ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий
Кафедра прикладной математики, дифференциальных уравнений и теоретической механики

УТВЕРЖДАЮ
Зав. кафедрой
канд. физ.-мат. наук, доц.
_____ Р. В. Жалнин
«10» 06 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

ЧИСЛЕННОЕ РЕШЕНИЕ ДВУМЕРНЫХ ЗАДАЧ ГАЗОВОЙ ДИНАМИКИ С ИСПОЛЬЗОВАНИЕМ НЕСТРУКТУРИРОВАННЫХ СЕТОК

Автор магистерской диссертации 04.06.19 Т. Е. Фролов

Обозначение магистерской диссертации МД–02069964–01.04.02–11–19
Направление 01.04.02 Прикладная математика и информатика

Руководитель работы
д-р физ.-мат. наук, проф., чл.-кор. РАН 04.06.19 В. Ф. Тишкин

Нормоконтролер
канд. физ.-мат. наук, доц. 05.06.19 Д. К. Егорова

Рецензент
канд. физ.-мат. наук, доц. 06.06.19 Т. Е. Бадокина

Саранск
2019

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий
Кафедра прикладной математики, дифференциальных уравнений и теоретической механики

УТВЕРЖДАЮ
Зав. кафедрой
канд. физ.-мат. наук, доц.
_____ Р. В. Жалнин
«22» 04 2019 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Фролов Тимур Евгеньевич, 201М группа

1 Тема Численное решение двумерных задач газовой динамики с использованием неструктурированных сеток

Утверждена приказом № 2846-с от 22.04.2019

2 Срок представления работы 06.06.2019

3 Исходные данные для научного исследования (проектирования) Литература по газовой динамике и программированию

4 Содержание выпускной квалификационной работы

- 4.1 Уравнение газовой динамики
- 4.2 Алгоритм решения задачи
- 4.3 Решение задачи

5 Приложения

- 5.1 Приложение А (обязательное) Листинг программы
- 5.2 Приложение Б (обязательное) Диск с магистерской диссертацией

Руководитель работы 23.04.19 В. Ф. Тишкин

Задание принял к исполнению 23.04.19 Т. Е. Фролов

РЕФЕРАТ

Магистерская работа содержит 67 страниц, 34 рисунка, 15 использованных источников, 2 приложения.

ГАЗОВАЯ ДИНАМИКА, МАТЕМАТИЧЕСКЕ МОДЕЛИРОВАНИЕ, РАСЧЕТНАЯ СЕТКА, ПРОФИЛЬ КРЫЛА, NACA0012, VISUAL STUDIO C++, СХЕМА ЛАКСА-ФРИДРИХСА, СХЕМА HLLC, ТЕЧЕНИЕ ГАЗА, GMSH, VTK, PARAVIEW.

Цель работы – разработка и реализация программного обеспечения для исследования аэродинамических характеристик профилей крыльев.

При решении задачи использовалась среда разработки Microsoft Visual Studio 2015, язык программирования C++, сеточный генератор Gmsh и графический пакет ParaView.

Степень внедрения – частичная.

Область применения – учебная.

Эффективность – практическое применение при решении уравнений газовой динамики.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день методы вычислительной математики являются наиболее эффективным средством решения многих прикладных задач. Этому способствует бурное развитие прикладной математики и компьютерных технологий, которые кроме получения теоретического описания процесса позволяют получить визуальную картину и наглядно изучить детали явления. Специалист в области прикладной математики должен владеть основами современных методов и средств математического моделирования.

Задачи вычислительной газовой динамики имеют большую важность и актуальность во многих областях современной науки и техники. Наиболее эффективными методами решения этих задач остаются численные методы. В научно-исследовательской работе будем рассматривать решение двумерной задачи газовой динамики.

Методология математического моделирования широко используется в современной науке. Основная ее идея состоит в том, чтобы заменить исследуемый объект на его «образ» (математическую модель), который в дальнейшем изучается на компьютерах с помощью вычислительно-логических алгоритмов.

Математическое моделирование сочетает в себе преимущества, как теории, так и эксперимента. Работа с моделью объекта, явления или процесса дает возможность без существенных затрат и относительно быстро исследовать его свойства и поведение в любых ситуациях (преимущества теории). В то же время вычислительные эксперименты с моделями объектов позволяют, опираясь на мощь современных вычислительных методов и технических инструментов программирования, подробно и глубоко изучать объекты в достаточной полноте, недоступной чисто теоретическим подходам (преимущества эксперимента) [1].

Большую роль в жизни человека играют газодинамические процессы. В изучении газовой динамики традиционным предметом является ракетостроение

и авиационная техника. Но динамикой газов и жидкостей обусловлены не только полеты летательных аппаратов, работа реактивных двигателей, турбин, двигателей внутреннего сгорания, но и атмосферные явления, дыхание человека и животных, распространение загрязнений в океане и многие другие физические процессы. Перед современной наукой и техникой стоит широкий круг проблем, связанный с решением уравнений газовой динамики.

1 Уравнение газовой динамики

1.1 Двумерная система уравнений газовой динамики

Математическое моделирование – широко распространенный метод для анализа газодинамических явлений, в котором рассматривается некоторый упрощенный, идеальный процесс вместо реального процесса. Этот упрощенный процесс выбирается так, что с одной стороны отражает основные качественные стороны явления и, с другой стороны, допускает достаточно простое математическое описание.

Численное моделирование сложных газодинамических течений, возникающих при обтекании тел различной формы потоками газа под углами атаки с произвольными начальными и граничными условиями для рассматриваемой области, в настоящее время является необходимым этапом при разработке авиационных и космических аппаратов и их элементов. Достоверность полученных результатов для различных характеристик влияет, в конечном итоге, на принимаемые технические решения при проектировании [4].

Для математического моделирования процессов и явлений в механике жидкости и газа используются адаптированные сетки, которые, как правило, являются неструктурированными [6]. Каждая ячейка сетки содержит необходимые данные (температура, давление и т.д.). Чем больше ранг разбиения сетки, тем точнее вычисления.

