. Похожая задача для уже решалась ранее, но рассматривался либо только стационарный случай [3], либо использовались другие методы [4].

Актуальность данной работы обусловлена тем, что сейчас есть потребность в более точном и быстром решении уже решенных физических задач. Этой потребности отвечает появление новых быстродействующих систем и методов решения. Так, в данной бакалаврской работе была разработана параллельная реализация МКЭ для уравнений МВ, которая позволит более эффективно исследовать различные течения, которые могут быть описаны с помощью уравнений МВ.

В данной работе будут представлены основные определения и понятия для уравнений мелкой воды, которые позволяют составить и решить поставленную задачу, состоящую из двух частей: аналитической и численной. Аналитически будут описаны течения жидкости при пренебрежении температурными эффектами, для которых широко используются классические уравнения Сен-Венана. После этого с помощью МКЭ была построена и рассчитана математическая модель МВ. В качестве реализации модели будет написана программа на языке Python [5, 6], которая выдает результаты решения системы уравнений мелкой воды, а также строит графики, наглядным образом отображающие полученные результаты.

1 Вывод уравнений мелкой воды

Запишем два основных уравнения для жидкости – уравнение количества движения и уравнение неразрывности [7]:

$$-\frac{\partial p}{\partial x_k} + \frac{\partial \tau_{ik}}{\partial x_i} + \rho b_k = \frac{\partial}{\partial x_i}(\rho v_i v_k) + \frac{\partial}{\partial t}(\rho v_k); \quad (1.1)$$

$$\frac{D\rho}{Dt} + \rho \frac{\partial v_i}{\partial x_i} = 0. \quad (1.2)$$

Если в 1.1 и 1.2 пренебречь температурными эффектами, получим:

$$-\frac{\partial p}{\partial x_k} + \frac{\partial \tau_{ik}}{\partial x_i} + \rho b_k = \frac{D(\rho v_k)}{Dt} \quad (1.3)$$

$$\frac{\partial(\rho v_i)}{\partial t} + \frac{\partial \rho}{\partial t} = 0, \quad (1.4)$$

где p – давление, оказываемое на единицу площади поверхности воды, b_k – массовые силы, приходящиеся на единицу массы, v_k – скорость частицы воды в направлении оси x_k , τ_{ik} – вязкостные составляющие вектора напряжения, ρ – массовая плотность воды.

При математическом моделировании движения мелкой воды сложно применять данные уравнения вследствие наличия свободной поверхности, изменения границ во время приливов и отливов и большого количества переменных.

Данных трудностей можно избежать после ряда упрощений, в результате чего получаются уравнения мелкой воды. Первое упрощение состоит в том, что уравнение количества движения в проекции на ось x_3 записывается в виде:

$$-\frac{\partial p}{\partial x_3} = \rho g, \quad (1.5)$$

где массовые силы отрицательны, поскольку действуют в направлении, противоположном оси x_3 . При выводе формулы 1.5 пренебрегаем всеми членами,

которые характеризуют ускорение, и соответствующими им напряжениями. Проинтегрируем выражение 1.5 и в результате получим:

$$p = \int_{x_3}^{\eta} \rho g dx_3 = \rho g(\eta - x_3) + p_a, \quad (1.6)$$

где p_a – атмосферное давление на поверхности воды, η – возвышение свободной поверхности в соответствии с рисунком 1.1.

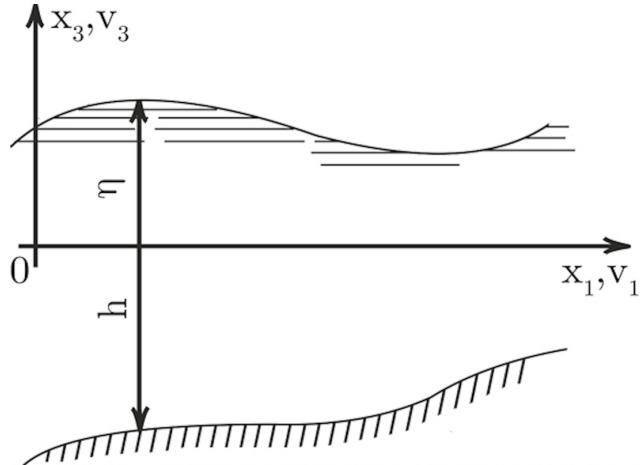


Рисунок 1.1 – Возвышение свободной поверхности.

Оставшиеся два уравнения количества движения по направлениям x_1 и x_2 останутся без изменений:

$$-\frac{\partial p}{\partial x_k} + \frac{\partial \tau_{ik}}{\partial x_i} + \rho b_k = \frac{D(\rho v_k)}{Dt}. \quad (1.7)$$

При $k = 1$ выражение 1.7 дает проекцию на ось x_1 , при $k = 2$ – на ось x_2 . В выражении 1.7 величина v – средняя скорость, ρ – переменная массовая плотность и τ – сумма вязкостных и турбулентных напряжений.

Проинтегрируем выражения 1.4 и 1.7 по x_3 . Для уравнения неразрывности это даст:

$$\int_{-h}^{\eta} \left(\frac{\partial(\rho v_i)}{\partial x_i} + \frac{\partial \rho}{\partial t} \right) dx_3 = 0, \quad (1.8)$$

где h – это глубина, измеряемая от базовой поверхности (в общем случае не горизонтальной).

Определим поток q_k количества жидкости (массу жидкости, приходящуюся на единицу длины и времени):

$$q_i = \int_{-h}^{\eta} \rho v_i dx_3 = \rho \int_{-h}^{\eta} v_i dx_3. \quad (1.9)$$

Предполагается, что $\rho(x_1, x_2)$ не зависит от x_3 .

При интегрировании уравнения 1.8 необходимо использовать кинематическое условие и правило Лейбница [8] для вычисления частной производной интеграла с переменными пределами. Согласно этому правилу:

$$\frac{\partial}{\partial x_1} \int_{h_1(x_1, x_2)}^{h_2(x_1, x_2)} f(x_1, x_2, x_3) dx_3 = \int_{h_1(x_1, x_2)}^{h_2(x_1, x_2)} \frac{\partial f}{\partial x_1} dx_3 + f \Big|_{h_2} \frac{\partial h_2}{\partial x_1} + f \Big|_{h_1} \frac{\partial h_2}{\partial x_1}. \quad (1.10)$$

Аналогично для производной по x_2 .

Кинематическое соотношение для свободной поверхности можно записать следующим образом:

$$v_3 \Big|_{x_2=\eta} = \frac{D\eta}{Dt} = \frac{\partial \eta}{\partial t} + v_1 \Big|_{\eta} \frac{\partial \eta}{\partial x_1} + v_2 \Big|_{\eta} \frac{\partial \eta}{\partial x_2}. \quad (1.11)$$

Применим формулы 1.9-1.11 к уравнению 1.8 и получим:

$$\frac{\partial q_i}{\partial x_i} + \frac{\partial(\rho H)}{\partial t} = 0, \quad (1.12)$$

где $H = \eta + h$.

Чтобы проинтегрировать уравнение количества движения 1.7 по x_3 , определим мгновенные скорости v_1, v_2 :

$$\begin{aligned} v_1 &= \bar{v}_1(x_1, x_2, t) + v'_1(x_1, x_2, x_3, t); \\ v_2 &= \bar{v}_2(x_1, x_2, t) + v'_2(x_1, x_2, x_3, t), \end{aligned} \quad (1.13)$$

где величина \bar{v} означает средние по вертикали скорости, а v' – отклонение от этих средних значений при различных значениях x_3 .

Следовательно:

$$\langle v_k \rangle = \int_{-h}^{\eta} v_k dx_3 = \frac{1}{\rho} q_k, \quad v_k = \frac{1}{H} \langle v_k \rangle, \quad (1.14)$$

так как $\langle v'_k \rangle = 0$, где знак $\langle \rangle$ означает среднее значение стоящее внутри величины.

Будем предполагать, что массовые силы обусловлены только эффектом Кориолиса. Таким образом получаем:

$$b_1 = \rho f v_2, \quad b_2 = -\rho f v_1. \quad (1.15)$$

Предположим, что наклоны поверхности и дна малы по сравнению с единицей, тогда составляющие внутреннего напряжения можно аппроксимировать следующим образом в соответствии с рисунком 1.2:

$$\begin{aligned} \tau_1|_s &\approx \left\{ -\tau_{11} \frac{\partial \theta}{\partial x_1} - \tau_{12} \frac{\partial \theta}{\partial x_2} + \tau_{13} \right\}, \\ \tau_1|_b &\approx \left\{ \tau_{11} \frac{\partial h}{\partial x_1} + \tau_{12} \frac{\partial h}{\partial x_2} - \tau_{13} \right\}. \end{aligned} \quad (1.16)$$

Аналогичные значения можно выписать для величин $\tau_2|_s$ и $\tau_2|_b$.

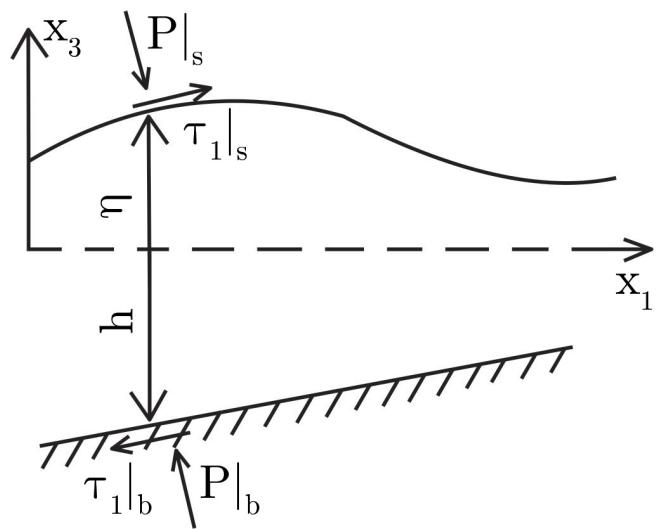


Рисунок 1.2 — Силы, действующие на свободную поверхность.

Величины $\tau_2|_s$ и $\tau_2|_b$ можно интерпретировать как компоненты внешней силы, приложенные к поверхности и дну.

Подставим теперь соотношения 1.13 - 1.15 в уравнения количества движения, проинтегрированные по x_3 , и дополнительно используем правило Лейбница [8] и кинематическое условие 1.11. Тогда:

$$\begin{aligned} \frac{\partial q_1}{\partial t} + \frac{\partial}{\partial x_1} \left(\frac{q_1^2}{H} \right) + \frac{\partial}{\partial x_2} \left(\frac{q_1 q_2}{H} \right) &= -\frac{\partial N_p}{\partial x_2} + \frac{\partial N_{11}}{\partial x_1} + \frac{\partial N_{12}}{\partial x_2} + \\ &\quad + f q_2 + p \left| \frac{\partial \eta}{\partial x_1} + \tau_1 \right|_s + p \left| \frac{\partial h}{\partial x_1} - \tau_1 \right|_b; \\ \frac{\partial q_2}{\partial t} + \frac{\partial}{\partial x_1} \left(\frac{q_1 q_2}{H} \right) + \frac{\partial}{\partial x_2} \left(\frac{q_2^2}{H} \right) &= -\frac{\partial N_p}{\partial x_2} + \frac{\partial N_{22}}{\partial x_2} + \frac{\partial N_{21}}{\partial x_1} + \\ &\quad + f q_1 + p \left| \frac{\partial \eta}{\partial x_2} + \tau_2 \right|_s + p \left| \frac{\partial h}{\partial x_2} - \tau_2 \right|_b, \end{aligned} \quad (1.17)$$

где

$$\begin{aligned} N_p = < p > &= \int_{-h}^{\eta} pdx_3 = \rho g \frac{H^2}{2} + H p_a; \\ N_{11} = < \tau_{11} > - < p v'_1 v'_1 >; \\ N_{22} = < \tau_{22} > - < p v'_2 v'_2 >; \\ N_{12} = < \tau_{12} > - < p v'_1 v'_2 >; \end{aligned} \quad (1.18)$$

Более того, элементы N_{ik} могут быть аппроксимированы следующими выражениями:

$$\begin{aligned} N_{11} &\approx 2\varepsilon_{11} \frac{\partial q_1}{\partial x_1}; \\ N_{22} &\approx 2\varepsilon_{22} \frac{\partial q_2}{\partial x_2}; \\ N_{12} &\approx \varepsilon_{12} \left(\frac{\partial q_2}{\partial x_1} + \frac{\partial q_1}{\partial x_2} \right), \end{aligned} \quad (1.19)$$

где ε_{ik} обобщенные коэффициенты вихревой вязкости. Для изотропного характера течения $\varepsilon_{11} = \varepsilon_{22} = \varepsilon_{12} = \varepsilon$.

Касательные напряжения на дне обычно определяются соотношениями:

$$\begin{aligned}\tau_1 \Big|_b &= \frac{g}{c^2} \frac{1}{\rho} \frac{q_1 \sqrt{(q_1^2 + q_2^2)}}{H^2}; \\ \tau_2 \Big|_b &= \frac{g}{c^2} \frac{1}{\rho} \frac{q_2 \sqrt{(q_1^2 + q_2^2)}}{H^2},\end{aligned}\quad (1.20)$$

где g – ускорение силы тяжести, c – коэффициент трения, ρ – плотность воды.

Составляющие напряжения трения на поверхности воды обычно обусловлены действием ветра и могут быть найдены по формулам:

$$\begin{aligned}\tau_1 \Big|_s &= \gamma^2 \rho_a W^2 \cos(\theta); \\ \tau_2 \Big|_s &= \gamma^2 \rho_a W^2 \sin(\theta),\end{aligned}\quad (1.21)$$

где γ^2 – коэффициент ветрового напряжения, ρ_a – плотность воздуха, W – скорость ветра, θ – угол между осью x_1 и направлением ветра.

Уравнения 1.17 перепишем в виде:

$$\begin{aligned}\frac{\partial q_1}{\partial t} + \frac{\partial}{\partial x_1} \left(\frac{q_1^2}{H} \right) + \frac{\partial}{\partial x_2} \left(\frac{q_1 q_2}{H} \right) &= \frac{\partial}{\partial x_1} (N_{11} - N_p) + \frac{\partial N_{12}}{\partial x_2} + B_1; \\ \frac{\partial q_2}{\partial t} + \frac{\partial}{\partial x_1} \left(\frac{q_1 q_2}{H} \right) + \frac{\partial}{\partial x_2} \left(\frac{q_2^2}{H} \right) &= \frac{\partial}{\partial x_2} (N_{22} - N_p) + \frac{\partial N_{12}}{\partial x_1} + B_2,\end{aligned}\quad (1.22)$$

где

$$\begin{aligned}B_1 &= f q_2 + \gamma^2 \rho_a W^2 \cos(\theta) - \left(\frac{g}{c^2} \right) \frac{1}{\rho} \frac{q_1 \sqrt{(q_1^2 + q_2^2)}}{H^2} + p_a \frac{\partial H}{\partial x_1} + \rho g H \frac{\partial h}{\partial x_1}; \\ B_2 &= -f q_1 + \gamma^2 \rho_a W^2 \sin(\theta) - \left(\frac{g}{c^2} \right) \frac{1}{\rho} \frac{q_2 \sqrt{(q_1^2 + q_2^2)}}{H^2} + p_a \frac{\partial H}{\partial x_2} + \rho g H \frac{\partial h}{\partial x_2}.\end{aligned}$$

Для решения окончательной системы уравнений 1.22, дополненных условием 1.12, необходимо установить требуемые граничные условия. Будем считать, что граница S состоит из двух частей: твердой границы S_1 и жидкой

S_2 , представляющей границу рассматриваемого водоема с открытым морем в соответствии с рисунком 1.3.

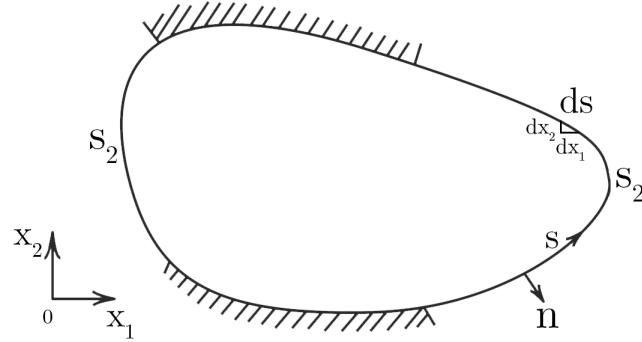


Рисунок 1.3 — Водоём с изображенными границами.

В системе координат $s - n$, связанной с границей течения, расход массы жидкости можно записать через q_s и q_n :

$$\begin{aligned} q_n &= \int_{-h}^{\eta} \rho v_n dx_3 = \alpha_{n1} q_1 + \alpha_{n2} q_2; \\ q_s &= \int_{-h}^{\eta} \rho v_s dx_3 = -\alpha_{n2} q_1 + \alpha_{n1} q_2, \end{aligned} \quad (1.23)$$

где $\alpha_{n1} = \cos(\bar{n}, x_1); \alpha_{n2} = \cos(\bar{n}, x_2)$.

Для установления результирующих сил можно воспользоваться формулами:

$$\begin{aligned} N_{n1} &= \alpha_{n1}(N_{11} - N_p) + \alpha_{n2}N_{12}; \\ N_{n2} &= \alpha_{n1}N_{12} + \alpha_{n2}(N_{22} - N_p). \end{aligned} \quad (1.24)$$

По значениям N_{n1} и N_{n2} определяется нормальная и касательная составляющие результирующей силы для наклонной площадки:

$$\begin{aligned} N_{nn} &= \alpha_{n1}N_{n1} + \alpha_{n2}N_{n2}; \\ N_{ns} &= \alpha_{n2}N_{n1} + \alpha_{n1}N_{n2}. \end{aligned} \quad (1.25)$$

Если же в исследуемую акваторию впадает река, то на S_1 :

$$\begin{aligned} q_n &= \overline{q_n} = |q|; \\ q_s &= 0, \end{aligned} \quad (1.26)$$

где $|q|$ - поток втекающей реки.