Математическая модель рассматриваемых в работе процессов представляет собой законы сохранения массы, импульса и энергии. Рассмотрим систему уравнений газовой динамики, описываемой двумерной системой уравнений (идеальный невязкий газ) в переменных Эйлера [2, 3]:

$$\begin{cases} \frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \\ \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2 + p)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} = 0 \\ \frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2 + p)}{\partial y} = 0 \\ \frac{\partial(\rho e)}{\partial t} + \frac{\partial((\rho e + p)u)}{\partial x} + \frac{\partial((\rho e + p)v)}{\partial y} = 0 \\ p = \rho\varepsilon(\gamma - 1) \end{cases}, \quad (1.1.1)$$

где $\rho = \rho(x, y, t)$ – плотность среды,

t – время,

(x, y) – декартовы координаты на плоскости,

$\vec{v} = (u, v)$ – вектор скорости,

$p = p(x, y, t)$ – давление,

$e = \varepsilon + \frac{u^2 + v^2}{2}$ – полная энергия,

ε – внутренняя энергия,

$\gamma = \frac{C_p}{C_v}$ – показатель адиабаты,

$T = \varepsilon(\gamma - 1)$ – температура, выраженная в энергетических единицах,

C_p, C_v – теплоемкость при постоянном давлении и постоянном объеме соответственно. За уравнение состояния принимается уравнение состояния идеального газа с показателем адиабаты γ (формула 1.1.2) [9]:

$$p = \rho\varepsilon(\gamma - 1). \quad (1.1.2)$$

При рассмотрении конкретной модели также необходимо задать начальные (состояние рассматриваемого объекта в начальный момент времени) и граничные (состояние на границе рассматриваемой области) условия, для полного описания решаемой задачи.

Для упрощения дальнейшей записи формул, введем некоторые обозначения:

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{pmatrix}, \quad (1.1.3)$$

$$F_1(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho e + p)u \end{pmatrix}, \quad (1.1.4)$$

$$F_2(U) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho e + p)v \end{pmatrix}, \quad (1.1.5)$$

Тогда формулу (1.1.1) можно представить в следующем виде:

$$\frac{\partial U}{\partial t} + \frac{\partial F_1(U)}{\partial x} + \frac{\partial F_2(U)}{\partial x} = 0. \quad (1.1.6)$$

1.2 Схема Лакса-Фридрихса

Схема для численного решения системы (1.1.6) может быть представлена в виде:

$$\frac{dU_i(t)}{dt} + \frac{1}{\Delta_i} \oint_{\partial \Delta_i} \vec{F} \cdot \vec{n}\, ds = 0. \quad (1.2.1)$$

Здесь $\vec{F} = (F_1, F_2)^T$, $\vec{n}$ – внешняя нормаль к грани ячейки Δ_i.

Интеграл в выражении (1.2.1) вычисляется по формуле прямоугольников:

$$\oint_{\partial \Delta_i} \vec{F} * \vec{n}\, ds \approx \sum_{k=1}^{3} |(\partial \Delta_i)_k| \, F\left(U^+\left(x_c^{(k)}, t\right), U^-\left(x_c^{(k)}, t\right)\right) \cdot n_k, \quad (1.2.2)$$

где $x_c^{(k)}$ – центр k-й стороны i-й ячейки, $F\left(U^+\left(x_c^{(k)},t\right),U^-\left(x_c^{(k)},t\right)\right)\cdot n_k$ – дискретные потоки, которые рассчитываются по схеме Лакса-Фридрихса:

$$F\left(U^+\left(x_c^{(k)},t\right),U^-\left(x_c^{(k)},t\right)\right)\cdot n_k =$$

$$= \frac{1}{2}\left[F\left(U^+(x_c,t)\right)+F\left(U^-(x_c,t)\right)-\alpha\left(U^+(x_c,t)-U^-(x_c,t)\right)\right], \quad (1.2.3)$$

$$\alpha = \max\left\{|\mathrm{v}^-|+\sqrt{\gamma p^-/\rho^-},|\mathrm{v}^+|+\sqrt{\gamma p^+/\rho^+}\right\}. \quad (1.2.4)$$

Здесь $U^-(x_c,t), U^+(x_c,t)$ – значение вектора U в точке x_c с внутренней и внешней стороны от границы ячейки соответственно.

Таким образом, для нахождения газодинамических параметров в ячейке Δ_i на следующем шаге по времени используем формулу:

$$U_i^{n+1} = U_i^n + \frac{\Delta t}{|\Delta_i|}\times$$

$$\times\sum_{k=1}^{3}\frac{|(\partial\Delta_i)_k|}{2}\left[\begin{array}{c}F\left(U^+\left(x_c^{(k)},t\right)\right)+\\ +\ F\left(U^-\left(x_c^{(k)},t\right)\right)-\alpha\left(U^+\left(x_c^{(k)},t\right)-U^-\left(x_c^{(k)},t\right)\right)\end{array}\right]. \quad (1.2.5)$$

Для выполнения условий устойчивости необходимо учитывать ограничение на шаг по времени:

$$\max_i\left\{\max_j\{\alpha_{ij}\}\frac{\Delta t}{|\Delta_i|}\right\}\le 1, \quad (1.2.6)$$

где $\alpha_{ij}=\max\left\{|\mathrm{v}^-|+\sqrt{\gamma p^-/\rho^-},|\mathrm{v}^+|+\sqrt{\gamma p^+/\rho^+}\right\}$ вычисляется для j-го ребра i-й ячейки по формуле 1.2.4.

1.3 HLLC

Схема Лакса – Хатера – Ван-Лира (HLLC) представлена в виде левой, правой и промежуточной ударных волн, разделяющих пространство параметров на четыре области, с рассмотрением контактного разрыва (рисунок 1.3.1) [5].