На жидкой границе S_2 необходимо задать нормальные и касательные силы:

$$\begin{aligned} N_{nn} &= \overline{N_{nn}}; \\ N_{ns} &= \overline{N_{ns}}. \end{aligned} \quad (1.27)$$

Но так как слагаемыми, учитывающими вихревую вязкость в уравнении 1.22 можно пренебречь, то касательные силы или скорости не могут быть заданы. Таким образом, граничные условия сводятся к следующим:

$$\begin{aligned} q_n &= 0 \text{ или } q_n = \overline{q_n} \text{ на } S_1 \\ N_{nn} &= \overline{N_{nn}} = -N_p \text{ на } S_2. \end{aligned} \quad (1.28)$$

2 Постановка задачи

Рассматривается течение мелкой воды в закрытом водоеме при учете влияния на этот водоем ветра. Это явление описывается системой дифференциальных уравнений в частных производных [9], которая состоит из двух уравнений мелкой воды и уравнения неразрывности:

$$\begin{cases} \frac{\partial q_i}{\partial x_i} + \frac{\partial(\rho H)}{\partial t} = 0 \\ \frac{\partial q_1}{\partial t} + \frac{\partial}{\partial x_1} \left(\frac{q_1^2}{H} \right) + \frac{\partial}{\partial x_2} \left(\frac{q_1 q_2}{H} \right) = \frac{\partial}{\partial x_1} (N_{11} - N_p) + \frac{\partial N_{12}}{\partial x_2} + B_1 \\ \frac{\partial q_2}{\partial t} + \frac{\partial}{\partial x_1} \left(\frac{q_1 q_2}{H} \right) + \frac{\partial}{\partial x_2} \left(\frac{q_2^2}{H} \right) = \frac{\partial}{\partial x_2} (N_{22} - N_p) + \frac{\partial N_{12}}{\partial x_1} + B_2, \end{cases} \quad (2.1)$$

где

$$\begin{aligned} B_1 &= f q_2 + \gamma^2 \rho_a W^2 \cos(\theta) - \left(\frac{g}{c^2} \right) \frac{1}{\rho} \frac{q_1 \sqrt{(q_1^2 + q_2^2)}}{H^2} + p_a \frac{\partial H}{\partial x_1} + \rho g H \frac{\partial h}{\partial x_1}; \\ B_2 &= -f q_1 + \gamma^2 \rho_a W^2 \sin(\theta) - \left(\frac{g}{c^2} \right) \frac{1}{\rho} \frac{q_2 \sqrt{(q_1^2 + q_2^2)}}{H^2} + p_a \frac{\partial H}{\partial x_2} + \rho g H \frac{\partial h}{\partial x_2}. \end{aligned} \quad (2.2)$$

В системе задействованы следующие переменные физические величины:

ρ – плотность воды	1000 [кг/м ³]
c – коэффициент трения Шэзи	10 [м ^{1/2} с ⁻¹]
g – ускорение свободного падения	9,832 [м/с ²]
γ^2 – коэффициент ветрового напряжения	0.002
p_a – атмосферное давление на поверхности воды	10^5 [Па]
ρ_a – плотность воздуха	1,2754 [кг/м ³]
f – сила Кориолиса	$0.973 \cdot 10^{-4}$ [1/c]

Также в системе используются следующие переменные:

$$W - \text{скорость ветра} \quad [\text{м/с}]$$

$$\theta - \text{угол между } x_1 \text{ и направлением ветра}$$

$$h - \text{возвышение свободной поверхности} \quad [\text{м}]$$

Для данного двумерного случая системы дифференциальных уравнений требуется найти решение с помощью метода конечных элементов. Реализация должна работать с любой произвольно заданной областью. В качестве примеров таких областей должны быть как искусственно вырытые пруды, так и реальные озера.

Так как задача рассматривается двумерная, то КЭ будет представлять из себя треугольник. Простые двумерные области разбиваются на квадраты, которые впоследствии разбиваются на треугольники. В случае более сложной геометрии триангуляция рассматриваемой области осуществляется с помощью более сложных алгоритмов, которые необходимо рассмотреть перед решением поставленной задачи, для того чтобы корректно генерировать сетку, на которой будет найдено решение поставленной задачи. В следующем разделе рассмотрены два основных алгоритма триангуляции, дающие оптимальные разбиения исходной области.

3 Триангуляция двумерной области

В геометрии триангуляция дискретного множества точек $P \subset \mathbb{R}^{n+1}$ в наиболее общем значении — это разбиение выпуклой оболочки некоторого набора точек, которое представляет собой планарный граф [10]. Одна из фигур разбиения является выпуклой оболочкой разбиваемого множества, а остальные симплексами — геометрическими фигурами, которые являются n -мерным обобщением треугольника. В любой триангуляции (T) формально должны выполняться следующие свойства:

1. любые два симплекса в T пересекаются в общей грани ребра или вершины или вообще не пересекаются;
2. множество точек, являющихся вершинами симплексов разбиения, совпадает с множеством P ;
3. нельзя добавить ни одного нового ребра в граф без нарушения планарности.

Одно и то же множество можно триангулировать разными способами. Триангуляция дает тем лучшую аппроксимацию, чем больше её минимальный угол, при этом формируемые симплексы стремятся к равноугольности. Очень важна максимизация минимального угла в вычислительных задачах, когда точность производимых вычислений очень сильно зависит от размера минимального угла триангуляции. Наилучшей в этом смысле триангуляцией является триангуляция Делоне. Простейшим способом её построения является инкрементальный алгоритм, работающий за $o(n^2)$ операций, но он не поддерживает вырожденные случаи, когда 4 точки из множества лежат на одной окружности: в этом случае триангуляция Делоне не уникальна, и способов разбиения существует несколько, но минимальные углы этих триангуляций равны. Иногда даже минимальный угол триангуляции Делоне оказывается слишком малым для устойчивой работы использующего её алгоритма, и тогда можно произвести улучшение, используя алгоритм Рапперта [J. Ruppert]. При этом будут добавлены новые вершины триангуляции, а также образованы дополнительные треугольники. Стабильность численного алгоритма (метода конечных элементов, к примеру) может возрасти многократно за счет появления нижней границы для углов.

3.1 Алгоритм Рапперта [Jim Ruppert]

Алгоритм Рапперта [J. Ruppert], или же усовершенствованный алгоритм Делоне, довольно новый, он был опубликован в 1994 году. Данный алгоритм адаптирован для МКЭ, а это значит, что он удовлетворяет всем требованиям, которые предъявляются к конечно-элементным сеткам, а именно:

1. треугольники не должны быть сильно вытянутыми, так как наличие таких треугольников отрицательно сказывается на точности результатов расчета;
2. должна быть учтена геометрия рассматриваемой области, и в местах со сложной геометрией сетка должна сгущаться.

Если говорить точнее, алгоритм Рапперта [J. Ruppert] не является алгоритмом генерации сеток. Он является лишь алгоритмом улучшения качества сетки, от чего и произошло название. В общем случае входными данными для него будут являться геометрия конструкции в виде замкнутых полигонов и первичное разбиение конструкции на треугольники.

При описании алгоритма Рапперта [J. Ruppert] следует ввести следующую терминологию:

- Элемент по смыслу совпадает с конечным элементом. Это элементарная часть пространства, на множество которых мы хотим разбить более сложную область этого пространства;
- Узел — точка в которой сходятся грани и соприкасаются несколько элементов;
- Сегмент — это отрезок, соединяющий две соседние точки лежащие на границе области разбиения; Другими словами, этот отрезок принадлежит контурам области;
- Грань — это отрезок, по которому граничат два соприкасающихся треугольника;
- Включенная точка — любая вершина текущей сетки, находящаяся внутри окружности, радиусом которой является любой сегмент;
- «Неправильный треугольник» — треугольник, не удовлетворяющий глобальным условиям, которые накладываются на генерируемую сетку;

- Вытянутый треугольник — треугольник, имеющий одну сторону, намного отличающуюся от других двух. То есть треугольник слишком вытянут вдоль некоторой прямой.

Алгоритм опирается на две базовые процедуры:

1. Разбиение неправильного треугольника с введением нового узла.
 - (a) Вычисляются координаты центра окружности, описанной вокруг треугольника, подлежащего разбиению
 - (b) В найденную точку добавляется новый узел
 - (c) Удаляется треугольник, подлежащий разбиению, и прилегающие к нему треугольники, а затем добавляются новые в соответствии с рисунком 3.1

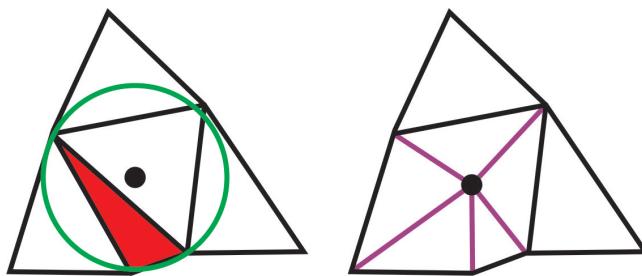


Рисунок 3.1 — Разбиение «неправильного» треугольника.

2. Разбиение сегмента, принадлежащего границе области разбиения с введением нового узла.
 - (a) Если в окружность, диаметром которой является сегмент, принадлежащий границе области разбиения, попадает точка, не принадлежащая этому сегменту, то сегмент делится на две части;
 - (b) Удаляется треугольник, которому принадлежал первоначальный сегмент, и добавляются два новых треугольника в соответствии с рисунком 3.2.

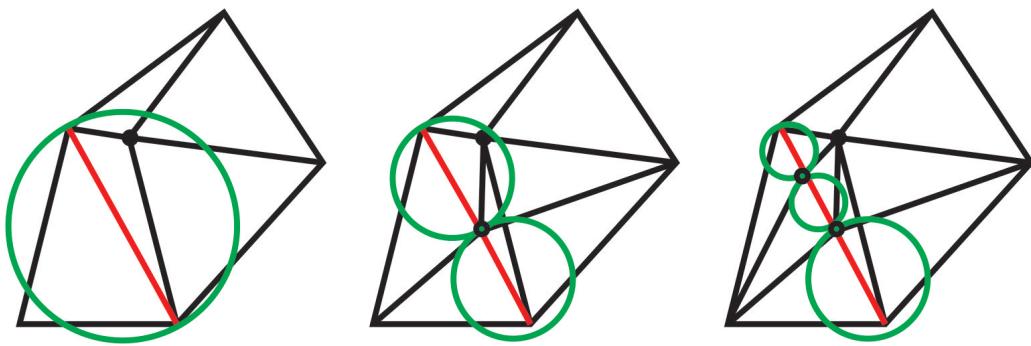


Рисунок 3.2 — Разбиение «неправильного» сегмента посередине и добавление двух новых треугольников.

Принцип работы алгоритма состоит в цикле определения «неправильных» треугольников. В каждом треугольнике сетки определяется минимальный угол и, затем сравниваются минимальные углы для всех треугольников с целью нахождения минимального угла сетки. Другими словами, минимальный угол в сетке равен минимально возможному углу, который образован двумя сегментами, либо двумя гранями, либо сегментом и гранью, имеющими общий узел.

Критерий минимального угла автоматически обеспечивает сгущение сетки вблизи мелких подробностей: отверстий, углов и вырезов.

Все описанные процедуры выполняются в определенной последовательности, они имеют разный приоритет. Общая схема работы алгоритма следующая:

1. Производится поиск включенной точки перебором всех точек и сегментов (только сегментов, но не граней).
2. Если такая точка найдена, то над сегментом, который включает ее, выполняется процедура деления сегмента пополам и производится возврат на шаг 1. Если включенных точек не было найдено, то алгоритм переходит на следующий шаг.
3. Производится поиск треугольника с минимальным углом.
4. Если минимальный угол меньше заданного параметра, то над треугольником, и его содержащим производится процедура деления и производится возврат на шаг 1. В противном случае происходит переход на следующий шаг.
5. Производится поиск треугольника максимальной площадью.

6. Если максимальная площадь больше заданного параметра, то над таким треугольником производится процедура деления и производится возврат на шаг 1. В противном случае происходит переход на следующий шаг.

7. Все условия удовлетворены. Конец работы алгоритма

Автор в своей работе [11] приводит строгие математические доказательства, гарантирующие правильную работу алгоритма при минимальном угле $\alpha < 20^\circ$. Данный алгоритм имеет ряд теоретических и практических преимуществ для генерации сетки. Для не острого ввода и минимального угла порога около 20.70° алгоритм гарантированно завершает работу и создает сетку оптимального размера с постоянным коэффициентом, а также дает теоретически подтвержденные гарантии качества получаемой с его помощью сетки.

3.2 Алгоритм Чу [L. Paul Chew]

В некоторых случаях в алгоритме Рапперта [J. Ruppert] получаются слишком большие сегменты, а для МКЭ очень хорошо иметь как можно более мелкие элементы в сетке. Для того чтобы этого избежать, был разработан алгоритм Чу [L. Chew], в котором сетка получается мельче за счет того, что он более консервативен в отношении разбиения на сегменты, когда граница угла меньше 30° . Главное преимущество данного алгоритма по сравнению с алгоритмом Рапперта [J. Ruppert] состоит в том, что КЭ в сетке получаются с углом до 28.6° , что является несомненным плюсом, так как элементы в сетке получаются приблизительно одинаковые. Алгоритм построен на более сложной теоретической базе [12], и главное его отличие от алгоритма Рапперта в том, что Рапперт только гарантирует хорошее разбиение и угол меньший, чем 20.7° , а Чу обязательно даст хорошее разбиение, правда, с углом до 26.5° . На практике данные алгоритмы дают примерно одинаковые разбиения, но алгоритм Чу является более популярным среди разработчиков математических пакетов.

4 Метод конечных элементов в двумерной области

Метод конечных элементов [13, 14] – численный метод решения дифференциальных уравнений в частных производных, возникающими при решении задач прикладной физики. В его основе лежат две главные идеи: дискретизация исследуемого объекта и кусочно-элементная аппроксимация исследуемых функций (в физической интерпретации - температуры, давления, перемещения и т.д.).

Идея состоит в том, что область, в которой ищется решение дифференциальных уравнений, разбивается на конечное количество элементов. В каждом из элементов произвольно выбирается вид аппроксимирующей функции. Вне своего элемента аппроксимирующая функция равна нулю. Значения функций в узлах являются решением задачи и заранее неизвестны. Коэффициенты аппроксимирующих функций обычно ищутся из условия равенства значения соседних функций на границах между элементами (в узлах). Затем эти коэффициенты выражаются через значения функций в узлах элементов. Составляется система уравнений. Количество уравнений равно количеству неизвестных значений в узлах, на которых ищется решение исходной системы, и прямо пропорционально количеству элементов. Так как каждый из элементов связан с ограниченным количеством соседних, система уравнений имеет разрежённый вид, что упрощает её решение.

Если говорить в матричных терминах, то собираются так называемые матрицы жёсткости и масс. Далее на эти матрицы накладываются граничные условия (например, при условиях Неймана в матрицах не меняется ничего, а при условиях Дирихле из матриц вычёркиваются строки и столбцы, соответствующие граничным узлам, так как в силу краевых условий значение соответствующих компонент решения известно). После собирается система уравнений, тип которых может различаться в зависимости от поставленной задачи. Они могут быть как алгебраическими, так и дифференциальными. После того как система получена, она решается любым из доступных методов.

Основное отличие МКЭ от классических алгоритмов вариационных принципов и от методов невязок заключается в выборе базисных функций. Они берутся в виде кусочно-непрерывных функций, которые обращаются в ноль

всюду, кроме ограниченных подобластей, являющихся конечными элементами. Это в свою очередь ведет к разреженной структуре матрицы коэффициентов разрешающей системы уравнений.

Главные достоинства МКЭ состоят в следующем:

1. исследуемые объекты могут иметь любую форму и различную физическую природу, это твёрдые тела, жидкости, газы, электромагнитные среды;
2. конечные элементы могут иметь различную криволинейную форму и различные размеры;
3. можно исследовать однородные и неоднородные, изотропные и анизотропные тела с линейными и нелинейными свойствами;
4. можно решать как стационарные, так и нестационарные задачи;
5. можно моделировать любые граничные условия;
6. вычислительный алгоритм, представленный в матричной форме, формально единообразен для различных физических задач и для задач различной размерности. Это удобно для вычисления на компьютере, так как методология не меняется и фактически используется единая программа численного решения;
7. на одной и той же сетке конечных элементов можно решать различные физические задачи, что облегчает анализ связанных между собой задач;
8. разрешающая система уравнений имеет разреженную симметричную ленточную матрицу жёсткости, что ускоряет вычислительный процесс.

Перед тем как рассматривать метод конечных элементов, рассмотрим метод взвешенных невязок, который является частным случаем метода конечных элементов с одним элементом.

4.1 Метод взвешенных невязок

Рассмотрим дифференциальное уравнение вида

$$\mathcal{A} = \mathcal{L}\phi + p = 0 \quad (4.1)$$

в некоторой области Ω .

Здесь \mathcal{L} – некоторый линейный дифференциальный оператор, а r не зависит от неизвестной функции ϕ .

Решение уравнения 4.1 должно удовлетворять условию

$$\mathcal{B} = \mathcal{M}\phi + r = 0 \quad (4.2)$$

на замкнутой кривой Γ , ограничивающей область Ω .

Здесь \mathcal{M} – соответствующий линейный дифференциальный оператор, а r не зависит от неизвестной функции ϕ .

Построим аппроксимацию для решения ϕ , которая на граничной кривой Γ принимает те же значения, что и ϕ . Если найти некоторую функцию ψ , принимающую одинаковые с ϕ значения на Γ , т.е. $\psi|_{\Gamma} = \phi|_{\Gamma}$, и ввести систему линейно независимых базисных функций $\{N_m; m = 1, 2, \dots, N\}$, то на Ω можно предложить следующую аппроксимацию $\hat{\phi}$ для ϕ :

$$\phi \approx \hat{\phi} = \psi + \sum_{m=1}^M a_m N_m, \quad (4.3)$$

где $a_m, m = \overline{1, M}$ – некоторые параметры, вычисляемые так, чтобы получить хорошее приближение, а функция ψ и базисные функции N_m выбраны таким образом, что

$$\mathcal{M}\psi = -r, \quad \mathcal{M}N_m = 0, \quad m = \overline{1, M} \text{ на } \Gamma, \quad (4.4)$$

и поэтому ϕ автоматически удовлетворяет краевым условиям 4.2 при произвольных коэффициентах a_m . Отметим, что система базисных функций (которые также называют функциями формы) должна быть выбрана так, чтобы гарантировать улучшение аппроксимации при возрастании числа базисных функций для M .

Для того чтобы найти аппроксимации производных от ϕ , продифференцируем 4.3 и получим:

$$\begin{aligned}\phi &\approx \hat{\phi} = \psi + \sum_{m=1}^M a_m N_m, \\ \frac{\partial \phi}{\partial x} &\approx \frac{\partial \hat{\phi}}{\partial x} = \frac{\partial \psi}{\partial x} + \sum_{m=1}^M a_m \frac{\partial N_m}{\partial x}, \\ \frac{\partial^2 \phi}{\partial x^2} &\approx \frac{\partial^2 \hat{\phi}}{\partial x^2} = \frac{\partial^2 \psi}{\partial x^2} + \sum_{m=1}^M a_m \frac{\partial^2 N_m}{\partial x^2}\end{aligned}$$

и т.д.

Так как построенное разложение 4.3 удовлетворяет краевым условиям 4.2, то для получения аппроксимации искомой функции ϕ осталось гарантировать, что $\hat{\phi}$ – приближённое решение уравнения 4.1. Для этого вначале введём невязку R_Ω в аппроксимации, определяемую по формуле:

$$R_\Omega = A(\hat{\phi}) = \mathcal{L}\hat{\phi} + p = \mathcal{L}\psi + \left(\sum_{m=1}^M a_m \mathcal{L}N_m \right) + p. \quad (4.5)$$

Чтобы уменьшить эту невязку, потребуем равенства нулю соответствующего числа интегралов от погрешности [8], взятых с различными весами, т.е.

$$\int_{\Omega} W_l R_\Omega d\Omega = \int_{\Omega} W_l \left\{ \mathcal{L}\psi + \left(\sum_{m=1}^M a_m \mathcal{L}N_m \right) + p \right\} d\Omega = 0; \quad l = \overline{1, M}, \quad (4.6)$$

где $\{W_l : l = 1, 2, \dots, M\}$ – это множество линейно независимых весовых (или пробных) функций.