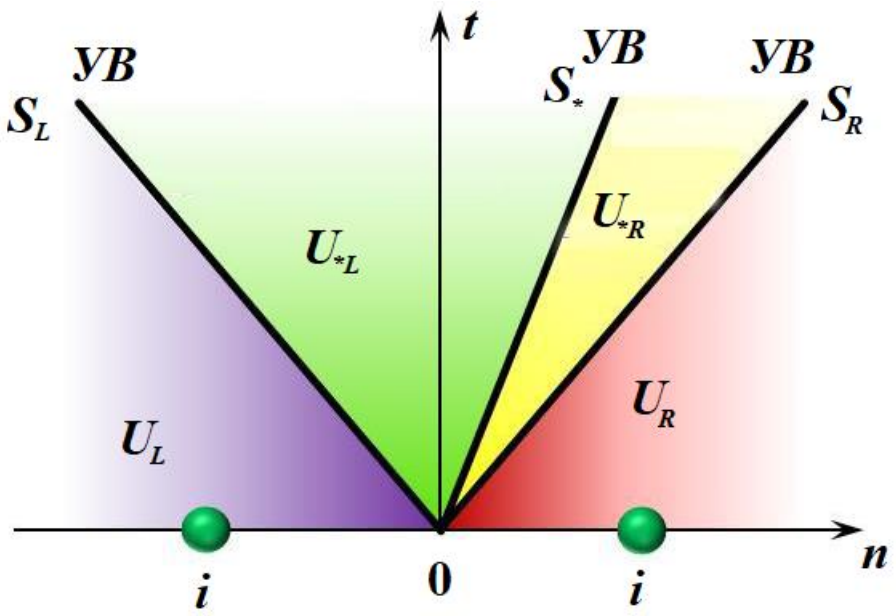

Рисунок 1.3.1

Приближенное решение задачи по схеме HLLC имеет вид (формула 1.3.1) [13]:

$$\overline{U}(x,t) = \begin{cases} U_L, \text{если } \frac{x}{t} \le S_L; \\ U_{*L}, \text{если } S_L \le \frac{x}{t} \le S_*; \\ U_{*R}, \text{если } S_* \le \frac{x}{t} \le S_R; \\ U_R, \text{если } \frac{x}{t} \ge S_R; \end{cases} \quad (1.3.1)$$

где S_L и S_R – скорости распределения ударных волн (левой и правой), которые высчитываются на основе давления в промежуточной области по следующим формулам (1.3.2 – 1.3.3):

$$S_L = v_L - c_L q_L; \tag{1.3.2}$$

$$S_R = v_R + c_R q_R, \tag{1.3.3}$$

где функции q_K для $K = L, R$ принимают вид (формула 1.3.4):

$$q_K = \begin{cases} 1, \text{ для } p_* \le p_K; \\ \sqrt{1 + \frac{\gamma+1}{2\gamma}\left(\frac{p_*}{p_K} - 1\right)}, \text{для } p_* > p_K; \end{cases} \tag{1.3.4}$$

а скорость звука c_K при $K = L, R$ вычисляется по формуле 1.3:5:

$$c_K = \sqrt{\gamma \frac{p_K}{\rho_K}}. \tag{1.3.5}$$

Давление промежуточной области может быть вычислено следующим образом (формула 1.3.6 – формула 1.3.8):

$$p_* = \frac{(p_L + p_R) - (v_R - v_L)\bar{\rho}\,\bar{c}}{2}; \tag{1.3.6}$$

$$\bar{\rho} = \frac{\rho_L + \rho_R}{2}; \tag{1.3.7}$$

$$\bar{c} = \frac{c_L + c_R}{2}. \tag{1.3.8}$$

Формула для оценки скорости волны промежуточной области приведена ниже (формула 1.3.9):

$$S_* = \frac{(p_R - p_L) + \rho_L u_L (S_L - u_L) - \rho_R u_R (S_R - u_R)}{\rho_L (S_L - u_L) - \rho_R (S_R - u_R)}. \quad (1.3.9)$$

Значения соответствующих параметров области решений U_{*K} при $K = L, R$ из системы 1.3.1 определяются из соотношений (формула 1.3.10 – формула 1.3.11):

$$U_{*K} = \rho_K \left(\frac{S_K - u_K}{S_K - S_*}\right) \begin{pmatrix} 1 \\ S_* \\ v_K \\ e_K + (S_* - u_K)\left[S_* + \frac{p_K}{\rho_K (S_K - u_K)}\right] \end{pmatrix}; \quad (1.3.10)$$

$$F_{*K} = \frac{S_* (S_K U_K - F_K) + S_K (p_K + \rho_K (S_K - u_K)(S_* - u_K)) D_*}{S_K - S_*}, \quad (1.3.11)$$

где $D_* = [0, 1, 0, S_*]^T$.

2 Алгоритм решения задачи

2.1 Расчетная сетка

Для численного решения математической модели задачи необходимо сгенерировать расчетную сетку в физической области. Задача построения расчетной сетки заключается в нахождении отображения, которое переводит узлы сетки из физической области в вычислительную. Данное отображение должно быть однозначным и иметь сгущение в тех областях, которые нам наиболее важны для решения задачи. Элементами сетки в одномерном случае являются отрезки, в двумерном – треугольники или четырехугольники, а в трехмерном – тетраэдры или призмы [8].

Расчетные сетки делятся на два основных класса:

1. Структурированные сетки – это сетки, у которых множество сеточных узлов являются упорядоченными (рисунок 2.1.1) Ячейки структурированной расчетной сетки представляют собой прямоугольники (2D) или параллелепипеды (3D).