Таким образом, система уравнений метода взвешенных невязок 4.6 сводится к системе линейных алгебраических уравнений для неизвестных коэффициентов a_m , которую можно записать следующим образом:

$$Ka = f, \quad (4.7)$$

где

$$a = (a_1, a_2, \dots, a_M)^T, \quad (4.8)$$

$$K_{lm} = \int_{\Omega} W_l \mathcal{L} N_m d\Omega, \quad l, m = \overline{1, M}; \quad (4.9)$$

$$f_l = - \int_{\Omega} W_l p d\Omega - \int_{\Omega} W_l \mathcal{L} \psi d\Omega \quad l, m = \overline{1, M}. \quad (4.10)$$

Вычислив элементы матрицы K и столбца свободных членов f и решив затем полученную систему, мы найдем a_m , где $m = \overline{1, M}$, и тем самым закончим процесс построения приближенного решения 4.1.

4.2 Метод конечных элементов

В методе взвешенных невязок предполагалось, что базисные функции N_m определены одним выражением на всей области Ω . При этом интегралы в аппроксимирующем уравнении 4.5 вычисляются по всей области.

С другой стороны, область Ω можно разбить на несколько непересекающихся подобластей (конечных элементов Ω^e), которые могут иметь различную форму и размеры. В результате разбивки создается сетка из границ элементов:

$$\Omega = \bigcup_{e=1}^E \Omega^e.$$

Пересечение границ областей Ω^e образуют узлы. Выбор типа и размера КЭ напрямую зависит от рассматриваемой задачи.

После того как мы разбили рассматриваемую область Ω сеткой конечных элементов, можно аппроксимировать решение уравнения отдельно для каждой подобласти Ω^e , где базисные функции могут быть определены различным образом для каждой из подобластей. В этом случае определённые интегралы, входящие в аппроксимирующие уравнения, можно посчитать просуммировав, вклады по каждому элементу:

$$\begin{aligned}\int_{\Omega} W_l R_{\Omega} d\Omega &= \sum_{e=1}^E \int_{\Omega^e} W_l R_{\Omega} d\Omega^e, \\ \int_{\Gamma} \overline{W}_l R_{\Gamma} d\Gamma &= \sum_{e=1}^E \int_{\Gamma^e} W_l R_{\Gamma} d\Gamma^e.\end{aligned}\quad (4.11)$$

В данных интегралах:

$$\sum_{e=1}^E \Omega^e = \Omega, \quad (4.12)$$

$$\sum_{e=1}^E \Gamma^e = \Gamma, \quad (4.13)$$

где E - число подобластей, на которые разбивается область Ω ; Γ^e – часть границы Ω^e , лежащая на Γ .

Система уравнений метода конечных элементов сводится к аналогичной системе метода взвешенных невязок, но за исключением того, что:

$$K_{lm} = \sum_{e=1}^E K_{lm}^e = \sum_{e=1}^E \int_{\Omega^e} W_l^e \mathcal{L} N_m^e d\Omega^e, \quad l, m = \overline{1, M}; \quad (4.14)$$

$$f_l = \sum_{e=1}^E f_l^e = \sum_{e=1}^E \left(- \int_{\Omega^e} W_l^e p d\Omega^e - \int_{\Omega^e} W_l^e \mathcal{L} \psi d\Omega^e \right) \quad l, m = \overline{1, M}. \quad (4.15)$$

Отметим также, что для матрицы K^e не нужно вычислять каждую компоненту, так как её компонента K_{lm}^e равна нулю, если узлы l и m не принадлежат элементу e .

Идея метода конечных элементов заключается в следующем: если подобласти имеют простую форму и базисные функции на этих подобластях вычисляются однотипно, то решать задачу указанным выше способом становиться очень просто.

4.3 МКЭ для рассматриваемой задачи

4.3.1 Интегрирование функции двух переменных по произвольному треугольнику

Интегрирование по конечному элементу является очень важной составляющей МКЭ. Так как в общем случае симплексы в разбиении являются различными, то необходимо уметь интегрировать функцию двух переменных по произвольному треугольнику 4.1. В декартовой системе координат это является достаточно сложной задачей, и именно поэтому необходимо осуществить переход к барицентрической системе координат, в которой эта задача является достаточно простой.

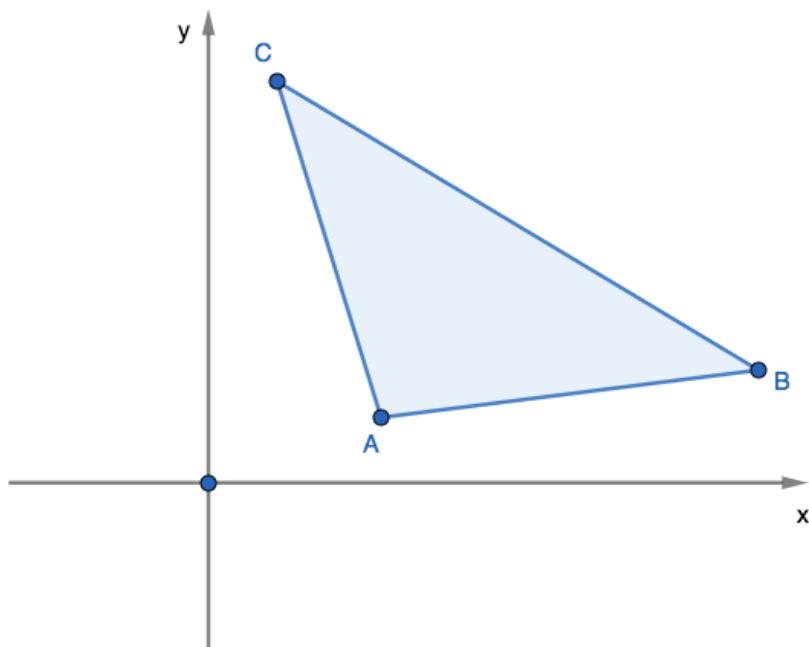


Рисунок 4.1 — Произвольный треугольник в двумерной области.

Барицентрическая система координат представляет собой систему координат, в которой расположение точки симплекса определяется как центр массы или барицентр, как правило, неравных масс, размещенных в его вершинах.

Мы можем записать декартовы координаты точки \mathbf{r} через декартовы компоненты треугольных вершин $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ где $\mathbf{r}_i = (x_i, y_i)$, и в терминах барицентрической системы координат переход будет выглядеть следующим образом:

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \quad (4.16)$$

$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 \quad (4.17)$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \quad (4.18)$$

Вычисление двойного интеграла в барицентрических координатах осуществляется посредством следующей формулы [15]:

$$\int_{\Delta} f(\mathbf{r}) d\mathbf{r} = 2S_{\Delta} \int_0^1 \int_0^{1-\lambda_2} f(\lambda_1 \mathbf{r}_1 + \lambda_2 \mathbf{r}_2 + (1 - \lambda_1 - \lambda_2) \mathbf{r}_3) d\lambda_1 d\lambda_2 \quad (4.19)$$

4.3.2 Уравнения мелкой воды в терминах МКЭ

В рассматриваемой системе 2.1-2.2 можно пренебречь членами $-fq_1$ и fq_2 , так как коэффициент Кориолиса очень мал, а также конвективными членами. После отбрасывания неучитываемых членов в 2.1-2.2 можно перейти к конечно-элементной формулировке. Для этого необходимо представить функции $q_1(x_1, x_2, t), q_2(x_1, x_2, t), H(x_1, x_2, t)$ как разложения:

$$q_1(x_1, x_2, t) = \sum_{m=1}^M a_m(t) N_m(x, y) \quad (4.20)$$

$$q_2(x_1, x_2, t) = \sum_{m=1}^M a_{m+k}(t) N_m(x, y) \quad (4.21)$$

$$H(x_1, x_2, t) = \sum_{m=1}^M a_{2m+k}(t) N_m(x, y) \quad (4.22)$$

Важно заметить, что в данных разложениях применяются одни и те же весовые функции N_k .

Следующим шагом необходимо подставить полученные разложения в исходную систему уравнений. После этого каждое из уравнений необходимо умножить на весовую функцию $W_l(x_1, x_2)$ и проинтегрировать во треугольнику. Также каждое из уравнений необходимо разрешить относительно производной, чтобы получить систему обыкновенных дифференциальных уравнений [16] вида:

$$A \cdot \frac{da}{dt} + B \cdot a + \dots = f. \quad (4.23)$$

Важно заметить, что каждое из трех исходных уравнений в частных производных дало по M обыкновенных дифференциальных уравнений, и в конечном итоге задача сводится к решению системы из $3 \cdot M$ дифференциальных уравнений.

В данной бакалаврской работе конечная система уравнений не была получена аналитически, так как для упрощения решения и минимизации ошибок был использован программный пакет `sympy` [17]. С помощью его и системы `Jupiter Notebook` три формулы для q_1 , q_2 и H соответственно были выведены, а далее запрограммированы с помощью функции `solve_ivp` из пакета `scipy` и приведены в приложении В.

5 Решение задачи

5.1 Построение графиков

Для того чтобы визуализировать результаты численных экспериментов, была написана функция, которая строит графики для q_1 , q_2 и H в различные моменты времени. Для лучшей визуализации был написан алгоритм, который комбинирует данные графики в GIF анимацию. Так как количество графиков для некоторых из примеров было достаточно велико, то для создания анимации в случае некоторых экспериментов пришлось писать данные в буфер обмена напрямую, так как в противном случае происходило переполнение и программа переставала работать.

Все графики строятся по изначальным точкам разбиения области. Для того чтобы получить более детальную картинку, была произведена кубическая интерполяция.

Для получения дополнительной информации об эксперименте была написана функция для визуализации графика функции тока:

$$\Psi = \int q_1(x_1, x_2) dx_2$$

Визуализация данной функции дает лучшее понимание о том, каким именно образом распространяются волны в водоеме. Для данных графиков тоже была произведена кубическая интерполяция.

5.2 Изменение формы водоёма

Для программной реализации триангулирования произвольной области был использован Python модуль triangle [1], который реализует триангуляцию как с помощью алгоритма Рапперта, так и с помощью алгоритма Чу.

Данный модуль получает на вход геометрию в формате POLY и возвращает список вершин, сегментов, пустых областей, треугольников, маркеров вершин и маркеров сегментов, которые представляют собой нерегулярную сетку для исходной области. Для структуризации этой информации была написана своя реализация для объекта «точка» и «треугольник». Экземпляр

класса «точка» хранит в себе координаты (x, y) и номер точки. Экземпляр класса «треугольник» хранит в себе список из трех точек, а также имеет методы для вычисления площади, базисных функций и интегрирования по данному треугольнику. Таким образом, в реализации результат триангуляции представляет из себя массив экземпляров класса «треугольник».

5.2.1 Пруд

В качестве самой простой геометрии был выбран квадратный пруд, в соответствии с рисунком 5.1.



Рисунок 5.1 — Квадратный пруд.

Для него было построено разбиение в соответствии с рисунком 5.2.

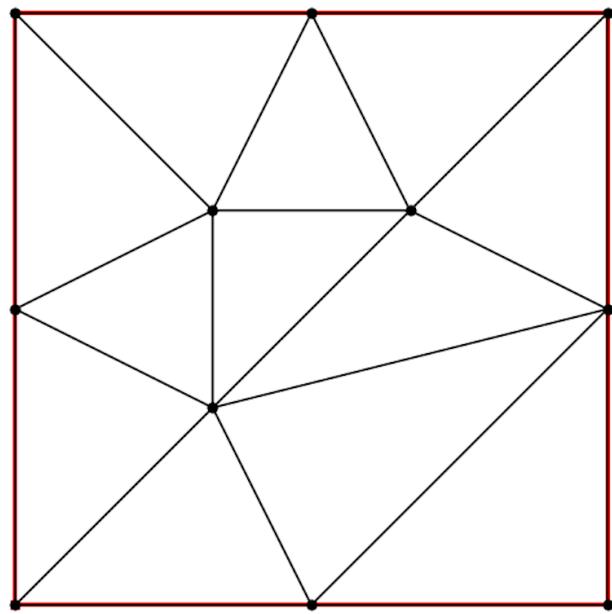


Рисунок 5.2 — Триангуляция квадратного пруда.

Данный пример рассматривался в моменты времени 0с, 1с и 2с. Для этих моментов времени были получены графики 5.3-5.6.

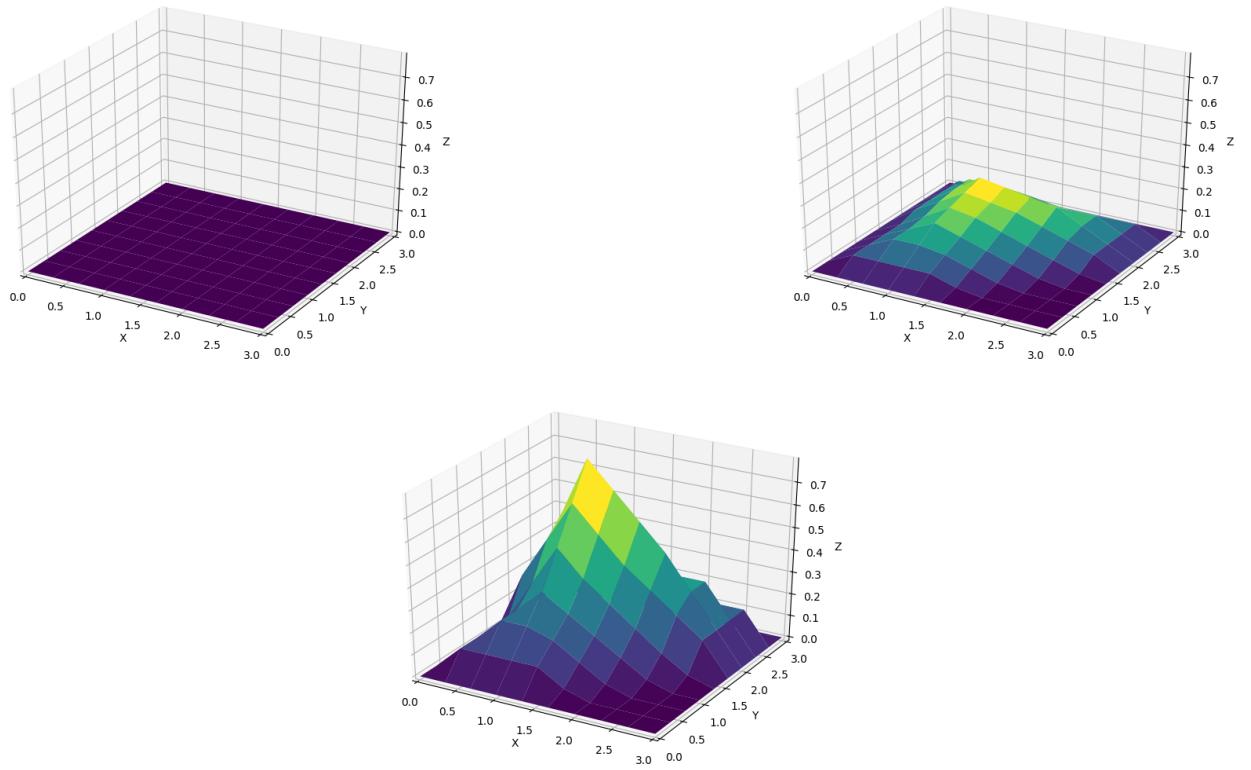


Рисунок 5.3 — Изменение проекции потока жидкости на ось x_1 в различные промежутки времени.

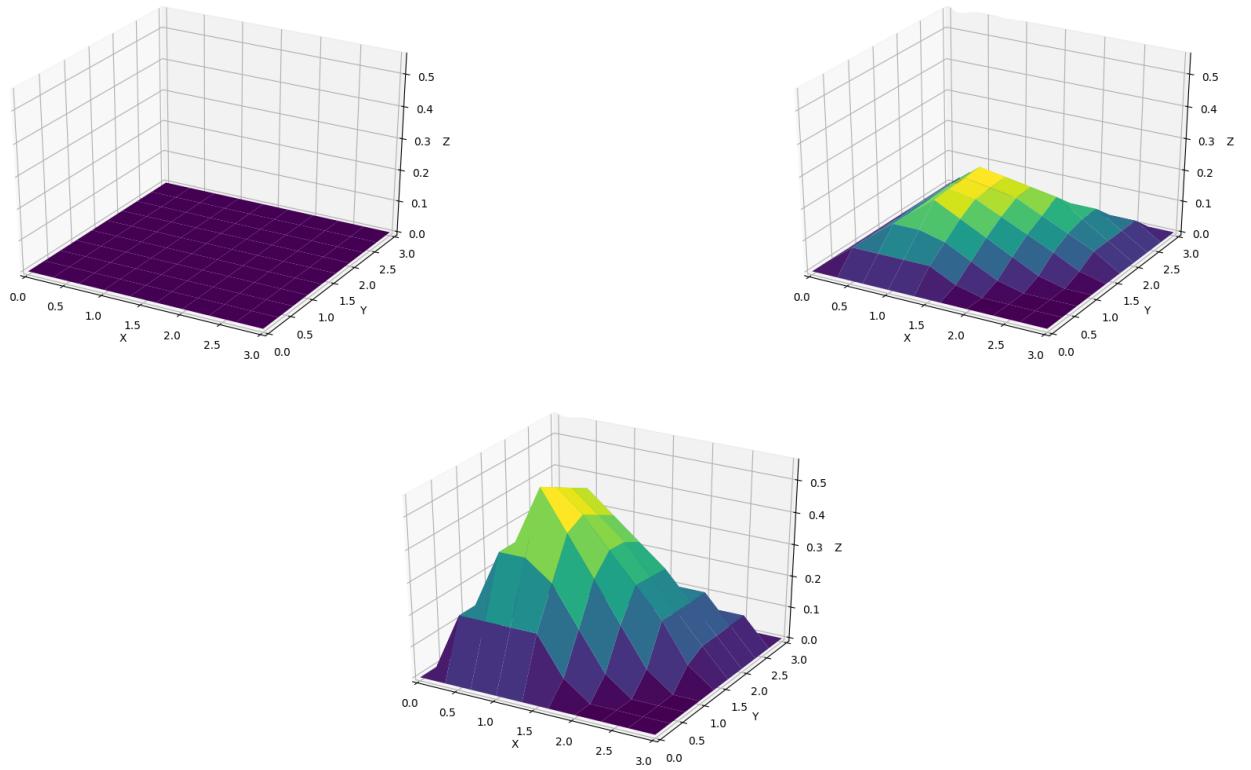


Рисунок 5.4 — Изменение проекции потока жидкости на ось x_2 в различные промежутки времени.