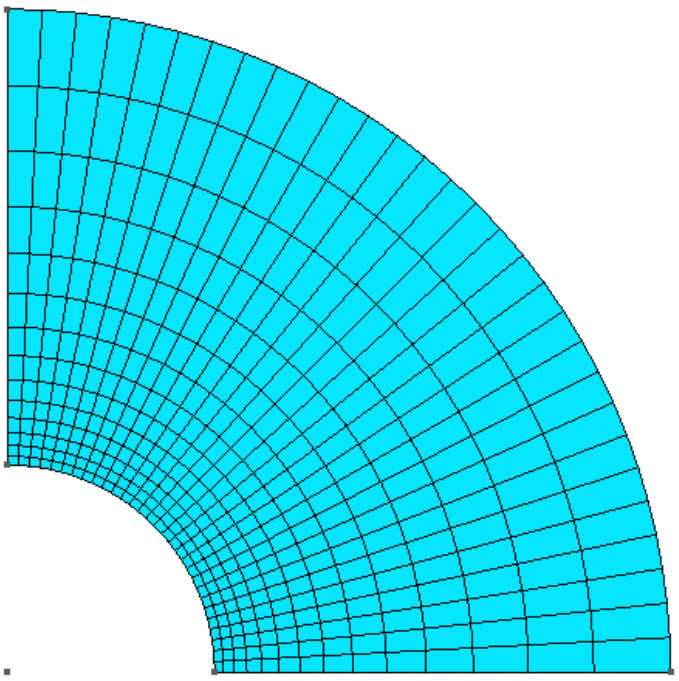

Рисунок 2.1.1

2. Неструктурированные сетки – сетки с произвольным расположением узлов, которые объединяются в многоугольники (на плоскости) или в многогранники (в пространстве) произвольной формы (рисунок 2.1.2). Обычно, на плоскости используют треугольные ячейки, а в трехмерном случае – тетраэдры. Использовать более сложные конструкции неструктурированных сеток чаще всего нет необходимости.

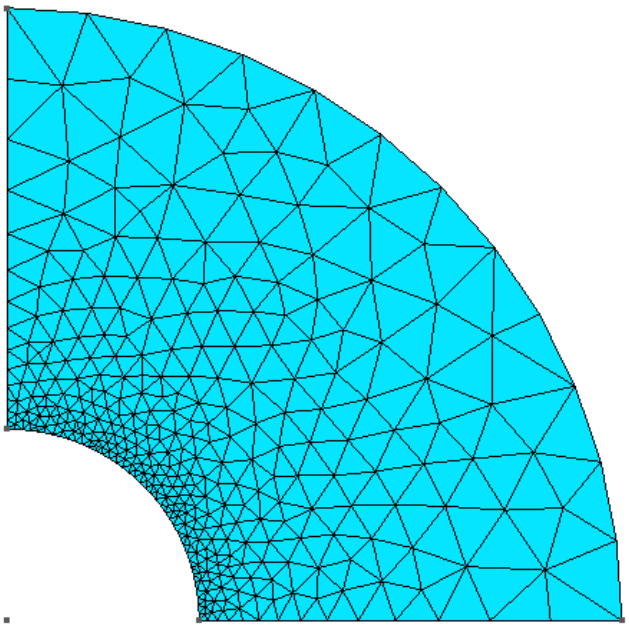

Рисунок 2.1.2

Существует два основных критерия оптимальности расчетной сетки:

1. все ячейки (треугольники) должны быть близки к равносторонним треугольникам, то есть не должно быть слишком много острых углов (критерий оптимизации);
2. площади соседних треугольников не должны сильно различаться (критерий равномерности).

2.1.1 Генератор сетки

Gmsh – бесплатный генератор конечно-элементных сеток. Это быстрый, легкий и удобный инструмент для создания сетки с параметрическим вводом и расширенными возможностями визуализации [10].

Gmsh содержит 4 модуля:

1) геометрия,
2) сетка,
3) средства решений,
4) средства постобработки.

2.1.1.1 Структура входного файла

Генератор сеток *Gmsh* поддерживает несколько вариантов создания геометрии:

1) с использованием графического интерфейса;
2) с помощью текстового редактора, используя язык сценариев *Gmsh* (файлы с расширением *.geo*).

Ниже приведен список основных команд языка сценариев *Gmsh* для создания геометрии.

1) Точки: *Point (тэг) = {x, y, z};*
2) кривые:
    a. линия – *Line (тэг) = {point1, point2};*
    b. сплайн – *Spline (тэг) = {список точек}* – линия, проходящая через все точки в списке;
    c. дуга окружности – *Circle (тэг) = {point1, point_center, point2};*
    d. дуга эллипса – *Ellipse (тэг) = {point1, point_center, point_major_axis, point2};*
    e. петля кривых – *Curve Loop (тэг) = {список кривых};*
3) поверхности:

a. плоская поверхность – *Plane Surface (тэг) = {список петель кривых}*, где первая петля определяет внешнюю границу поверхности, а остальные – отверстия на поверхности;
b. круг – *Disk (тэг) = {center.x, center.y, center.z, radius};*
c. прямоугольник – *Rectangle (тэг) = {point.x, point.y, point.z, width, height}*, где *point* – левый нижний угол;
d. оболочка (петля поверхностей) – *Surface Loop (тэг) = {список поверхностей}*;

4) объемные тела:
a. объемное тело – *Volume (тэг) = {список петель поверхностей}*, где первая петля определяет внешнюю границу объема, а остальные – отверстия в объеме;
b. сфера – *Sphere (тэг) = {center.x, center.y, center.z, radius};*
c. тор – *Torus (тэг) = {center.x, center.y, center.z, radius1, radius2};*
d. конус – *Cone (тэг) = {center.x, center.y, center.z, radius, height}.*

2.1.1.2 Структура файла-результата

Программа Gmsh сохраняет сгенерированную сетку в файл с расширением *.msh*, который имеет следующую структуру:

1) *$MeshFormat* – обязательный раздел, содержащий информацию о файле:
a. *version* – версия msh-сетки;
b. *file-type* – тип файла (0 – ASCII, 1 – бинарный файл);
c. *data-size* – размер данных;
2) *$PhysicalName* – определяет имена физических групп:
a. *numPhysicalNames* – количество физических групп;
b. *dimension* – размерность;

c. *physicalTag* – тэг физической группы;

d. *name* – название физической группы;

3) *$Entities* – хранит информацию о геометрии:

a. *numPoints (size_t)* – количество точек;

b. *numCurves (size_t)* – количество кривых;

c. *numSurfaces (size_t)* – количество плоскостей;

d. *numVolumes (size_t)* – количество объемных тел;

e. *pointTag (int)* – тэг точки;

f. *X (double), Y (double), Z (double)* – координаты точки;

g. *numPhysicalTags (size_t)* – количество тэгов физической группы*;*

h. *physicalTag (int)* – тэг физической группы*;*

i. *curveTag (int)* – тэг кривой*;*

j. *minX (double), minY (double), minZ (double);*

k. *maxX (double), maxY (double), maxZ (double);*

l. *numPhysicalTags (size_t), physicalTag (int);*

4) *$Nodes* – раздел, который содержит список узлов сетки. Узлы разделены на группы:

a. *numEntityBlocks (size_t)* – количество групп;

b. *numNodes (size_t)* – количество всех узлов;

c. *minNodeTag (size_t)* – минимальный тэг узла;

d. *maxNodeTag (size_t)* – максимальный тэг узла;

e. *entityTag (int)* – тэг группы;

f. *numNodesInBlock (size_t);*

g. *nodeTag (size_t)* – тэг узла;

h. *x (double), y (double), z (double)* – координаты узла;