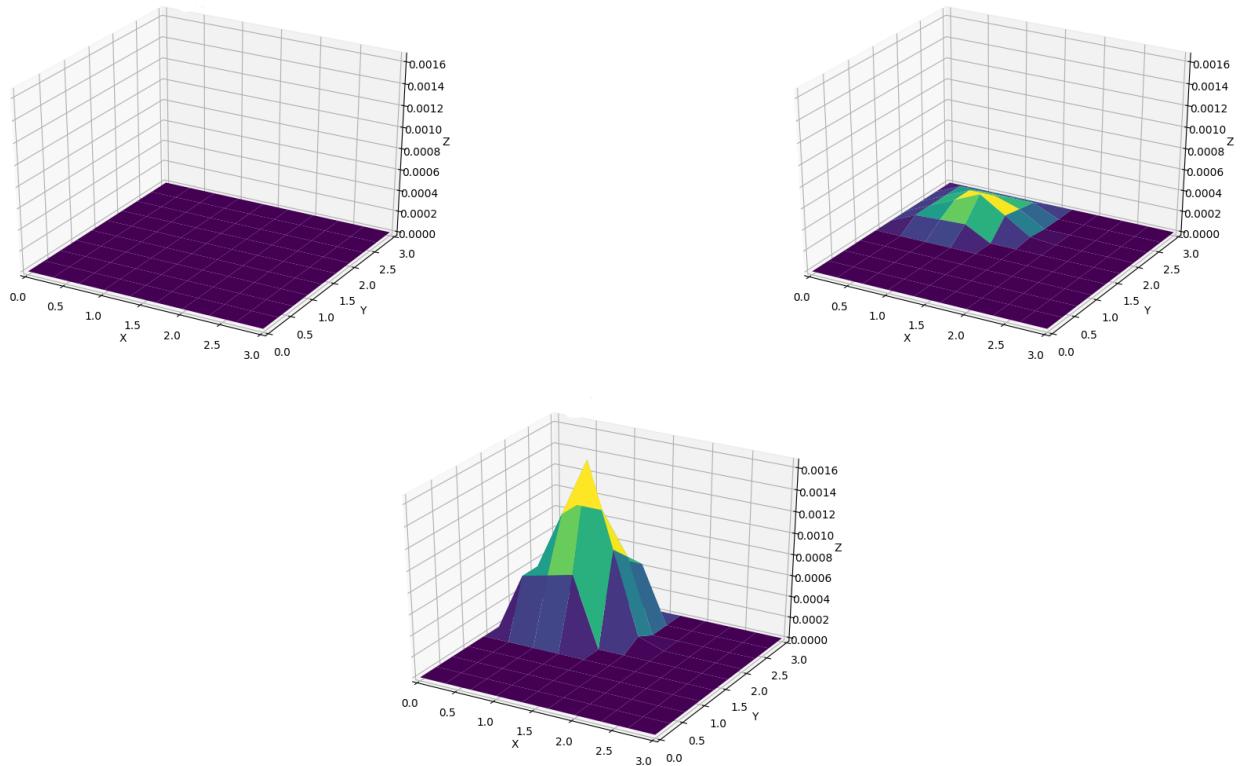


Рисунок 5.5 — Изменение высоты в различные промежутки времени.

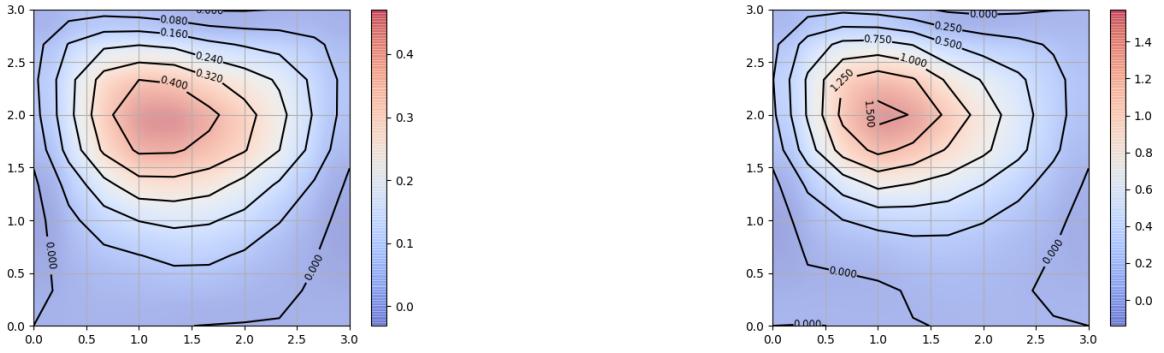


Рисунок 5.6 — Изменение функции тока в различные промежутки времени.

5.2.2 Реальный водоём

В качестве реального водоема, у которого нет островов, было выбрано озеро Эльтон – солёное бессточное самосадочное озеро на севере Прикаспийской низменности, изображенное на рисунке 5.7. Данное озеро имеет площадь в 152 квадратных километра и наибольшую глубину до 1.5 метров. В среднем глубина данного озера составляет 0.05 – 0.07 метра, а это значит, что для данного озера будут справедливыми уравнения мелкой воды.

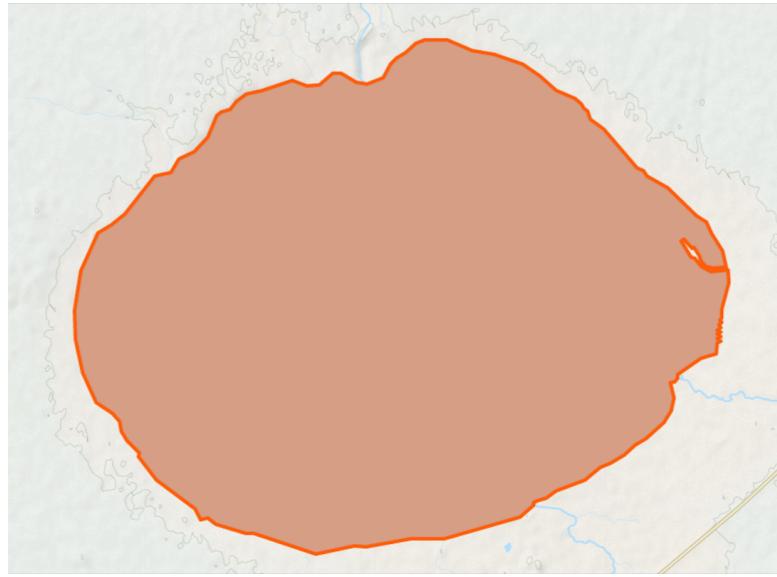


Рисунок 5.7 — Озеро Эльтон, выделенное на карте OSM.

Данные для расчета задачи по озеру Эльтон были получены на основании системы OSM [18] – некоммерческого веб-картографического проекта.

Для этого был использован сервис overpass-turbo [19], который является интерфейсом к API OSM и позволяет очень просто выполнять необходимые запросы. Запрос для озера Эльтон выглядит следующим образом:

```
[out:json][timeout:25];
(
  node["name"="Elton"]({{bbox}});
  way["name"="Elton"]({{bbox}});
  relation["name"="Elton"]({{bbox}});
);
out body;
>;
out skel qt;
```

Одним из результатов запроса будет *geojson* файл, в котором содержатся точки запрашиваемого географического объекта и некоторая метаинформация. Так как для триангуляции требуется файл с геометрией в формате POLY, то была написана соответствующая функция на языке *Python*, которая конвертирует формат *geojson* в формат POLY. Для озера Эльтон было построено разбиение в соответствии с рисунком 5.8.

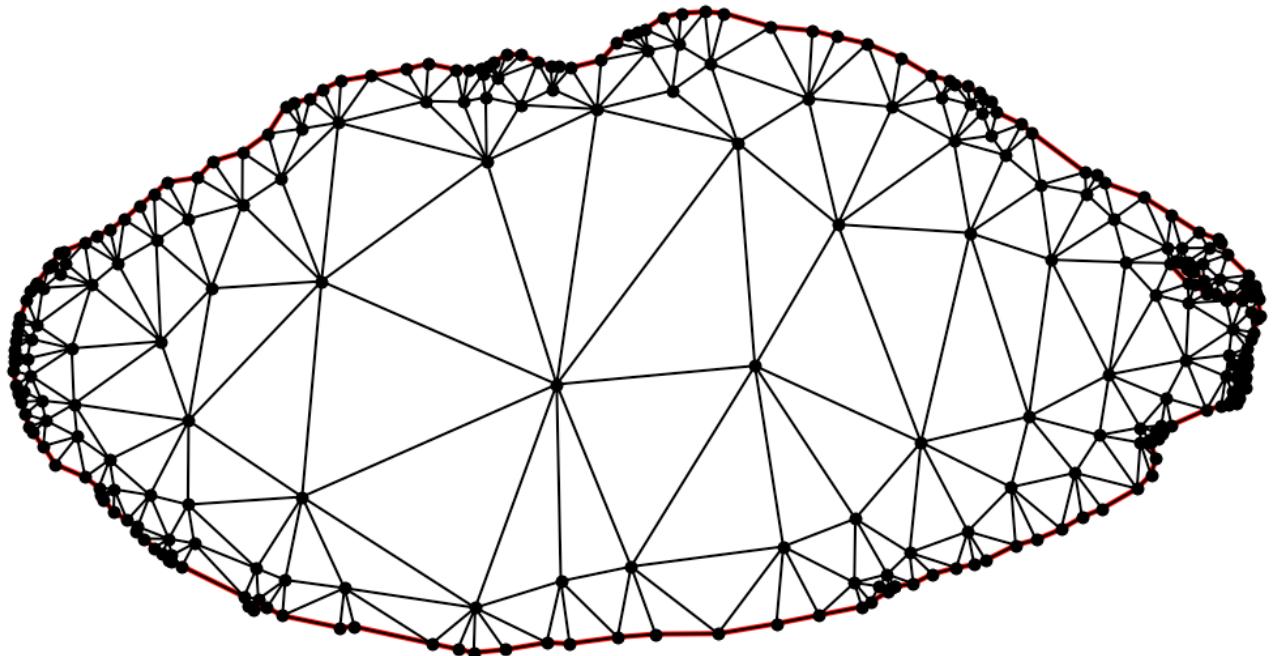


Рисунок 5.8 — Триангуляция озера Эльтон.

Данный пример рассматривался в моментах времени 1.902с и 1.905с. Для этих моментов времени были получены графики 5.9-5.12.

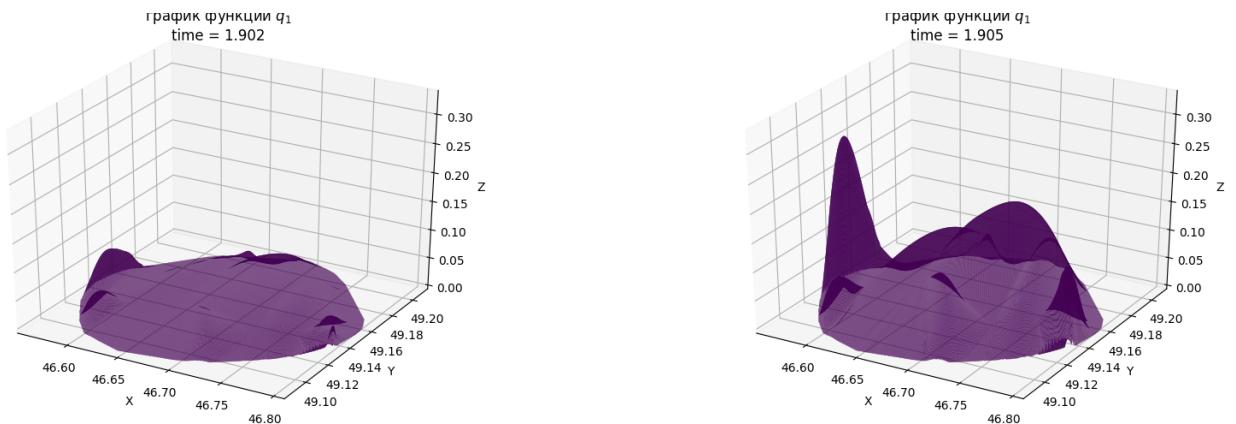


Рисунок 5.9 — Изменение проекции потока жидкости на ось x_1 в различные промежутки времени.

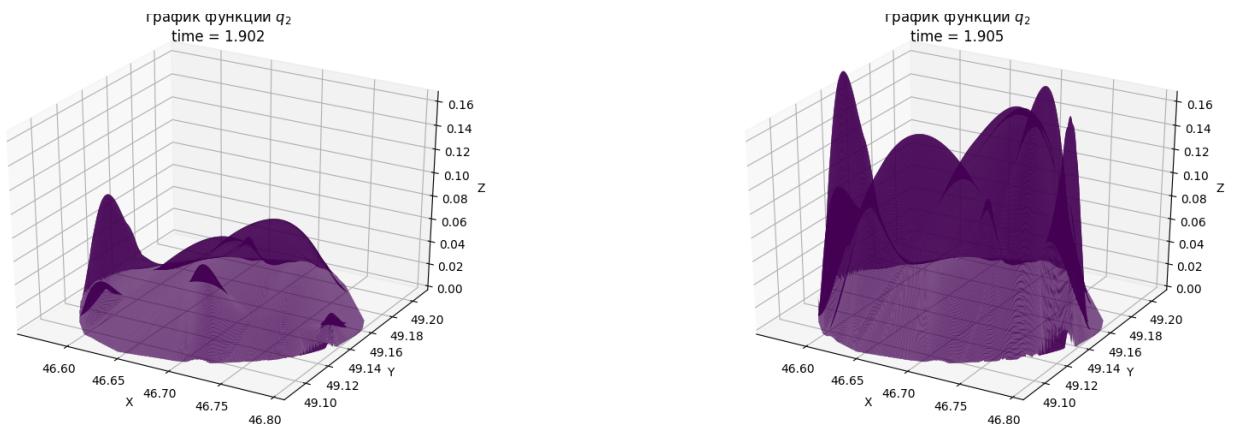


Рисунок 5.10 — Изменение проекции потока жидкости на ось x_2 в различные промежутки времени.

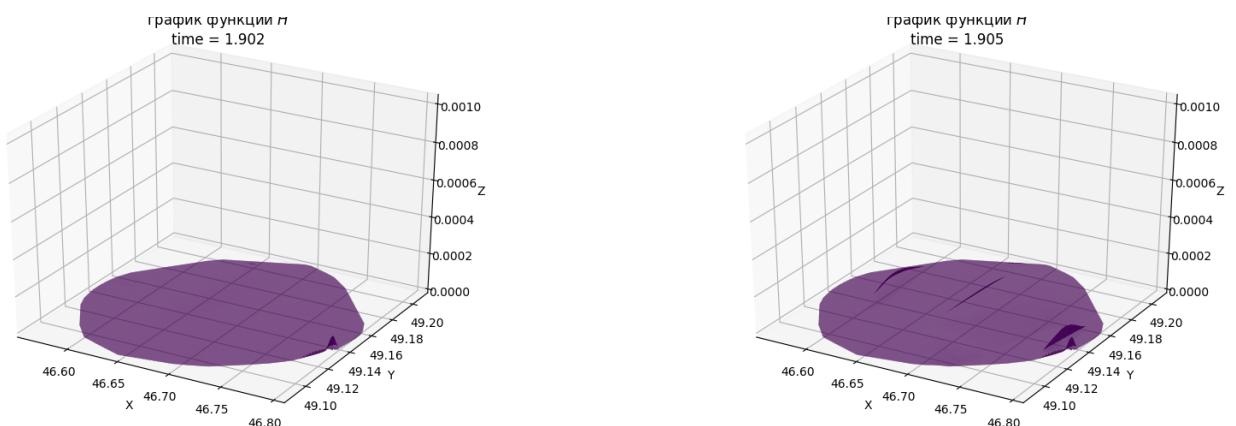


Рисунок 5.11 — Изменение высоты в различные промежутки времени.

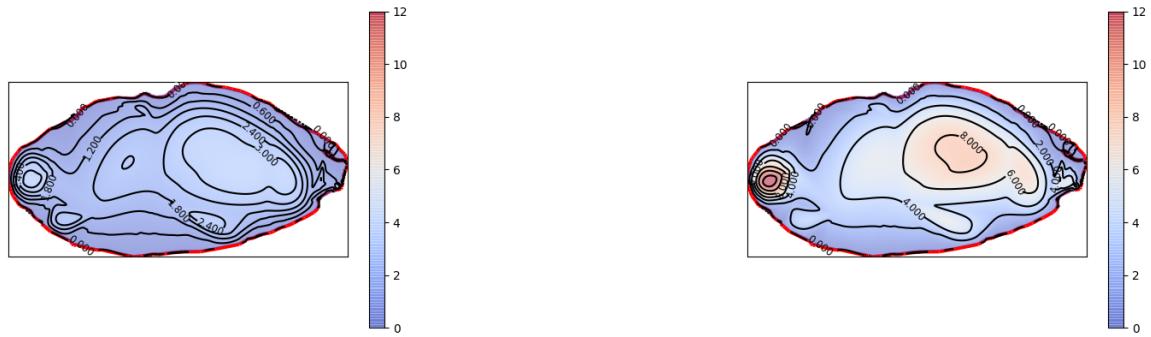


Рисунок 5.12 – Изменение функции тока в различные промежутки времени.

5.3 Учёт наличия острова (островов) внутри водоёма

5.3.1 Пруд

Для того чтобы проверить корректность алгоритма на сетках с «дырами», в пруд, рассмотренный ранее, добавлено два острова, в соответствии с рисунком 5.13.

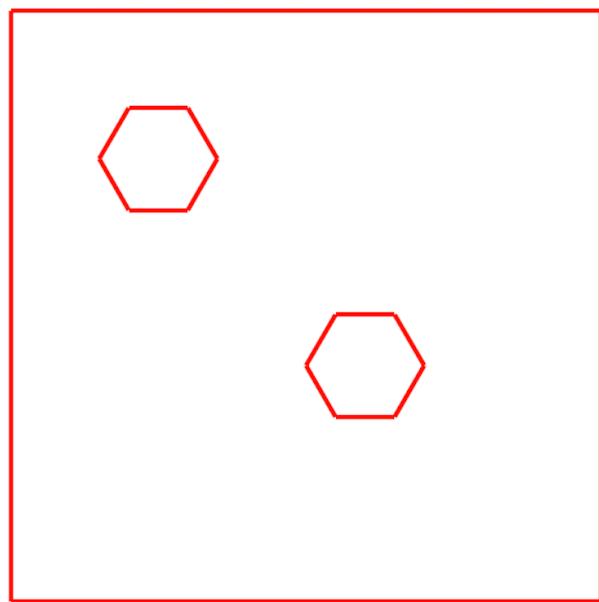


Рисунок 5.13 – Квадратный пруд с двумя островами.

Для этой геометрии было построено разбиение в соответствии с рисунком 5.14.

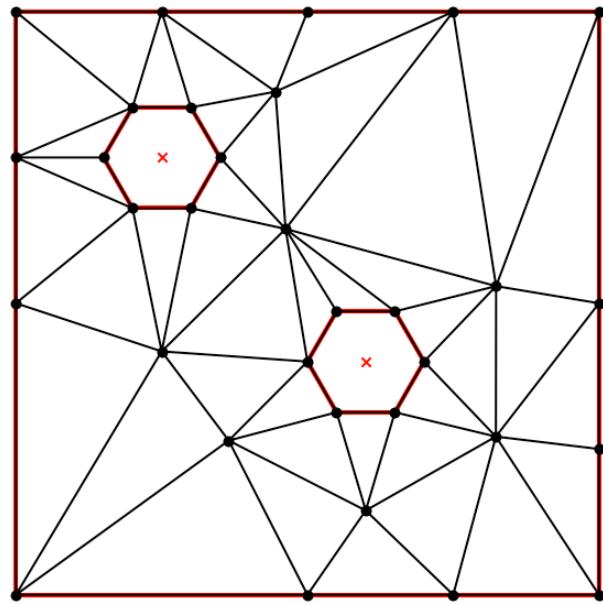


Рисунок 5.14 — Триангуляция прямоугольного пруда с двумя островами.

Данный пример рассматривался на двух моментах времени: 1.91с и 1.99с. Для этих моментов времени были получены графики 5.3-5.18.



Рисунок 5.15 — Изменение проекции потока жидкости на ось x_1 в различные промежутки времени.

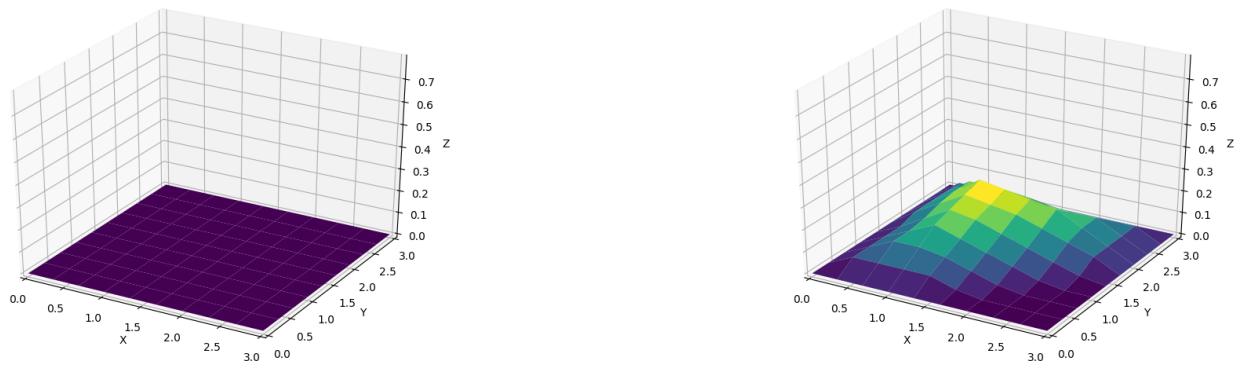


Рисунок 5.16 — Изменение проекции потока жидкости на ось x_2 в различные промежутки времени.

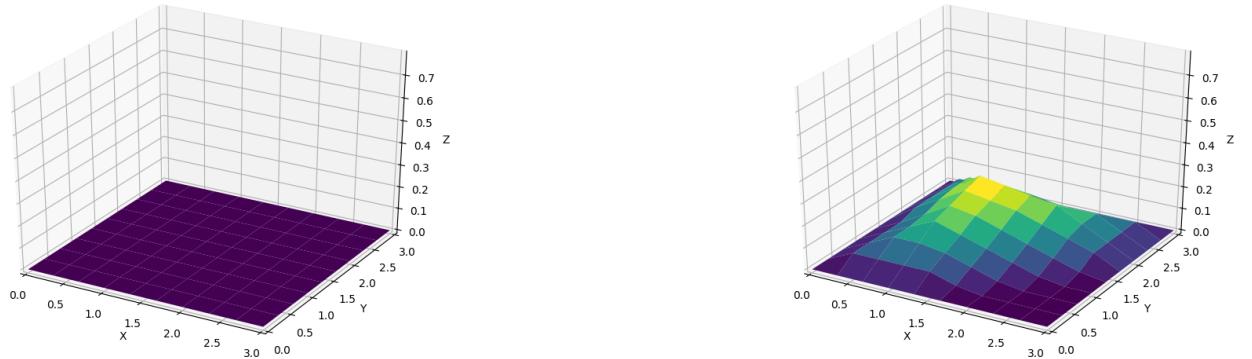


Рисунок 5.17 — Изменение высоты в различные промежутки времени.

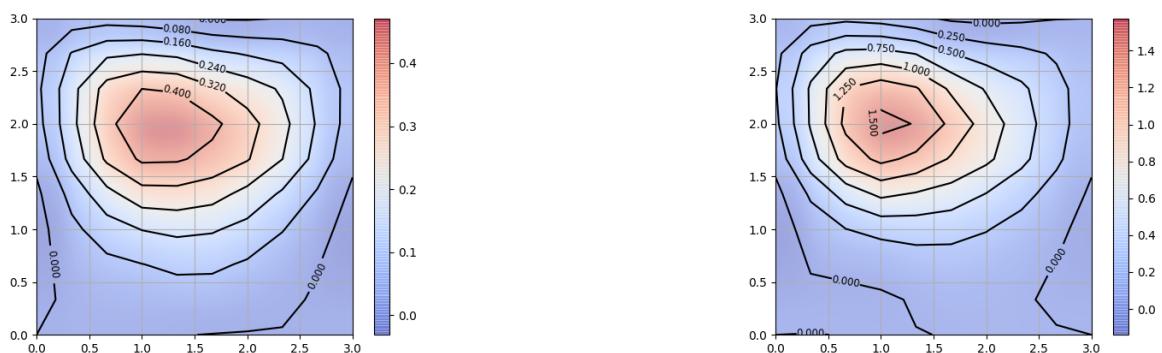


Рисунок 5.18 — Изменение функции тока в различные промежутки времени.

5.3.2 Реальный водоём

В качестве реального водоема, у которого есть острова, было выбрано озеро Верхнее, изображенное на рисунке 5.19. Оно находится в Северной Америке и является самым крупным и глубоким в системе Великих озёр. С физической точки зрения глубина этого озера составляет максимум 406 метров, а его размеры – 563 x 257 км. Получается, что для него можно использовать уравнения мелкой воды.

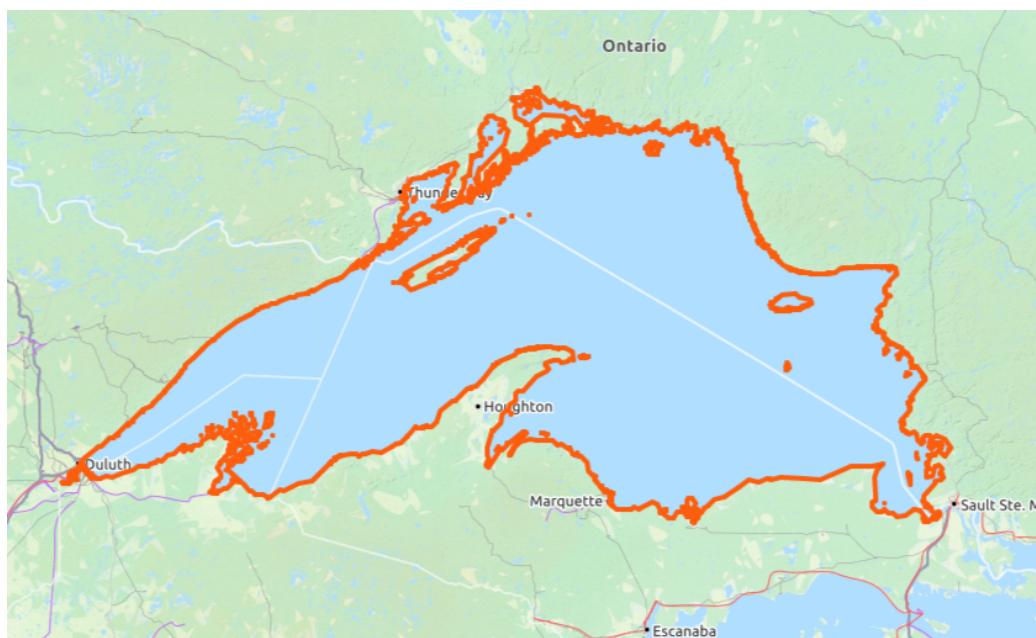


Рисунок 5.19 — Озеро Верхнее выделенное на карте OSM.

Для этой геометрии было построено разбиение в соответствии с рисунком 5.20.

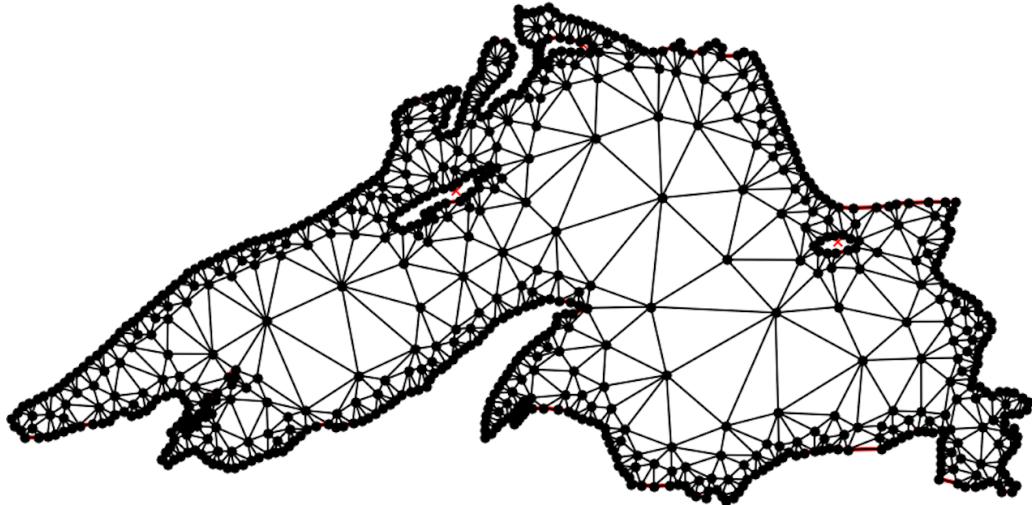


Рисунок 5.20 — Триангуляция озера Верхнее.

TODO: ГРАФИКИ БУДУТ ДОБАВЛЕНЫ ПОСЛЕ ТОГО, КАК ОЗЕРО ДОСЧИТАЕТСЯ

5.4 Программная реализация

МКЭ является очень ресурсоемким методом, и именно из-за этого он долго не применялся. Все современные задачи считаются на больших сетках, и занимает это достаточно большое количество времени.

Именно по этой причине реализация МКЭ в данной бакалаврской работе является многопоточной. В первую очередь это удалось сделать благодаря самому МКЭ, который позволяет считать элементы в сетке независимо друг от друга.

Для решения этой задачи был написан пулл потоков, который позволяет в один и тот же момент времени считать произвольное количество элементов, которое варьируется от 1 до n , где n – максимально возможное количество потоков, которое может одновременно исполнять компьютер на котором запущена программа.

Так как при использовании потоков Python не может задействовать все ядра процессора для выполнения нескольких потоков одновременно, то вместо них в реализации задачи были использованы процессы из библиотеки multiprocessing.

Помимо этого все модули приложения вместе с внешними зависимостями были упакованы в Docker контейнер, который позволяет очень просто переносить приложение на любую систему, которая поддерживает данную технологию. Данная упаковка позволила производить вычисления не на личном стационарном компьютере, а на удаленном сервере, который имеет намного большие вычислительные мощности и позволяет быстрее получить результат.

6 Разработка базы данных для хранения информации о проведённых экспериментах

Данные, полученные в результате численных экспериментов, являются разрозненными, так как они представляют из себя не только GIF анимацию, но и некоторую метаинформацию, которая представлена в виде JSON файлов. Из-за этого, помимо общепринятых требований к БД, следует основываясь на приведенной выше специфике данных, которые были получены в результате решения поставленной задачи, ввести следующие требования, которым должны удовлетворять БД и СУБД, управляющая ей:

1. Запись об эксперименте должна позволять хранить любую метаинформацию, которая может различаться в зависимости от проведенного эксперимента
2. БД должна позволять добавлять поля к уже существующим данным. Это, к примеру, нужно для добавления в запись данных, которые получены в результате постобработки эксперимента человеком
3. БД должна легко масштабироваться, так как данных может быть много
4. Должно быть отсутствие необходимости сопоставления объекта в базе и программе для упрощения разработки
5. Хранение графической информации должно быть реализовано максимально просто и эффективно

6.1 Описание базы данных

Основываясь на приведенных выше требованиях, в качестве СУБД решено выбрать MongoDB, которая классифицируется как NoSQL. MongoDB — документо-ориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Каждая запись — это документ без жёстко заданной схемы, который может содержать вложенные документы.

Так как документо-ориентированные базы данных еще не так распространены, как реляционные, но их популярность с каждым годом все увеличивается, целесообразно будет разъяснить некоторые нюансы терминологии документо-ориентированных баз данных.

В книге о MongoDB [20] приведены шесть основных концепций MongoDB:

1. MongoDB концептуально является тем же самым, что и привычная нам база данных (в терминологии Oracle RDBMS – схема). Внутри MongoDB может быть ноль или более баз данных, каждая из которых является контейнером для прочих сущностей.
2. База данных может иметь ноль или более «коллекций». «Коллекция» настолько похожа на традиционную «таблицу», что можно смело считать их одним и тем же.
3. «Коллекции» состоят из нуля или более «документов», каждый из которых можно рассматривать как «строку».
4. «Документ» состоит из одного или более «полей», которые подобны «колонкам».
5. «Индексы» в MongoDB почти идентичны таковым в реляционных базах данных.
6. «Курсыры» отличаются от предыдущих пяти концепций, но они очень важны. Когда мы запрашиваем у MongoDB какие-либо данные, то БД возвращает «курсор», с которым мы можем делать все что угодно: подсчитывать, пропускать определённое число предшествующих записей, - при этом не загружая сами данные.

Если резюмировать сказанное выше, то MongoDB состоит из «баз данных», которые, в свою очередь, состоят из «коллекций». «Коллекции» состоят из «документов», а каждый «документ» - из «полей». «Коллекции» могут быть проиндексированы, что улучшает производительность выборки и сортировки. И наконец, получение данных из MongoDB сводится к получению «курсора», который отдаёт эти данные по мере надобности.

Также стоит привести основные достоинства данной базы данных:

1. Документо-ориентированное хранилище (простая и мощная JSON-подобная схема данных)
2. Достаточно гибкий язык для формирования запросов
3. Динамические запросы
4. Полная поддержка индексов
5. Профилирование запросов

6. Эффективное хранение двоичных данных больших объёмов, например фото и видео
7. Журнализование операций, модифицирующих данные в БД
8. Поддержка отказоустойчивости и масштабируемости: асинхронная репликация, набор реплик и шардинг

Основными плюсами MongoDB, помимо её документо-ориентированности, для данной бакалаврской работы стали:

1. Схожий с реляционными СУБД подход к хранению данных, который построен на следующем: база данных-коллекция-документ-поле
2. Формат хранения данных. Данные в MongoDB хранятся в формате BSON. BSON - это бинарный формат, используемый для хранения документов и осуществления удаленного вызова процедур в MongoDB. Тот факт, что данные хранятся именно в бинарном виде, значительно ускоряет время их автоматической обработки
3. Простота разработки. MongoDB, в отличии от конкурентов, достаточно просто разворачивается и администрируется, а также имеет хороший API
4. GridFS – файловая система, которая поставляется вместе с MongoDB и служит для хранения больших файлов

6.2 Формат представления данных в БД

Так как в качестве СУБД была выбрана документо-ориентированная MongoDB, то и структура базы данных будет приводится в её терминологии.

Основной и единственной базой данных является база «эксперименты» («experiments»). В ней находится коллекция «данные» («data»). Данная коллекция содержит в себе документы, каждый из которых является проведенным экспериментом. Структура базы данных изображена на рисунке 6.1.

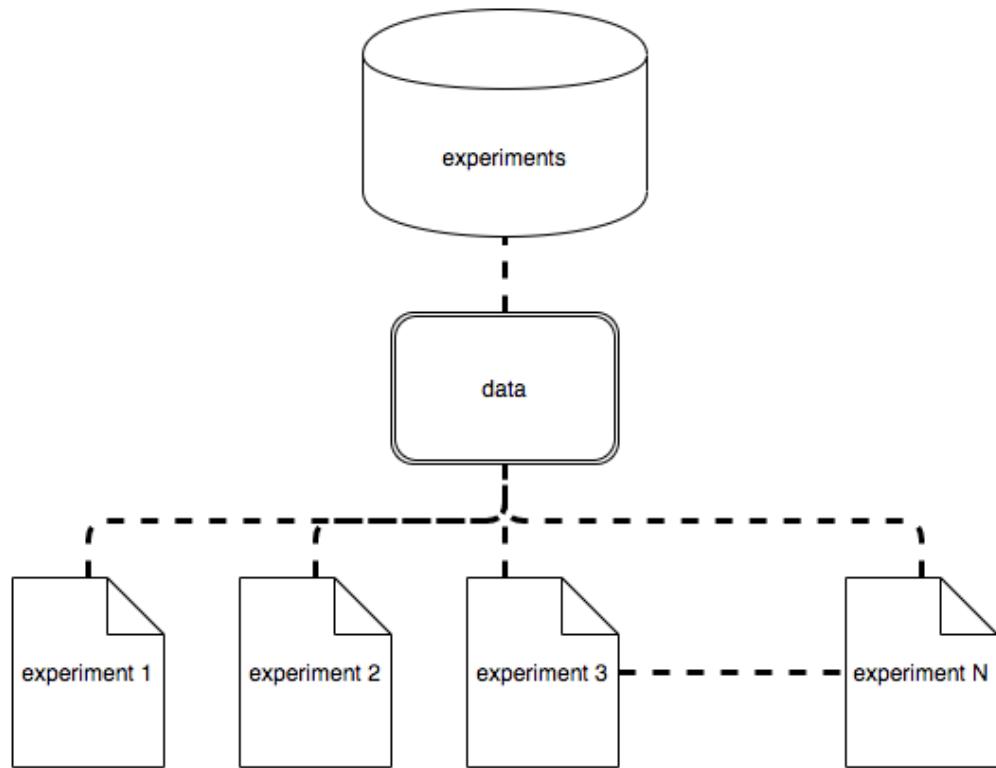


Рисунок 6.1 – Структура БД.

Каждый из содержащихся в БД документов хранит в себе следующие поля:

1. дата проведения эксперимента
2. время выполнения эксперимента
3. уникальный идентификатор, указывающий на изображения, полученные в результате экспериментов, которые хранятся в GridFS
4. матрица решения системы ОДУ
5. вектор, хранящий в себе моменты времени, в которых была решена ОДУ
6. тип и файл сетки
7. количество базисных функций

Документ, который описывает эксперимент, имеет следующую структуру (в подобном виде он хранится в MongoDB):

```

{
  "date": <date>,
  "images": {
    "frame": {
      "fluid_flow": {
        "1": frame_q1_id,
        "2": frame_q2_id
      },
      "surface_elevation": frame_H_id
    },
    "surface": {
      "fluid_flow": {
        "1": surf_q1_id,
        "2": surf_q2_id
      },
      "surface_elevation": surf_H_id
    },
    "stream_function": {
      "1": wave_psi1_id,
      "2": wave_psi2_id
    }
  },
  "solution": {
    "matrix": <matrix_data>,
    "times": <times_data>
  },
  "meta": {
    "mesh": {
      "type": mesh_type,
      "name": mesh_name
    },
    "execution_time": <time>,
    "quantity_of_basis_functions": <quantity_of_fe>
  }
}

```

Поля с постфиксами `id` являются уникальными идентификаторами для GIF анимаций, которые сохранены на файловой системе GridFS. Благодаря этому изображения результатов экспериментов не пропадут в случае отказа одной из нод кластера MongoDB. Таким образом обеспечивается надежная сохранность данных.