5) *$Elements* – перечень всех ячеек сетки:

a. *numEntityBlocks (size_t)* – количество групп;

b. *numElements (size_t)* количество всех ячеек;

c. *minElementTag (size_t)* – минимальный тэг узла;

d. *maxElementTag (size_t)* – максимальный тэг узла;
e. *entityDim (int)* – размерность группы;
f. *entityTag (int)* – тэг группы;
g. *elementType (int)* – тип ячейки:
    i. 1 – линия;
    ii. 2 – треугольник;
    iii. 3 – четырехугольник;
    iv. 4 – тетраэдр;
    v. 5 – шестигранник;
    vi. 6 – призма;
    vii. 7 – пирамида;
h. *numElementsInBlock (size_t)* – количество ячеек в группе;
i. *elementTag (size_t)* – тэг ячейки сетки;
j. *nodeTag (size_t)* – список тэгов узлов, принадлежащих данной ячейке.

2.1.2 Алгоритм работы с сеткой

2.1.2.1 Чтение сетки

Расчетная сетка состоит из трех основных элементов:

1. узел – базовая структура сетки (рисунок 2.1.2.1.1);

```
struct Point
{
	double x;
	double y;

	Point() : x(0.0), y(0.0) {}
	Point(double ax, double ay) : x(ax), y(ay) {}
	inline void operator  = (double q) { x = q;  y = q; }
	inline void operator  = (Point p) { x = p.x;  y = p.y; }
};
```

Рисунок 2.1.2.1.1 – Листинг 1

2. ячейка (рисунок 2.1.2.1.2);

```
class Cell
{
public:
	Cell(): nodesInd(0), edgesInd(0), nCount(0), eCount(0) {};
	~Cell();

	int   nCount;		// количество узлов ячейки
	int   eCount;		// количество ребер ячейки

	int*  nodesInd;		// список номеров узлов
	int*  edgesInd;		// список номеров ребер
	int   neigh[3];		// список соседних ячеек
	int   type;		// тип ячейки
	double  S;		// площадь ячейки
	Point c;		// центр ячейки
	double HX;		// высота ячейки
	double HY;		// ширина ячейки
};
```

Рисунок 2.1.2.1.2 – Листинг 2

3. ребро – граница между ячейками (рисунок 2.1.2.1.3).

```
class Edge
{
public:
	Edge(): c(0), cCount(0) {};
	~Edge();

	int	n1;		// узел в начале
	int	n2;		// узел в конце

	int	c1;		// ячейка слева
	int	c2;		// ячейка справа, нормаль из c1 в c2
	Vector	n;		// нормаль к грани
	double	l;		// длина грани
	int	cCount;		// количество точек на грани
	Point*	c;		// точки на грани
	int	type;		// тип грани (внутр., гранич.)

public:
	static const int TYPE_INNER	= 0;
	static const int TYPE_OUTLET	= 1;
	static const int TYPE_INLET	= 2;
	static const int TYPE_WALL	= 3;
};
```

Рисунок 2.1.2.1.3 – Листинг 3

На рисунке 2.1.2.1.4 показан код структуры сетки:

```
class Mesh
{
public:
	Mesh(): nodes(0), cells(0), edges(0), nCount(0), cCount(0), eCount(0) {};
	~Mesh();

	Point* nodes;
	Cell*  cells;
	Edge*  edges;
	int    nCount;      //количество узлов (nodes).
	int    cCount;      //количество ячеек (cells).
	int    eCount;      //количество ребер (edges).
};
```

Рисунок 2.1.2.1.4 – Листинг 4

Из файла со сгенерированной сеткой формата *.msh*, нам понадобятся только два раздела: *$Nodes* (узлы сетки), *$Elements* (ячейки сетки). Поэтому все остальные разделы пропускаем.

Код, считывающий по блокам все узлы сетки, представлен на рисунке 2.1.2.1.5:

```
fscanf(fp, "%s", &stmp);                //skip $Nodes

printf("%s\n", stmp);

int countBlocks;                        //numEntityBlocks(size_t)
int countNodesBlock;                    //numNodesBlock(size_t)

fscanf(fp, "%d %d %d %d", &countBlocks, &nCount, &tmp, &tmp);

nodes = new Point[nCount];

int k = 0;                              //индекс для считывания координат

for (int i = 0; i < countBlocks; i++) {
	fscanf(fp, "%d %d %d %d", &tmp, &tmp, &tmp, &countNodesBlock);
	for (int j = 0; j < countNodesBlock; j++)
		fscanf(fp, "%d", &tmp);

	for (int j = 0; j < countNodesBlock; j++) {
		fscanf(fp, "%lf %lf %lf", &(nodes[k].x), &(nodes[k].y), &ftmp);
		k++;
	}
}

fscanf(fp, "%s", &stmp);                //skip $EndNodes
```

Рисунок 2.1.2.1.5 – Листинг 5

Ячейки представлены в трех видах:

1) точки;

2) отрезки – границы области и границы профиля крыла;

3) треугольники – внутренние ячейки сетки.