6.3 Примеры запросов к БД

Для того чтобы использовать данные, сохраненные в БД в результате экспериментов, можно использовать запросы вида:

```
db.data.find({ "date" : "2018-04-24-20-07" })
```

Это не очень удобно, так как результатами экспериментов является не только текстовая информация, но еще и графическая. В случае прямого запроса в ответе будут бинарные объекты, с которыми тяжело работать в чистом виде. Именно из-за этого для более простой работы с базой данных была написана функция `save(db, data, dir)`, которая находится в модуле `fem.db.mongo`. На вход данной функции подается JSON, который является результатом прямого запроса к базе, а также папка, в которую будут сохранены графики эксперимента. Помимо функции `save(db, data, dir)`, в том же пакете находится функция `read(db, collection, date)`, которая позволяет пользователю не делать прямые запросы в базу и облегчает работу с результатами. Комбинированный пример получения данных из базы выглядит следующим образом:

```
save(db, read(db, 'data', '2018-04-24-20-07'), '/data')
```

Данный вызов, во-первых, найдет в базе данных документ, в котором хранится информация о результатах эксперимента, который был проведен 2018-04-24-20-07, а во-вторых, сохранит все необходимые результаты на диск в папку `/data`. Это очень удобно, так как не требует от конечного пользователя знаний о структуре запросов в MongoDB.

ЗАКЛЮЧЕНИЕ

В представленной бакалаврской работе были получены уравнения мелкой воды при пренебрежении температурными эффектами, а также написана многопоточная программная реализация МКЭ для решений данных уравнений.

Реализована возможность переносимости разработанного алгоритма, а также возможность его удаленного использования посредством контейнеризации.

В качестве примера были произведены расчеты на различных сетках, в том числе и на реальных озерах(Эльтон и Верхнее). Для данных расчетов были построены графики на которых изображены изменения потока жидкости и высоты волны, кроме того была сконструированна функция тока для которой тоже были построены соответствующие графики. Благодаря данным результатам удалось качественно оценить полученные результаты проведенных экспериментов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. — URL: <https://www.cs.cmu.edu/~quake/triangle.html> (дата обращения: 22.04.2017).
2. Норри., Д., Фриз., Ж. Введение в метод конечных элементов. — М.: Мир, 1981. — 304 с.
3. Панкратов, И.А., Рымчук, Д.С. Расчет течения мелкой воды // Математика. Механика. — 2014. — Т. 16. — С. 121–124.
4. Панкратов, И.А. Об аппроксимации нестационарных уравнений мелкой воды // Juvenis scientia. — 2016. — Т. 5. — С. 3–4.
5. Lutz, M. Learning Python, 5th Edition. — O'Reilly Media, 2013. — 1648 p. — ISBN: 978-1449355739.
6. Kiusalaas, J. Numerical Methods in Engineering with Python 3, 3rd Edition. — New York: Cambridge University Press, 2013. — 432 p. — ISBN: 978-1107033856.
7. Коннор, Дж., Бреббиа, К. Метод конечных элементов в механике жидкости. — Л.: Судостроение, 1979. — 264 с.
8. Зорич, В. А. Математический анализ. — М.: ФАЗИС, 1997. — Т. 1. — 554 с. — ISBN: 5-7036-0031-6.
9. Юрко, В. А. Уравнения математической физики : учеб. пособие для студентов механико-математического и физического факультетов. — Саратов: Изд-во Сарат. ун-та, 2004. — 118 с.
10. Скворцов, А. В. Триангуляция Делоне и её применение. — Издательство Томского университета, 2002. — 42 с. — ISBN: 5-7511-1501-5.
11. Ruppert, J. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. — 1995. — Vol. 18. — P. 548–585.

12. Rand, A. Where and How Chew's Second Delaunay Refinement Algorithm Works. — URL: <http://2011.cccg.ca/PDFschedule/papers/paper91.pdf> (дата обращения: 11.12.2017).
13. Панкратов, И.А. Введение в методы взвешенных невязок. — URL: http://www.sgu.ru/sites/default/files/textdocsfiles/2013/12/10/fem_introduction.pdf (дата обращения: 20.11.2017).
14. Зенкевич, О., Морган, К. Конечные элементы и аппроксимация. — М.: Мир, 1986. — 318 с.
15. Голованов, Н. Н. Геометрическое моделирование. — М.: Издательство Физико-математической литературы, 2002. — 472 с. — ISBN: 978-5-7695-7168-8.
16. Гуревич, А. П. Основы теории обыкновенных дифференциальных уравнений. — Саратов: Изд-во СГУ, 2013. — 176 с.
17. Mehta, H. K. Mastering Python Scientific Computing. — Packt Publishing, 2015. — 345 p. — ISBN: 978-1783288823.
18. OpenStreetMap. — URL: <https://www.openstreetmap.org> (дата обращения: 27.01.2018).
19. Over pass turbo. — URL: <https://overpass-turbo.eu/> (дата обращения: 28.01.2018).
20. Seguin., K. The Little MongoDB Book. — Self-Published, 2013. — 34 p. — ISBN: 978-1493786602.

ПРИЛОЖЕНИЕ А

Исходный код реализации

Для того, чтобы код можно было легко поддерживать и переиспользовать было произведено разбиение на пакеты в соответствии с рисунком А.1.

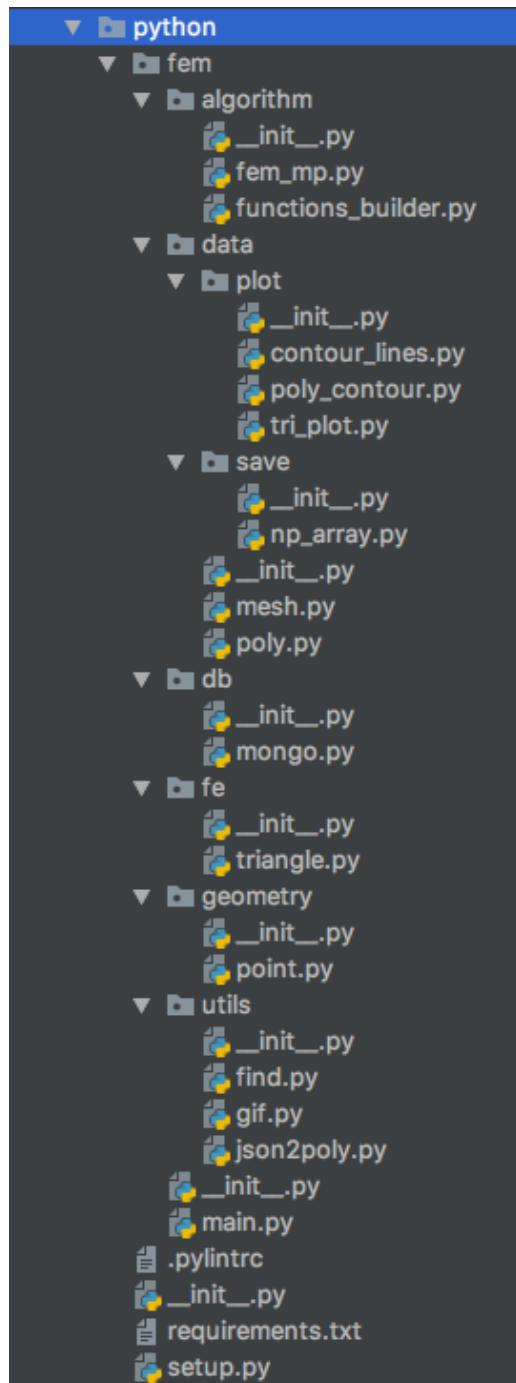


Рисунок А.1 — Структура пакетов программной реализации метода конечных элементов.

Листинг А.1: Модуль fem_mp.py

```
1  # -*- coding: utf-8 -*-
```

```

2 # @Author: Alexander Sharov
3
4 import os
5 import time
6 import logging
7
8 from numpy import abs
9 from scipy.integrate import solve_ivp
10 from multiprocessing import Process, Manager
11 from sympy import *
12
13 # from threading import Thread
14
15 # CONSTANTS
16 ##### [ / ^3]
17 # rho = 1000 [ / ^3]
18 rho_a = 1.2754 [ ]
19 # P_a = 10 ** 5 [ ]
20 # W = 1 [ / c]
21 # gamma = 0.002 [ * c^2]
22 g = 9.832 [ ^{1/2} * c^{-1}]
23 # g / c^2
24 gc2 = 0.002 [ ]
25 # C = 40 [ ^{1/2} * c^{-1}]
26 H0 = 0.01
27
28
29
30
31
32
33
34
35
36
37 class Solver(object):
38     def __init__(self, max_tasks, mesh, t_span, t_eval):
39         self.mesh = mesh
40         self.M = mesh.quantity
41         self.elements = mesh.splitting
42         self.contour = mesh.contour
43         self.t_span = t_span
44         self.t_eval = t_eval
45         self.sys_fun = Manager().list([0] * (3 * self.M))
46         self.max_tasks = max_tasks
47
48     def on_boundary(self, element, number):
49         if element[1].number == number:
50             x, y = element[1].x, element[1].y
51         elif element[2].number == number:
52             x, y = element[2].x, element[2].y
53         elif element[3].number == number:
54             x, y = element[3].x, element[3].y
55         else:
56             return False
57
58         for point in self.contour:
59             if (abs(float(point[0]) - x)) < 1e-6 and (abs(float(point[1]) - y)) < 1e-6:
60                 return True
61
62     return False
63
64     def calculate_element(self, element, variables):
65
66         x1, x2 = symbols('x_1 x_2')
67         f_eq1 = 0
68         f_eq2 = 0
69         f_eq3 = 0
70
71         for k in range(0, self.M):
72             if not self.on_boundary(element, k):
73                 n_k = element.get_basic_function_by_number(k)
74                 weight_functions = 0

```

```

75     if k == 0:
76         logging.debug('[Process:%s] I_am_alive' % os.getpid())
77     elif k % 50 == 0:
78         logging.debug('[Process:%s] I_am_still_alive.[element=%s, k=%d]' % (os.
79                         getpid(), str(element), k))
80
81     for l in range(0, self.M):
82         w_l = element.get_basic_function_by_number(l)
83         weight_functions += w_l
84         f_eq1 += \
85             + element.integrate(w_l * diff(
86                 -P_a * H0 - P_a * variables[2 * self.M + k] * n_k - (
87                     g * rho / 2) * H0 ** 2) - g * rho * H0 *
88                     variables[2 * self.M + k] * n_k - g * rho / 2 * variables[2 * self.
89                         .M + k] ** 2 * n_k ** 2,
90                         x1)) \
91             + element.integrate(P_a * w_l * diff(H0 + variables[2 * self.M + k] *
92                 n_k, x1)) \
93             + element.integrate(sqrt(2) / 2 * W ** 2 * gamma * rho_a * w_l) \
94             - element.integrate(
95                 (gc2 * w_l * variables[k] / (rho * H0 ** 2) * variables[2 * self.M
96                     + k] ** 2 * n_k) * sqrt(
97                         variables[k] ** 2 * n_k ** 2 + variables[self.M + k] ** 2 *
98                         n_k ** 2))
99
100            f_eq2 += \
101                + element.integrate(w_l * diff(
102                    -P_a * H0 - P_a * variables[2 * self.M + k] * n_k - (
103                        g * rho / 2) * H0 ** 2) - g * rho * H0 *
104                        variables[2 * self.M + k] * n_k - g * rho / 2 * variables[2 * self.
105                            .M + k] ** 2 * n_k ** 2,
106                            x2)) \
107                            + element.integrate(P_a * w_l * diff(H0 + variables[2 * self.M + k] *
108                                n_k, x2)) \
109                                + element.integrate(sqrt(2) / 2 * W ** 2 * gamma * rho_a * w_l) \
110                                - element.integrate(
111                                    (gc2 * w_l * variables[self.M + k] / (rho * H0 ** 2) * variables[
112                                        2 * self.M + k] ** 2 * n_k) * sqrt(
113                                            variables[k] ** 2 * n_k ** 2 + variables[self.M + k] ** 2 *
114                                            n_k ** 2))
115
116            f_eq3 += \
117                - element.integrate(w_l * diff(n_k, x1)) * variables[k] \
118                - element.integrate(w_l * diff(n_k, x2)) * variables[self.M + k]
119
120            coefficient_of_d_eq1 = element.integrate(weight_functions * n_k)
121            coefficient_of_d_eq2 = element.integrate(weight_functions * n_k)
122            coefficient_of_d_eq3 = element.integrate(rho * weight_functions * n_k)
123
124            if coefficient_of_d_eq1 != 0:
125                self.sys_fun[k] = Float((f_eq1.evalf() / coefficient_of_d_eq1))
126
127            if coefficient_of_d_eq2 != 0:
128                self.sys_fun[self.M + k] = Float((f_eq2.evalf() / coefficient_of_d_eq2))
129
130            if coefficient_of_d_eq3 != 0:
131                self.sys_fun[2 * self.M + k] = Float((f_eq3.evalf() / coefficient_of_d_eq3))
132
133        else:
134            self.sys_fun[k] = 0
135            self.sys_fun[k + self.M] = 0
136            self.sys_fun[k + 2 * self.M] = 0
137
138        logging.debug('[Process:%s] I_am_dead' % os.getpid())
139
140    def system(self, time, variables):
141        logging.info('time=%s' % time)
142
143        elements_copy = self.elements.copy()
144
145        if self.max_tasks > 0:
146            while len(elements_copy) > 0:
147                tasks = []
148                cur_len = len(elements_copy)

```

```

139
140         for ind in range(0, self.max_tasks):
141             if ind < cur_len:
142                 # thread = Thread(target=self.calculate_element, args=(elements_copy[
143                     ind], variables))
143                 process = Process(target=self.calculate_element, args=(elements_copy[
144                     ind], variables))
145             else:
146                 continue
147             tasks.append(process)
148
149             for task in tasks:
150                 task.start()
151
152             for task in tasks:
153                 task.join()
154
155             if self.max_tasks < cur_len:
156                 elements_copy = elements_copy[self.max_tasks:]
157             else:
158                 break
159
160             logging.info('system_of_functions=%s\n' % self.sys_fun)
161
162     return self.sys_fun
163
163 def solve(self):
164     start_time = time.time()
165     logging.info('Number_of_elements=%d' % self.M)
166     y0 = [0] * (3 * self.M)
167     solution = solve_ivp(self.system,
168                           y0=y0,
169                           t_span=self.t_span,
170                           t_eval=self.t_eval,
171                           method='RK45',
172                           rtol=1e-3,
173                           atol=1e-3)
174
175     # it needs only for debug
176     logging.info(solution.y)
177     logging.info(solution.t)
178
179     execution_time = (time.time() - start_time) / 60
180
181     return solution.y, solution.t, execution_time

```

Листинг А.2: Модуль functions_builder.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4  from sympy import Symbol, symbols, integrate
5  from numpy import array
6
7  from geometry.point import Point
8
9
10 def get_functions(mesh, solution, times):
11     elements = mesh.splitting
12     points = mesh.points
13     M = mesh.quantity
14
15     q1_data = []
16     q2_data = []
17     H_data = []
18
19     psi1_data = []
20     psi2_data = []
21
22     for ind in range(0, len(times)):
23         q1_at_time = []
24         q2_at_time = []

```

```

25     H_at_time = []
26
27     # q1 = d /dx2
28     psi1_at_time = []
29
30     # TODO: probably we don't need psi_2 function
31     # q2 = - d /dx1
32     psi2_at_time = []
33
34     for point in points:
35         x1, x2 = point[0], point[1]
36         q1 = 0
37         q2 = 0
38         H = 0
39
40         for element in elements:
41             if element.contains(Point(x1, x2)):
42                 i, j, k = element.vertices_number
43
44                 sub = {
45                     symbols('x_1'): x1,
46                     symbols('x_2'): x2
47                 }
48
49                 n_i = (element.get_basic_function_by_number(i)).subs(sub)
50                 n_j = (element.get_basic_function_by_number(j)).subs(sub)
51                 n_k = (element.get_basic_function_by_number(k)).subs(sub)
52
53                 q1 += solution[i][ind] * n_i + solution[j][ind] * n_j + solution[k][ind] *
54                     n_k
55                 q2 += solution[i + M][ind] * n_i + solution[j + M][ind] * n_j + solution[k +
56                     M][ind] * n_k
57                 H += solution[i + 2 * M][ind] * n_i + solution[j + 2 * M][ind] * n_j +
58                     solution[k + 2 * M][
59                         ind] * n_k
60
61                 q1_at_time.append([x1, x2, q1])
62                 q2_at_time.append([x1, x2, q2])
63                 H_at_time.append([x1, x2, H])
64
65                 psi1_at_time.append([x1, x2, integrate(q1, Symbol('x_2')).subs(Symbol('x_2'), x2)
66                     ])
67                 psi2_at_time.append([x1, x2, -integrate(q1, Symbol('x_1')).subs(Symbol('x_1'), x1)
68                     ])
69
70                 psi1_data.append(array(psi1_at_time))
71                 psi2_data.append(array(psi2_at_time))
72
73             return q1_data, q2_data, H_data, psi1_data, psi2_data

```

Листинг А.3: Модуль poly_contour.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3  # This is modification of plot module from triangle package
4
5
6  def plot(ax, **kw):
7      ax.axes.set_aspect('equal')
8
9      if 'segments' in kw: segments(ax, **kw)
10     if 'holes' in kw: holes(ax, **kw)
11     if 'edges' in kw: edges(ax, **kw)
12
13     ax.get_xaxis().set_visible(False)
14     ax.get_yaxis().set_visible(False)
15
16

```

```

17 def vertices(ax, **kw):
18     verts = kw[ 'vertices' ]
19     ax.scatter(*verts.T, color='k')
20     if 'labels' in kw:
21         for i in range(verts.shape[0]):
22             ax.text(verts[i, 0], verts[i, 1], str(i))
23     if 'markers' in kw:
24         vm = kw[ 'vertex_markers' ]
25         for i in range(verts.shape[0]):
26             ax.text(verts[i, 0], verts[i, 1], str(vm[i]))
27
28
29 def segments(ax, **kw):
30     verts = kw[ 'vertices' ]
31     segs = kw[ 'segments' ]
32     for beg, end in segs:
33         x0, y0 = verts[beg, :]
34         x1, y1 = verts[end, :]
35         ax.fill([x0, x1], [y0, y1],
36                 facecolor='none', edgecolor='r', linewidth=3,
37                 zorder=0)
38
39
40 def holes(ax, **kw):
41     ax.scatter(*kw[ 'holes' ].T, marker=' ', color='r')
42
43
44 def edges(ax, **kw):
45     verts = kw[ 'vertices' ]
46     edges = kw[ 'edges' ]
47     for beg, end in edges:
48         x0, y0 = verts[beg, :]
49         x1, y1 = verts[end, :]
50         ax.fill([x0, x1], [y0, y1], facecolor='none', edgecolor='k', linewidth=.5)
51
52     if ('ray_origins' not in kw) or ('ray_directions' not in kw):
53         return
54
55     lim = ax.axis()
56     ray_origin = kw[ 'ray_origins' ]
57     ray_direct = kw[ 'ray_directions' ]
58     for (beg, (vx, vy)) in zip(ray_origin.flatten(), ray_direct):
59         x0, y0 = verts[beg, :]
60         scale = 100.0 # some large number
61         x1, y1 = x0 + scale * vx, y0 + scale * vy
62         ax.fill([x0, x1], [y0, y1], facecolor='none', edgecolor='k', linewidth=.5)
63     ax.axis(lim) # make sure figure is not rescaled by infinite ray

```

Листинг А.4: Модуль contour_lines.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4  import os
5  import numpy as np
6  import matplotlib.pyplot as plt
7  import io
8  import datetime
9
10 from PIL import Image
11 from scipy.interpolate import griddata
12 from utils import find
13 from data.plot.poly_contour import plot as dplot
14
15 INTERPOLATION_NODES = 500
16
17
18 def draw_psi_2d(path, title, functions, times, mesh):
19     os.makedirs(path + '/' + title, exist_ok=True)
20
21     x_max = find.maximum(functions, 'x')
22     x_min = find.minimum(functions, 'x')

```

```

23     y_max = find.maximum(functions, 'y')
24     y_min = find.minimum(functions, 'y')
25
26     for func, time in zip(functions, times):
27         plt.figure()
28
29         ax = plt.subplot(111, aspect='equal')
30         dplot(ax, **mesh.raw_splitting)
31
32         x = func[:, 0]
33         y = func[:, 1]
34         z = func[:, 2]
35
36         xi = np.linspace(x_min, x_max, INTERPOLATION_NODES)
37         yi = np.linspace(y_min, y_max, INTERPOLATION_NODES)
38
39         X, Y = np.meshgrid(xi, yi)
40         Z = griddata((x, y), z, (X, Y), method='cubic')
41
42         CS = plt.contour(X, Y, Z, colors='black')
43
44         plt.clabel(CS, fontsize='small', inline=10)
45         plt.imshow(Z, extent=[x_min, x_max, y_min, y_max],
46                     origin='lower', cmap='coolwarm',
47                     interpolation='kaiser', alpha=0.5,
48                     vmin=0, vmax=12)
49
50         plt.colorbar()
51         plt.grid()
52         plt.title('' + '$\\%s$' % title + '\n' + 'time = %s' % time)
53
54         buf = io.BytesIO()
55
56         plt.savefig(buf, format='png')
57         plt.close()
58
59         buf.seek(0)
60         im = Image.open(buf)
61         im.save(path + '/' + title + '/%s.png' % datetime.datetime.now().strftime('%d_%m_%Y_%H_%M_%S_%f'), 'PNG')
62         buf.close()

```

Листинг А.5: Модуль tri_plot.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4  import matplotlib.pyplot as plt
5  import datetime
6  import numpy as np
7  import os
8  import scipy.interpolate as interp
9  import io
10
11 from PIL import Image
12 from mpl_toolkits.mplot3d import Axes3D
13
14 from utils import find
15
16 INTERPOLATION_NODES = 500
17
18 # This option needs for escaping 'RuntimeWarning: More than 20 figures have been opened'
19 plt.rcParams.update({'figure.max_open_warning': 0})
20
21
22 def draw_3d_frame(directory, title, functions, times):
23     os.makedirs(directory + '/' + title, exist_ok=True)
24
25     z_max = find.maximum(functions, 'z')
26
27     for func, time in zip(functions, times):
28         fig = plt.figure()

```

```

29     ax = fig.add_subplot(111, projection='3d')
30     ax.set_xlabel('X')
31     ax.set_ylabel('Y')
32     ax.set_zlabel('Z')
33
34     X = func[:, 0]
35     Y = func[:, 1]
36     Z = func[:, 2]
37
38     ax.set_xlim3d(np.min(X), np.max(X))
39     ax.set_ylim3d(np.min(Y), np.max(Y))
40     ax.set_zlim3d(0, z_max)
41
42     ax.plot(X, Y, Z)
43
44     plt.title('' + title + '\n' + 'time=' + time)
45     buf = io.BytesIO()
46
47     plt.savefig(buf, format='png')
48     plt.close()
49
50     buf.seek(0)
51     im = Image.open(buf)
52     im.save(directory + '/' + title + '/%s.png' % datetime.datetime.now().strftime('%d_%m_'
53                               '%Y_%H_%M_%S_%f'), 'PNG')
54     buf.close()
55
56 def draw_3d_surf(directory, title, functions, times, view='surface'):
57     os.makedirs(directory + '/' + title, exist_ok=True)
58
59     z_max = find.maximum(functions, 'z')
60
61     for func, time in zip(functions, times):
62         fig = plt.figure()
63         ax = Axes3D(fig)
64         ax.set_xlabel('X')
65         ax.set_ylabel('Y')
66         ax.set_zlabel('Z')
67
68         X = func[:, 0]
69         Y = func[:, 1]
70         Z = func[:, 2]
71
72         ax.set_xlim3d(np.min(X), np.max(X))
73         ax.set_ylim3d(np.min(Y), np.max(Y))
74         ax.set_zlim3d(0, z_max)
75
76         plotx, ploty = np.meshgrid(np.linspace(np.min(X), np.max(X), INTERPOLATION_NODES), \
77                                     np.linspace(np.min(Y), np.max(Y), INTERPOLATION_NODES))
78         plotz = interp.griddata((X, Y), Z, (plotx, ploty), method='cubic')
79
80         # Explanation: TODO
81         plotz[plotz < 0] = 0.0
82
83         if view == 'surface':
84             ax.plot_surface(plotx, ploty, plotz, cstride=1, rstride=1, cmap='viridis')
85         elif view == 'wireframe':
86             ax.plot_wireframe(plotx, ploty, plotz, cstride=1, rstride=1, cmap='viridis')
87
88         plt.title('' + title + '\n' + 'time=' + time)
89
90         buf = io.BytesIO()
91
92         plt.savefig(buf, format='png')
93         plt.close()
94
95         buf.seek(0)
96         im = Image.open(buf)
97         im.save(directory + '/' + title + '/%s.png' % datetime.datetime.now().strftime('%d_%m_'
98                               '%Y_%H_%M_%S_%f'), 'PNG')
99         buf.close()

```

Листинг А.6: Модуль np_array.py

```

1 # -*- coding: utf-8 -*-
2 # @Author: Alexander Sharov
3
4 import json
5 import os
6
7
8 def write(directory, name, data):
9     os.makedirs(directory, exist_ok=True)
10    with open(directory + '/' + name, 'w') as the_file:
11        json.dump(data.tolist(), the_file)

```

Листинг А.7: Модуль mesh.py

```

1 # -*- coding: utf-8 -*-
2 # @Author: Alexander Sharov
3
4 import matplotlib.pyplot as plt
5
6 from triangle import triangulate, plot as tplot
7
8 from fe.triangle import Triangle
9 from geometry.point import Point
10 from data.poly import read_tri, read_pts
11 from data.plot.poly_contour import plot as dplot
12
13 DEFAULT_GRID = 'pqas.001D'
14
15
16 class Mesh(object):
17     """
18     The class Mesh represents a 2D mesh
19     """
20
21     def __init__(self, file):
22         self.raw_file = file
23         self.raw_splitting = None
24         self.splitting = []
25         self.contour = []
26
27     def generate_raw(self, step=DEFAULT_GRID):
28         self.raw_splitting = triangulate(read_tri(self.raw_file), step)
29
30     def generate(self, step=DEFAULT_GRID):
31         self.generate_raw(step)
32
33         for triangle in self.raw_splitting['triangles']:
34             points = []
35             for v in triangle:
36                 point = Point(*self.raw_splitting['vertices'][v])
37                 point.number = v
38                 points.append(point)
39             tri = Triangle(*points)
40             tri.is_boundary = (sum([self.raw_splitting['vertex_markers'][v][0] for v in
41                                     triangle]) >= 1)
42             self.splitting.append(tri)
43
44     def generate_contour(self):
45         for vertex, vertex_matrix in zip(self.raw_splitting['vertices'],
46                                         self.raw_splitting['vertex_markers']):
47             if vertex_matrix == 1:
48                 self.contour.append(vertex)
49
50     def show(self):
51         plt.figure(figsize=(8, 8))
52         ax = plt.subplot(111, aspect='equal')
53         tplot.plot(ax, **self.raw_splitting)
54         plt.show()
55
56     @property
57     def quantity(self):

```

```

56         return len(self.splitting)
57
58     @property
59     def points(self):
60         return self.raw_splitting['vertices']
61
62     def draw_contour(self):
63         plt.figure(figsize=(8, 8))
64         ax = plt.subplot(111, aspect='equal')
65         dplot(ax, **self.raw_splitting)
66         plt.show()

```

Листинг А.8: Модуль poly.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4  from numpy import array
5
6
7  def read_pts(file_name):
8      file = open(file_name, 'r')
9
10     lines = file.readlines()
11     file.close()
12     lines = [x.strip('\n').split() for x in lines]
13
14     vertices = []
15     N_vertices, dimension, _, _ = [int(x) for x in lines[0]]
16     for k in range(N_vertices):
17         label, x, y = [items for items in lines[k + 1]]
18         vertices.append([float(x), float(y)])
19     output = array(vertices)
20
21     return output
22
23
24  def read_tri(file_name):
25      """
26          Simple poly-file reader, that creates a python dictionary
27          with information about vertices, edges and holes.
28          It assumes that vertices have no attributes or boundary markers.
29          It assumes that edges have no boundary markers.
30          No regional attributes or area constraints are parsed.
31      """
32
33      output = {'vertices': None, 'holes': None, 'segments': None}
34
35      # open file and store lines in a list
36      file = open(file_name, 'r')
37
38      lines = file.readlines()
39      file.close()
40      lines = [x.strip('\n').split() for x in lines]
41
42      # store vertices
43      vertices = []
44      N_vertices, dimension, attr, bdry_markers = [int(x) for x in lines[0]]
45      # We assume attr = bdry_markers = 0
46      for k in range(N_vertices):
47          label, x, y = [items for items in lines[k + 1]]
48          vertices.append([float(x), float(y)])
49      output['vertices'] = array(vertices)
50
51      # store segments
52      segments = []
53      N_segments, bdry_markers = [int(x) for x in lines[N_vertices + 1]]
54      for k in range(N_segments):
55          label, pointer_1, pointer_2 = [items for items in lines[N_vertices + k + 2]]
56          segments.append([int(pointer_1) - 1, int(pointer_2) - 1])
57      output['segments'] = array(segments)
58

```

```

59     # Store holes
60     N_holes = int(lines[N_segments + N_vertices + 2][0])
61     holes = []
62     for k in range(N_holes):
63         label, x, y = [items for items in lines[N_segments + N_vertices + 3 + k]]
64         holes.append([float(x), float(y)])
65
66     output['holes'] = array(holes)
67
68     return output

```

Листинг А.9: Модуль mongo.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4  import json
5  import gridfs
6
7  from pymongo import MongoClient
8  from datetime import datetime
9
10
11 def connect(url='mongodb://localhost:27017/', db_name='experiments'):
12     client = MongoClient(url)
13     db = client[db_name]
14     return db
15
16
17 def write(db, collection, directory, mesh_type, mesh_name, time, quantity_of_fe):
18     fs = gridfs.GridFS(db)
19
20     frame_q1_id = fs.put(open('%s/frame/q_1/image.gif' % directory, 'rb'))
21     frame_q2_id = fs.put(open('%s/frame/q_2/image.gif' % directory, 'rb'))
22     frame_H_id = fs.put(open('%s/frame/H/image.gif' % directory, 'rb'))
23
24     surf_q1_id = fs.put(open('%s/surf/q_1/image.gif' % directory, 'rb'))
25     surf_q2_id = fs.put(open('%s/surf/q_2/image.gif' % directory, 'rb'))
26     surf_H_id = fs.put(open('%s/surf/H/image.gif' % directory, 'rb'))
27
28     wave_psi1_id = fs.put(open('%s/wave/psi_1/image.gif' % directory, 'rb'))
29     wave_psi2_id = fs.put(open('%s/wave/psi_2/image.gif' % directory, 'rb'))
30
31     record = {
32         "date": datetime.now().strftime("%Y-%m-%d-%H-%M"),
33         "images": {
34             "frame": {
35                 "fluid_flow": {
36                     "1": frame_q1_id,
37                     "2": frame_q2_id
38                 },
39                 "surface_elevation": frame_H_id
40             },
41             "surface": {
42                 "fluid_flow": {
43                     "1": surf_q1_id,
44                     "2": surf_q2_id
45                 },
46                 "surface_elevation": surf_H_id
47             },
48             "stream_function": {
49                 "1": wave_psi1_id,
50                 "2": wave_psi2_id
51             }
52         },
53         "solution": {
54             "matrix": json.load(open('%s/json/solution.json' % directory)),
55             "times": json.load(open('%s/json/times.json' % directory))
56         },
57         "meta": {
58             "mesh": {
59                 "type": mesh_type,

```

```

60         "name": mesh_name
61     },
62     "execution_time": str(time),
63     "quantity_of_basis_functions": str(quantity_of_fe)
64   }
65 }
66 db[collection].insert(record)
67
68
69 def read(db, collection, date):
70   return db[collection].find_one({"date": date})
71
72
73 def save(db, data, directory):
74   fs = gridfs.GridFS(db)
75
76   with open('%s/solution.json' % directory, 'w') as f:
77     json.dump(data['solution'], f)
78
79   with open('%s/meta.json' % directory, 'w') as f:
80     json.dump(data['meta'], f)
81
82   with open('%s/frame_fluid_flow_1.gif' % directory, 'wb') as f:
83     f.write(fs.get(data['images'][ 'frame'][ 'fluid_flow'][ '1']).read())
84
85   with open('%s/frame_fluid_flow_2.gif' % directory, 'wb') as f:
86     f.write(fs.get(data['images'][ 'frame'][ 'fluid_flow'][ '2']).read())
87
88   with open('%s/frame_surface_elevation.gif' % directory, 'wb') as f:
89     f.write(fs.get(data['images'][ 'frame'][ 'surface_elevation']).read())
90
91   with open('%s/surf_fluid_flow_1.gif' % directory, 'wb') as f:
92     f.write(fs.get(data['images'][ 'surf'][ 'fluid_flow'][ '1']).read())
93
94   with open('%s/surf_fluid_flow_2.gif' % directory, 'wb') as f:
95     f.write(fs.get(data['images'][ 'surf'][ 'fluid_flow'][ '2']).read())
96
97   with open('%s/surf_surface_elevation.gif' % directory, 'wb') as f:
98     f.write(fs.get(data['images'][ 'surf'][ 'surface_elevation']).read())
99
100  with open('%s/stream_function_1.gif' % directory, 'wb') as f:
101    f.write(fs.get(data['images'][ 'stream_function'][ '1']).read())
102
103  with open('%s/stream_function_2.gif' % directory, 'wb') as f:
104    f.write(fs.get(data['images'][ 'stream_function'][ '2']).read())

```

Листинг А.10: Модуль triangle.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4  import numpy as np
5
6  from sympy import symbols, diff, integrate as sp_integrate
7  from sympy.parsing.sympy_parser import parse_expr
8
9  from geometry.point import Point
10
11
12 class Triangle(object):
13   """
14   The class Triangle represents a 2D triangle
15   """
16
17   def __init__(self, *args):
18     self.points = []
19
20     s = len(args)
21     if s is 3:
22       for arg in args:
23         if isinstance(arg, Point):
24           self.points.append(arg)

```

```

25             else:
26                 raise ValueError()
27         else:
28             self.points.append(Point())
29             self.points.append(Point())
30             self.points.append(Point())
31
32     self.basic_functions = self.get_basic_functions()
33
34     def __getitem__(self, index):
35         if index == 1:
36             return self.points[0]
37         elif index == 2:
38             return self.points[1]
39         elif index == 3:
40             return self.points[2]
41
42     def __setitem__(self, index, value):
43         if isinstance(value, Point):
44             if index == 1:
45                 self.points[0] = value
46             elif index == 2:
47                 self.points[1] = value
48             elif index == 3:
49                 self.points[2] = value
50         else:
51             raise TypeError()
52
53     @property
54     def vertexes(self):
55         return self.points
56
57     @staticmethod
58     def centroid(triangle):
59         return Point(((triangle[1].x + triangle[2].x + triangle[3].x) / 3),
60                     ((triangle[1].y + triangle[2].y + triangle[3].y) / 3))
61
62     def centroid(self):
63         return Point(((self[1].x + self[2].x + self[3].x) / 3),
64                     ((self[1].y + self[2].y + self[3].y) / 3))
65
66     def contain(self, point):
67         return ((self[1].x == point.x and self[1].y == point.y) or (self[2].x == point.x and
68                     self[2].y == point.y) or (
69                     self[3].x == point.x and self[3].y == point.y))
70
71     @property
72     def vertices_number(self):
73         return self[1].number, self[2].number, self[3].number
74
75     def get_basic_functions(self):
76         system = [[self[1].x, self[1].y, 1],
77                   [self[2].x, self[2].y, 1],
78                   [self[3].x, self[3].y, 1]]
79
80         x1, x2 = symbols('x_1~x_2')
81
82         f = [1, 0, 0]
83         solution = np.linalg.solve(system, f)
84         basic_i = solution[0] * x1 + solution[1] * x2 + solution[2]
85
86         f = [0, 1, 0]
87         solution = np.linalg.solve(system, f)
88         basic_j = solution[0] * x1 + solution[1] * x2 + solution[2]
89
90         f = [0, 0, 1]
91         solution = np.linalg.solve(system, f)
92         basic_k = solution[0] * x1 + solution[1] * x2 + solution[2]
93
94         return basic_i, basic_j, basic_k
95
96     def get_basic_function_by_number(self, number):
97         if self[1].number == number:

```

```

97         return self.basic_functions[0]
98     elif self[2].number == number:
99         return self.basic_functions[1]
100    elif self[3].number == number:
101        return self.basic_functions[2]
102    else:
103        return 0
104
105    def __str__(self):
106        return '(({g},{g}), ({g},{g}), ({g},{g}))' % (self.points[0].x, self.points[0].y,
107                                                       self.points[1].x, self.points[1].y,
108                                                       self.points[2].x, self.points[2].y)
109
110    def __repr__(self):
111        return 'Triangle(({g},{g}), ({g},{g}), ({g},{g}))' % (self.points[0].x, self.points[0].y,
112                                                               self.points[1].x, self.points[1].y,
113                                                               self.points[2].x, self.points[2].y)
114
115    def area(self):
116        a = np.sqrt((self[2].y - self[3].y) ** 2 + (self[2].x - self[3].x) ** 2)
117        b = np.sqrt((self[1].y - self[3].y) ** 2 + (self[1].x - self[3].x) ** 2)
118        c = np.sqrt((self[1].y - self[2].y) ** 2 + (self[1].x - self[2].x) ** 2)
119        p = (a + b + c) / 2.0
120        return np.sqrt(p * (p - a) * (p - b) * (p - c))
121
122    @staticmethod
123    def random():
124        return Triangle((np.random.uniform(0, 1000), np.random.uniform(0, 1000)),
125                        (np.random.uniform(0, 1000), np.random.uniform(0, 1000)),
126                        (np.random.uniform(0, 1000), np.random.uniform(0, 1000)))
127
128    def integrate_by_jac(self, func):
129        """Deprecated method for integrate over triangle"""
130
131        def d_j(x1, x2):
132            """Jacobian scaling factor"""
133            u, v = symbols('u v')
134
135            det_j = \
136                + diff(x1, u) * diff(x2, v) \
137                - diff(x1, v) * diff(x2, u)
138
139            abs_det_j = parse_expr(str(det_j).replace('-', '+'), evaluate=False)
140
141            return abs_det_j
142
143        x1, y1 = self[1].x, self[1].y
144        x2, y2 = self[2].x, self[2].y
145        x3, y3 = self[3].x, self[3].y
146
147        u, v = symbols('u v')
148
149        # (x_1, x_2) is analogue of (x, y)
150        x_1 = (1 - u) * x1 + u * ((1 - v) * x2 + v * x3)
151        x_2 = (1 - u) * y1 + u * ((1 - v) * y2 + v * y3)
152
153        if func != 0:
154            func = func.subs({symbols('x_1'): x_1,
155                             symbols('x_2'): x_2})
156
157            func *= d_j(x_1, x_2)
158        return sp_integrate(func, (v, 0, 1), (u, 0, 1)).doit()
159
160    def integrate(self, func):
161        """integrate over triangle through barycentric coordinates"""
162
163        x1, y1 = self[1].x, self[1].y
164        x2, y2 = self[2].x, self[2].y
165        x3, y3 = self[3].x, self[3].y
166
167        1, 2 = symbols('\lambda_1 \lambda_2')
168
169        x_1 = 1 * x1 + 2 * x2 + (1 - 1 - 2) * x3

```