На рисунке 2.1.2.1.6 показан код, считывающий внутренние ячейки (треугольники), и вычисляющих их некоторые характеристики (площадь ячейки, координаты центра треугольника).

```
fscanf(fp, "%s", &stmp);                    //skip $Elements

int countElements;                          //numElements(size_t)
int countElementsBlock;                     //numElementsBlock(size_t)
int elementType;                            //elementType(int)

fscanf(fp, "%d %d %d %d", &countBlocks, &countElements, &tmp, &tmp);

Cell* cellsTmp = new Cell[countElements];
int cCountTmp = 0;


for (int i = 0; i < countBlocks; i++) {
	fscanf(fp, "%d %d %d %d", &elementType, &tmp, &tmp, &countElementsBlock);
	elementType++;
	if (elementType == 1) {
		for (int j = 0; j < countElementsBlock; j++) {
			fscanf(fp, "%d %d", &tmp, &tmp);
		}
	}
	else if (elementType == 2) {
		for (int j = 0; j < countElementsBlock; j++) {
			fscanf(fp, "%d %d %d", &tmp, &tmp, &tmp);
		}
	}
	else if (elementType == 3) {
		for (int j = 0; j < countElementsBlock; j++) {
			cellsTmp[cCountTmp].nCount = 3;
			cellsTmp[cCountTmp].nodesInd = new int[cellsTmp[cCountTmp].nCount];
				fscanf(fp, "%d %d %d %d", &tmp,
&(cellsTmp[cCountTmp].nodesInd[0]), &(cellsTmp[cCountTmp].nodesInd[1]),
&(cellsTmp[cCountTmp].nodesInd[2]));
			cellsTmp[cCountTmp].nodesInd[0]--;
			cellsTmp[cCountTmp].nodesInd[1]--;
			cellsTmp[cCountTmp].nodesInd[2]--;

			cellsTmp[cCountTmp].c.x = (nodes[cellsTmp[cCountTmp].nodesInd[0]].x +
nodes[cellsTmp[cCountTmp].nodesInd[1]].x + nodes[cellsTmp[cCountTmp].nodesInd[2]].x) / 3.0;
			cellsTmp[cCountTmp].c.y = (nodes[cellsTmp[cCountTmp].nodesInd[0]].y +
nodes[cellsTmp[cCountTmp].nodesInd[1]].y + nodes[cellsTmp[cCountTmp].nodesInd[2]].y) / 3.0;

			cellsTmp[cCountTmp].eCount = 3;
			cellsTmp[cCountTmp].edgesInd = new int[cellsTmp[cCountTmp].eCount];
			for (int k = 0; k < 3; k++)
				cellsTmp[cCountTmp].edgesInd[k] = -1;
			cCountTmp++;
		}
	}
}
```

Рисунок 2.1.2.1.6, лист 1 – Листинг 6

```
cCount = cCountTmp;
cells = new Cell[cCount];
for (int i = 0; i < cCount; i++) {
	cells[i] = cellsTmp[i];
}

for (int i = 0; i < cCount; i++)
{
	double a = sqrt(pow(nodes[cells[i].nodesInd[0]].x - nodes[cells[i].nodesInd[1]].x, 2)
+ pow(nodes[cells[i].nodesInd[0]].y - nodes[cells[i].nodesInd[1]].y, 2));
	double b = sqrt(pow(nodes[cells[i].nodesInd[2]].x - nodes[cells[i].nodesInd[1]].x, 2)
+ pow(nodes[cells[i].nodesInd[2]].y - nodes[cells[i].nodesInd[1]].y, 2));
	double c = sqrt(pow(nodes[cells[i].nodesInd[0]].x - nodes[cells[i].nodesInd[2]].x, 2)
+ pow(nodes[cells[i].nodesInd[0]].y - nodes[cells[i].nodesInd[2]].y, 2));
	double p = (a + b + c) / 2.0;
	cells[i].S = sqrt(p*(p - a)*(p - b)*(p - c));
}
```

Рисунок 2.1.2.1.6, лист 2 – Листинг 6 (продолжение)

2.1.2.2 Создание списка ребер сетки

Ниже приведена процедура *CreateFileEdges(...)*, которая исходя из массива ячеек сетки (треугольников) создает файл, содержащий список всех ребер, причем к каждому ребру соответствует перечень треугольников, к которым принадлежит данное ребро. К сожалению, файл со сгенерированной сеткой формата *.msh* не содержит данные о ребрах, поэтому все данные полученные процедурой *CreateFileEdges(...)* сохраняются в файл *edges.txt*, чтобы не пересчитывать ребра при каждом запуске программы.

```
typedef std::pair<int, int> pointPair;                  // ребро - пара точек на концах ребра

void CreateFileEdges(int cCount, Cell* cells) {
	std::vector <pointPair> edgeTmp;                    // вектор из ребер
	std::map <pointPair, std::vector<int>> m;           // соответствие ребро - треугольники,
включающие это ребро
	int flag_p;                                         // 0 - пара не нашлась в списке, 1 -
пара нашлась, 2 - нашлась симметричная пара
	int p1, p2;
	for (int i = 0; i < cCount; i++)
	{
		for (int j = 0; j < 3; j++) {
			flag_p = 0;

			switch (j) {
			case 0:
				p1 = 0;
				p2 = 1;
				break;
```

Рисунок 2.1.2.2.1, лист 1 – Листинг 7

```
                        case 1:
                                p1 = 1;
                                p2 = 2;
                                break;
                        case 2:
                                p1 = 0;
                                p2 = 2;
                                break;
                        }


                        pointPair p(cells[i].nodesInd[p1], cells[i].nodesInd[p2]);
                        pointPair p_sim(cells[i].nodesInd[p2], cells[i].nodesInd[p1]);

                        for (int k = 0; k < edgeTmp.size(); k++) {
                                pointPair pointTmp = edgeTmp[k];
                                if (flag_p == 0) {
                                        if (p == pointTmp)  flag_p = 1;
                                        else if (p_sim == pointTmp) flag_p = 2;
                                }
                                else break;
                        }

                        switch (flag_p) {
                        case 0:
                                edgeTmp.push_back(p);
                                m[p].push_back(i);
                                break;
                        case 1:
                                m[p].push_back(i);
                                break;
                        case 2:
                                m[p_sim].push_back(i);
                                break;
                        }
                }
        }

        std::map <pointPair, std::vector<int>>::iterator it;
        printf("size: %d \n", m.size());
        printf("\n");

        FILE *file;
        file = fopen("edges.txt", "w");
        if (!file){
                fprintf(stderr, "Can not open file '%s'\n", "edges.txt");
                exit(1);
        }

        fprintf(file, "%d\n", m.size());
        for (it = m.begin(); it != m.end(); it++)
        {
                fprintf(file, "%d\n", (*it).second.size());
                fprintf(file, "%d %d", (*it).first.first, (*it).first.second);
                for (int i = 0; i < (*it).second.size(); i++)
                        fprintf(file, " %d", (*it).second[i]);
                fprintf(file, "\n");
        }
        fclose(file);
}
```

Рисунок 2.1.2.2.1, лист 2 – Листинг 7 (продолжение)

Если ребро является граничным, то есть принадлежит только одному треугольнику, то мы приписываем ему граничные условия. Если это граница области, то условие «входящего потока», а если граница поверхности профиля крыла, то условие «стены» (рисунок 2.1.2.2.2).

```
if ((nodes[n1_tmp].x == -8.0) && (nodes[n2_tmp].x == -8.0)) {
	edges[i].type = Edge::TYPE_INLET;
}
else if ((nodes[n1_tmp].x == 8.0) && (nodes[n2_tmp].x == 8.0)) {
	edges[i].type = Edge::TYPE_INLET;
}
else if ((nodes[n1_tmp].y == 8.0) && (nodes[n2_tmp].y == 8.0)) {
	edges[i].type = Edge::TYPE_INLET;
}
else if ((nodes[n1_tmp].y == -8.0) && (nodes[n2_tmp].y == -8.0)) {
	edges[i].type = Edge::TYPE_INLET;
}
else {
	edges[i].type = Edge::TYPE_WALL;
}
```

Рисунок 2.1.2.2.2 – Листинг 8

Ниже приведен листинг программы (рисунок 2.1.2.2.3), где вычисляются некоторые характеристики ребер (середина ребра, длина ребра, нормаль к грани).

```
for (int iEdge = 0; iEdge < eCount; iEdge++) {
	edges[iEdge].c.x = (nodes[edges[iEdge].n1].x + nodes[edges[iEdge].n2].x) / 2.0;
	edges[iEdge].c.y = (nodes[edges[iEdge].n1].y + nodes[edges[iEdge].n2].y) / 2.0;
	edges[iEdge].n.x = nodes[edges[iEdge].n2].y - nodes[edges[iEdge].n1].y;
	edges[iEdge].n.y = nodes[edges[iEdge].n1].x - nodes[edges[iEdge].n2].x;
	edges[iEdge].l = sqrt(edges[iEdge].n.x*edges[iEdge].n.x +
edges[iEdge].n.y*edges[iEdge].n.y);
	edges[iEdge].n.x /= edges[iEdge].l;
	edges[iEdge].n.y /= edges[iEdge].l;
}
```

Рисунок 2.1.2.2.3 – Листинг 9

2.2 Вычислительный алгоритм

Класс *MethodGas* реализует численные схемы, описанные в первой главе (рисунок 2.2.1)

```
class MethodGas : public Method
{
public:
	MethodGas():GAM(1.4){};
	virtual void convertToParam(int, Param&);

	virtual void init();
	virtual void run();

	~MethodGas();
protected:

	void initValues();

	void flux_lf(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv, double
&fe);

	void flux_hllc(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv,
double &fe);

	void bnd(Edge *e, Param p1, Param &p2);
	void calcTau();
	void _flux_hllc_x_1(Param prim[2], double* qr, double* qu, double* qv, double* qe);

	double *ro, *ru, *rv, *re, *tau;
	double *int_ro, *int_ru, *int_rv, *int_re, *v_max;

	double TMAX;
	double TAU;
	double CFL;

	double GAM;
};
```

Рисунок 2.2.1 – Листинг 10

Опишем переменные и методы класса:

- **ro*, **ru*, **rv*, **re* – массивы консервативных переменных $\rho, \rho u, \rho v$ и ρe соответственно (на текущий момент времени расчета);
- **int_ro*, ** int_ru*, ** int_rv*, ** int_re* – массивы криволинейных интегралов (формула 1.2.2);
- *TMAX* – значение времени окончания расчета;
- *TAU* – шаг по времени;
- *GAM* – адиабата γ;
- *initValues()* – вычисление значений консервативных переменных в начальный момент времени (рисунок 2.2.2);

```
void MethodGas::initValues()
{
	for (int i = 0; i < mesh->cCount; i++) {
		ro[i] = 0.647432;
		ru[i] = 220.8633;
		rv[i] = 5.741077;
		re[i] = (46066.16) / (GAM - 1.0) + 0.5*(ru[i] * ru[i] + rv[i] * rv[i]) /
ro[i];
	}
}
```

Рисунок 2.2.2 – Листинг 11

– *flux_lf(...)* – вычисление дискретных потоков на ребрах сетки по схеме Лакса-Фридрихса (рисунок 2.2.3);

```
void MethodGas::flux_lf(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv,
double &fe)
{
	double rol, rul, rvl, rel;
	double ror, rur, rvr, rer;
	double frl, ful, fvl, fel;
	double frr, fur, fvr, fer;
	double alpha, unl, unr, q1,q2;

	unl = n.x*pl.u+n.y*pl.v;
	unr = n.x*pr.u+n.y*pr.v;

	rol = pl.r;
	rul = pl.r*pl.u;
	rvl = pl.r*pl.v;
	rel = pl.r*(pl.e+0.5*pl.magU());

	ror = pr.r;
	rur = pr.r*pr.u;
	rvr = pr.r*pr.v;
	rer = pr.r*(pr.e+0.5*pr.magU());

	frl = pl.r*unl;
	ful = frl*pl.u+pl.p*n.x;
	fvl = frl*pl.v+pl.p*n.y;
	fel = (rel+pl.p)*unl;

	frr = pr.r*unr;
	fur = frr*pr.u+pr.p*n.x;
	fvr = frr*pr.v+pr.p*n.y;
	fer = (rer+pr.p)*unr;

	q1 = sqrt(pl.p*GAM/pl.r)+fabs(pl.magU());
	q2 = sqrt(pr.p*GAM/pr.r)+fabs(pr.magU());
	alpha = (q1 > q2) ? q1 : q2;

	fr = 0.5*((frl+frr)-alpha*(ror-rol));
	fu = 0.5*((ful+fur)-alpha*(rur-rul));
	fv = 0.5*((fvl+fvr)-alpha*(rvr-rvl));
	fe = 0.5*((fel+fer)-alpha*(rer-rel));
}
```