```

170     x_2 = 1 * y1 + 2 * y2 + (1 - 1 - 2) * y3
171
172     if func != 0:
173         func = func.subs({symbols('x_1'): x_1,
174                           symbols('x_2'): x_2})
175
176     return 2 * self.area() * sp_integrate(func, (1, 0, 1 - 2), (2, 0, 1)).doit()

```

Листинг А.11: Модуль point.py

```

1  # -*- coding: utf-8 -*-
2  # @Author: Alexander Sharov
3
4
5  class Point(object):
6      """
7          The class Point represents a 2D point
8      """
9
10     @property
11     def x(self):
12         return float(self.points[0])
13
14     @property
15     def y(self):
16         return float(self.points[1])
17
18     @property
19     def number(self):
20         return self.__number
21
22     @number.setter
23     def number(self, value):
24         if value >= 0:
25             self.__number = value
26         else:
27             raise ValueError()
28
29     def set_x(self, value):
30         self.points[0] = value
31
32     def set_y(self, value):
33         self.points[1] = value
34
35     def __init__(self, *args, **kwargs):
36         self.points = []
37         s = len(args)
38         if s is 0:
39             if len(kwargs) > 0:
40                 if ("number" in kwargs) or ("coordinates" in kwargs):
41                     number = kwargs.get("number", 0)
42                     if number is not None:
43                         self.number = int(number)
44                     self.points = kwargs.get("coordinates", ())
45             elif s is 2:
46                 self.points.append(float(args[0]))
47                 self.points.append(float(args[1]))
48             elif s is 3:
49                 self.points.append(float(args[0]))
50                 self.points.append(float(args[1]))
51                 self.number = args[2]
52
53     def __str__():
54         return '(%g, %g)' % (self.x, self.y)
55
56     def __repr__():
57         return 'Point(%g, %g)' % (self.x, self.y)
58
59     def __eq__(self, other):
60         return self.x == other.x and self.y == other.y

```

Листинг А.12: Модуль find.py

```
1 # -*- coding: utf-8 -*-
2 # @Author: Alexander Sharov
3
4 import numpy as np
5
6
7 def maximum(functions, coordinate):
8     if coordinate == 'x':
9         coordinate_number = 0
10    elif coordinate == 'y':
11        coordinate_number = 1
12    elif coordinate == 'z':
13        coordinate_number = 2
14    else:
15        return
16
17    max_val = np.max(functions[0][:, coordinate_number])
18    for func in functions:
19        current = np.max(func[:, coordinate_number])
20        if current > max_val:
21            max_val = current
22    return float(max_val)
23
24
25 def minimum(functions, coordinate):
26     if coordinate == 'x':
27         coordinate_number = 0
28     elif coordinate == 'y':
29         coordinate_number = 1
30     elif coordinate == 'z':
31         coordinate_number = 2
32     else:
33         return
34
35     min_val = np.min(functions[0][:, coordinate_number])
36     for func in functions:
37         current = np.min(func[:, coordinate_number])
38         if current < min_val:
39             min_val = current
40     return float(min_val)
```

Листинг А.13: Модуль gif.py

```
1 # -*- coding: utf-8 -*-
2 # @Author: Alexander Sharov
3
4 import glob
5 import imageio
6
7
8 def build_file(directory, file_list, duration):
9     images = []
10    for filename in file_list:
11        images.append(imageio.imread(filename))
12    output_file = directory + '/image.gif'
13    imageio.mimsave(output_file, images, duration=duration)
14
15
16 def create(directory, duration=1):
17    file_list = glob.glob(directory + '/*.png')
18    build_file(directory, file_list, duration)
```

Листинг А.14: Модуль json2poly.py

```
1 # -*- coding: utf-8 -*-
2 # @Author: Alexander Sharov
3
4 import json
5
6
```

```

7 def json2poly(file):
8     with open(file, 'r', encoding='utf8') as data_file:
9         data = json.load(data_file)
10    points = data['features'][0]['geometry']['coordinates'][0]
11    print(points)
12    count = len(points)
13
14    print('%d_2_0_0' % count)
15
16    for point, point_number in zip(points, range(len(points))):
17        print('%d_%g_%g' % (point_number + 1, point[0], point[1]))
18
19    print('%d_0' % count)
20
21    for ind in range(count):
22        if ind == count:
23            print('%d_%d_0' % (ind + 1, ind + 1))
24        else:
25            print('%d_%d_%d' % (ind + 1, ind + 1, ind + 2))
26
27    print('1')
28    print('1_0_0')
29    print('0')
30
```

TODO: main.py будет ПОТОМ

ПРИЛОЖЕНИЕ Б

Исходный код Docker контейнера

Предполагается, что в этой же папке с файлом для сборки контейнера лежит файл requirements.txt, а также исходный код реализации в папке fem и все необходимые ресурсы в папке resources.

Листинг Б.1: Dockerfile

```
1 FROM python:3
2
3 ENV DISPLAY=:0.0 \
4     MPLBACKEND="agg"
5
6 RUN mkdir -p /opt/diploma
7 WORKDIR /opt/diploma/fem
8
9 COPY ./python/requirements.txt /opt/diploma
10 RUN pip3 install --upgrade pip
11 RUN pip3 install --no-cache-dir -r /opt/diploma/
    requirements.txt
12
13 COPY ./fem /opt/diploma
14 COPY ./resources /opt/diploma/resources
```

Листинг Б.2: requirements.txt

```
1 pymongo
2 numpy
3 sympy
4 scipy
5 matplotlib
6 shapely
7 descartes
8 triangle
9 jupyter
10 ipython
11 bson
12 pillow
13 imageio
```

ПРИЛОЖЕНИЕ В

Выкладки для МКЭ

Для более краткой записи выкладок не будем записывать зависимость весовых функций от (x_1, x_2) , а переменных a_k, a_{m+k}, a_{2m+k} от времени t .

B.1 Уравнение неразрывности

Запишем исходный вид уравнения:

$$\frac{d}{dx_1}q_1 + \frac{d}{dx_2}q_2 + \frac{\partial}{\partial t}(H\rho) = 0 \quad (\text{B.1})$$

Подставим в него разложения q_1, q_2, H :

$$\frac{\partial}{\partial t} \left(\rho \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) \right) + \frac{\partial}{\partial x_1} \sum_{k=1}^M N_k a_k + \frac{\partial}{\partial x_2} \sum_{k=1}^M N_k a_{m+k} = 0$$

Умножим уравнение на $\sum_{l=1}^M W_l$, где W_l – весовая функция, и проинтегрируем по треугольнику Δ :

$$\sum_{l=1}^M \int_{\Delta} \left(\rho \sum_{k=1}^M N_k \frac{d}{dt} a_{2m+k} + \sum_{k=1}^M a_k \frac{\partial}{\partial x_1} N_k + \sum_{k=1}^M a_{m+k} \frac{\partial}{\partial x_2} N_k \right) W_l d\Delta = 0$$

Упростим выражение:

$$\begin{aligned} & \sum_{l=1}^M \int_{\Delta} \rho W_l \frac{d a_{2m+k}}{dt} \sum_{k=1}^M N_k d\Delta = \\ & = \left(- \sum_{l=1}^M \left(\int_{\Delta} W_l a_k \sum_{k=1}^M \frac{\partial}{\partial x_1} N_k d\Delta + \int_{\Delta} W_l a_{m+k} \sum_{k=1}^M \frac{\partial}{\partial x_2} N_k d\Delta \right) \right) \end{aligned}$$

Разрешим его относительно $\frac{da_{2m+k}}{dt}$:

$$\frac{da_{2m+k}}{dt} = \frac{1}{\sum_{l=1}^M \int_{\Delta} \rho W_l \sum_{k=1}^M N_k d\Delta} \cdot \left(- \sum_{l=1}^M \left(\int_{\Delta} W_l a_k \sum_{k=1}^M \frac{\partial}{\partial x_1} N_k d\Delta + \int_{\Delta} W_l a_{m+k} \sum_{k=1}^M \frac{\partial}{\partial x_2} N_k d\Delta \right) \right)$$

B.2 Первое уравнение мелкой воды

Запишем исходный вид уравнения:

$$-Pa \frac{d}{dx_1} H - W^2 \gamma^2 \rho_a + \frac{d}{dt} q_1 - \frac{\partial}{\partial x_1} \left(-\frac{g\rho}{2} H^2 - H P a \right) + \frac{gq_1 \sqrt{q_1^2 + q_2^2}}{H^2 c^2 \rho} = 0$$

Подставим в него разложения q_1, q_2, H :

$$\begin{aligned} & -Pa \frac{\partial}{\partial x_1} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - W^2 \gamma^2 \rho_a - \frac{\partial}{\partial x_1} \left(-Pa \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) \right. \\ & \quad \left. - \frac{g\rho}{2} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2 \right) + \frac{\partial}{\partial t} \sum_{k=1}^M N_k a_k + \\ & \quad + \frac{g \left(\sqrt{\left(\sum_{k=1}^M N_k a_k \right)^2 + \left(\sum_{k=1}^M N_k a_{m+k} \right)^2} \right)}{c^2 \rho \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2} \cdot \sum_{k=1}^M N_k a_k = 0 \end{aligned}$$

Умножим уравнение на $\sum_{l=1}^M W_l$, где W_l - весовая функция, и проинтегрируем по треугольнику Δ :

$$\begin{aligned}
& \sum_{l=1}^M \int_{\Delta} \left(-Pa \frac{\partial}{\partial x_1} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - W^2 \gamma^2 \rho_a - \right. \\
& \left. - \frac{\partial}{\partial x_1} \left(-Pa \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - \frac{g\rho}{2} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2 \right) + \frac{\partial}{\partial t} \sum_{k=1}^M N_k a_k + \right. \\
& \left. + \frac{g \left(\sqrt{\left(\sum_{k=1}^M N_k a_k \right)^2 + \left(\sum_{k=1}^M N_k a_{m+k} \right)^2} \right)}{c^2 \rho \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2} \cdot \sum_{k=1}^M N_k a_k \right) W_1 d\Delta = 0
\end{aligned}$$

Упростим выражение:

$$\begin{aligned}
& \sum_{l=1}^M \int_{\Delta} W_1 \frac{\partial}{\partial t} \left(a_k \sum_{k=1}^M N_k \right) d\Delta + \\
& + \sum_{l=1}^M \int_{\Delta} \left(-W_1 \frac{\partial}{\partial x_1} \left(-Pa H_0 - Pa a_{2m+k} \sum_{k=1}^M N_k - \frac{g\rho}{2} H_0^2 - g\rho H_0 a_{2m+k} \sum_{k=1}^M N_k - \right. \right. \\
& \left. \left. - \frac{g\rho}{2} a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2 \right) \right) d\Delta + \\
& + \sum_{l=1}^M \int_{\Delta} \left(-Pa W_1 \frac{\partial}{\partial x_1} \left(H_0 + a_{2m+k} \sum_{k=1}^M N_k \right) \right) d\Delta + \sum_{l=1}^M \int_{\Delta} \left(-W^2 \gamma^2 \rho_a W_1 \right) d\Delta + \\
& + \sum_{l=1}^M \int_{\Delta} \frac{g \left(\sqrt{a_k^2 \left(\sum_{k=1}^M N_k \right)^2 + a_{m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} \right)}{c^2 \rho H_0^2 + 2c^2 \rho H_0 a_{2m+k} \sum_{k=1}^M N_k + c^2 \rho a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} d\Delta \cdot \\
& \cdot W_1 a_k \sum_{k=1}^M N_k = 0
\end{aligned}$$

Разрешим его относительно $\frac{da_k}{dt}$:

$$\begin{aligned}
\frac{da_k}{dt} = & \frac{1}{\sum_{l=1}^M \int_{\Delta} W_1 \frac{\partial}{\partial t} \left(\sum_{k=1}^M N_k \right) d\Delta} \cdot \\
& \cdot \left(- \sum_{l=1}^M \int_{\Delta} \left(-W_1 \frac{\partial}{\partial x_1} \left(-Pa H_0 - Pa a_{2m+k} \sum_{k=1}^M N_k - \frac{g\rho}{2} H_0^2 - \right. \right. \right. \\
& \quad \left. \left. \left. - g\rho H_0 a_{2m+k} \sum_{k=1}^M N_k - \frac{g\rho}{2} a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2 \right) \right) d\Delta - \\
& - \sum_{l=1}^M \int_{\Delta} \left(-Pa W_1 \frac{\partial}{\partial x_1} \left(H_0 + a_{2m+k} \sum_{k=1}^M N_k \right) \right) d\Delta - \sum_{l=1}^M \int_{\Delta} \left(-W^2 \gamma^2 \rho_a W_1 \right) d\Delta - \\
& - \sum_{l=1}^M \int_{\Delta} \frac{g \left(\sqrt{a_k^2 \left(\sum_{k=1}^M N_k \right)^2 + a_{m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} \right) W_1 a_k \sum_{k=1}^M N_k}{c^2 \rho H_0^2 + 2c^2 \rho H_0 a_{2m+k} \sum_{k=1}^M N_k + c^2 \rho a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} d\Delta
\end{aligned}$$

B.3 Второе уравнение мелкой воды

Запишем исходный вид уравнения:

$$-Pa \frac{d}{dx_2} H + \frac{d}{dt} q_2 - \frac{\partial}{\partial x_2} \left(-\frac{g\rho}{2} H^2 - H P a \right) + \frac{gq_2 \sqrt{q_1^2 + q_2^2}}{H^2 c^2 \rho} = 0$$

Подставим в него разложения q_1, q_2, H :

$$\begin{aligned}
& -Pa \frac{\partial}{\partial x_2} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - \frac{\partial}{\partial x_2} \left(-Pa \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - \right. \\
& \quad \left. - \frac{g\rho}{2} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2 \right) + \frac{\partial}{\partial t} \sum_{k=1}^M N_k a_{m+k} + \\
& \quad + \frac{g \left(\sqrt{\left(\sum_{k=1}^M N_k a_k \right)^2 + \left(\sum_{k=1}^M N_k a_{m+k} \right)^2} \right) \sum_{k=1}^M N_k a_{m+k}}{c^2 \rho \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2} = 0
\end{aligned}$$

Умножим уравнение на $\sum_{l=1}^M W_l$, где W_l - весовая функция, и проинтегрируем по треугольнику Δ :

$$\begin{aligned}
& \sum_{l=1}^M \int_{\Delta} \left(-Pa \frac{\partial}{\partial x_2} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - \right. \\
& \quad \left. - \frac{\partial}{\partial x_2} \left(-Pa \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right) - \frac{g\rho}{2} \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2 \right) + \frac{\partial}{\partial t} \sum_{k=1}^M N_k a_{m+k} + \right. \\
& \quad \left. + \frac{g \left(\sqrt{\left(\sum_{k=1}^M N_k a_k \right)^2 + \left(\sum_{k=1}^M N_k a_{m+k} \right)^2} \right) \sum_{k=1}^M N_k a_{m+k}}{c^2 \rho \left(H_0 + \sum_{k=1}^M N_k a_{2m+k} \right)^2} \right) W_l d\Delta = 0
\end{aligned}$$

Упростим выражение:

$$\begin{aligned}
& \sum_{l=1}^M \int_{\Delta} W_1 \frac{\partial}{\partial t} \left(a_{m+k} \sum_{k=1}^M N_k \right) d\Delta + \\
& + \sum_{l=1}^M \int_{\Delta} \left(-W_1 \frac{\partial}{\partial x_2} \left(-Pa H_0 - Pa a_{2m+k} \sum_{k=1}^M N_k - \frac{g\rho}{2} H_0^2 - g\rho H_0 a_{2m+k} \sum_{k=1}^M N_k - \right. \right. \\
& \left. \left. - \frac{g\rho}{2} a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2 \right) \right) d\Delta + \sum_{l=1}^M \int_{\Delta} \left(-Pa W_1 \frac{\partial}{\partial x_2} \left(H_0 + a_{2m+k} \sum_{k=1}^M N_k \right) \right) d\Delta + \\
& + \sum_{l=1}^M \int_{\Delta} \frac{g \left(\sqrt{a_k^2 \left(\sum_{k=1}^M N_k \right)^2 + a_{m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} \right) W_1 a_{m+k} \sum_{k=1}^M N_k}{c^2 \rho H_0^2 + 2c^2 \rho H_0 a_{2m+k} \sum_{k=1}^M N_k + c^2 \rho a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} d\Delta = 0
\end{aligned}$$

Разрешим его относительно $\frac{da_{m+k}}{dt}$:

$$\begin{aligned}
\frac{da_{m+k}}{dt} &= \frac{1}{\sum_{l=1}^M \int_{\Delta} W_1 \frac{\partial}{\partial t} \left(\sum_{k=1}^M N_k \right) d\Delta} \cdot \\
&\cdot \left(- \sum_{l=1}^M \int_{\Delta} \left(-W_1 \frac{\partial}{\partial x_2} \left(-Pa H_0 - Pa a_{2m+k} \sum_{k=1}^M N_k - \right. \right. \right. \\
&\left. \left. \left. - \frac{g\rho}{2} H_0^2 - g\rho H_0 a_{2m+k} \sum_{k=1}^M N_k - \frac{g\rho}{2} a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2 \right) \right) d\Delta - \\
&- \sum_{l=1}^M \int_{\Delta} \left(-Pa W_1 \frac{\partial}{\partial x_2} \left(H_0 + a_{2m+k} \sum_{k=1}^M N_k \right) \right) d\Delta - \\
&- \sum_{l=1}^M \int_{\Delta} \frac{g \left(\sqrt{a_k^2 \left(\sum_{k=1}^M N_k \right)^2 + a_{m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} \right) W_1 a_{m+k} \sum_{k=1}^M N_k}{c^2 \rho H_0^2 + 2c^2 \rho H_0 a_{2m+k} \sum_{k=1}^M N_k + c^2 \rho a_{2m+k}^2 \left(\sum_{k=1}^M N_k \right)^2} d\Delta \right)
\end{aligned}$$