Рисунок 2.2.3 – Листинг 12

– *flux_hllc(...)* – вычисление дискретных потоков на ребрах сетки по схеме *HLLC* (рисунок 2.2.4);

```
void MethodGas::flux_hllc(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv,
double &fe)
{
	Param pn[2];
	int i;
	double _qu, _qv;

	pn[0] = pl; pn[1] = pr;
	pn[0].u =  n.x * pl.u + n.y * pl.v;
	pn[0].v = -n.y * pl.u + n.x * pl.v;
	pn[1].u =  n.x * pr.u + n.y * pr.v;
	pn[1].v = -n.y * pr.u + n.x * pr.v;
	_flux_hllc_x_1(pn, &fr, &_qu, &_qv, &fe); // n = (1,0)
	fu = _qu*n.x-_qv*n.y;
	fv = _qu*n.y+_qv*n.x;
}
#define F_HLLC_U(UK, FK, SK, SS, PK, RK, VK) (((SS)*((SK)*(UK)-(FK)) + (SK)*(
(PK)+(RK)*((SK)-(VK))*((SS)-(VK)) )) / ((SK)-(SS)))
#define F_HLLC_V(UK, FK, SK, SS, PK, RK, VK) (((SS)*((SK)*(UK)-(FK))) / ((SK)-(SS)))
#define F_HLLC_E(UK, FK, SK, SS, PK, RK, VK) (((SS)*((SK)*(UK)-(FK)) + (SK)*(
(PK)+(RK)*((SK)-(VK))*((SS)-(VK)) )*(SS)) / ((SK)-(SS)))

void MethodGas::_flux_hllc_x_1(Param prim[2], double* fr, double* fu, double* fv, double*
fe)
{
	int		i;
	double		sl, sr, p_star, s_star, p_pvrs, ql, qr, tmp, cz[2], E[2];
	cz[0] = sqrt(GAM*prim[0].p/prim[0].r);
	cz[1] = sqrt(GAM*prim[1].p/prim[1].r);
	E[0] = prim[0].e+0.5*prim[0].magU();
	E[1] = prim[1].e+0.5*prim[1].magU();

	p_pvrs = 0.5*(prim[0].p+prim[1].p)-0.5*(prim[1].u-
prim[0].u)*0.25*(prim[0].r+prim[1].r)*(cz[0]+cz[1]);
	p_star = (p_pvrs > 0.) ? p_pvrs : 0.;
	ql = (p_star <= prim[0].p) ? 1 : sqrt(1.+(GAM+1.)*(p_star/prim[0].p-1.)/(2.*GAM));
	qr = (p_star <= prim[1].p) ? 1 : sqrt(1.+(GAM+1.)*(p_star/prim[1].p-1.)/(2.*GAM));

	sl = prim[0].u-cz[0]*ql;
	sr = prim[1].u+cz[1]*qr;
	if (sl>sr) {
		tmp = sl;
		sl = sr;
		sr = tmp;
	}
	s_star  = prim[1].p-prim[0].p;
	s_star += prim[0].r*prim[0].u*(sl-prim[0].u);
	s_star -= prim[1].r*prim[1].u*(sr-prim[1].u);
	s_star /= (prim[0].r*(sl-prim[0].u)-prim[1].r*(sr-prim[1].u));

	if (s_star < sl) s_star = sl;
	if (s_star > sr) s_star = sr;
	if (!((sl <= s_star) && (s_star <= sr))) {
		printf("HLLC solver: ERROR!\n");
		exit(1);
	}
```

Рисунок 2.2.4, лист 1 – Листинг 13

```
	if (sl >= 0.) {
		*fr = prim[0].r * prim[0].u;
		*fu = prim[0].r * prim[0].u * prim[0].u + prim[0].p;
		*fv = prim[0].r * prim[0].v * prim[0].u;
		*fe = (prim[0].r * E[0] + prim[0].p) * prim[0].u;
	}
	else if (sr <= 0.) {
		*fr = prim[1].r * prim[1].u;
		*fu = prim[1].r * prim[1].u * prim[1].u + prim[1].p;
		*fv = prim[1].r * prim[1].v * prim[1].u;
		*fe = (prim[1].r * E[1] + prim[1].p) * prim[1].u;
	}
	else {
		if (s_star >= 0) {
			*fr = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[0].r,
				prim[0].r * prim[0].u,
				sl, s_star, prim[0].p, prim[0].r, prim[0].u
			);
			*fu = F_HLLC_U( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[0].r * prim[0].u,
				prim[0].r * prim[0].u * prim[0].u + prim[0].p,
				sl, s_star, prim[0].p, prim[0].r, prim[0].u
			);
			*fv = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[0].r * prim[0].v,
				prim[0].r * prim[0].u * prim[0].v,
				sl, s_star, prim[0].p, prim[0].r, prim[0].u
			);
			*fe = F_HLLC_E( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[0].r * E[0],
				(prim[0].r * E[0] + prim[0].p)*prim[0].u,
				sl, s_star, prim[0].p, prim[0].r, prim[0].u
			);
		}
		else {
			*fr = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[1].r,
				prim[1].r * prim[1].u,
				sr, s_star, prim[1].p, prim[1].r, prim[1].u
			);
			*fu = F_HLLC_U( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[1].r * prim[1].u,
				prim[1].r * prim[1].u * prim[1].u + prim[1].p,
				sr, s_star, prim[1].p, prim[1].r, prim[1].u
			);
			*fv = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[1].r * prim[1].v,
				prim[1].r * prim[1].u * prim[1].v,
				sr, s_star, prim[1].p, prim[1].r, prim[1].u
			);
			*fe = F_HLLC_E( /*  UK, FK, SK, SS, PK, RK, VK */
				prim[1].r * E[1],
				(prim[1].r * E[1] + prim[1].p)*prim[1].u,
				sr, s_star, prim[1].p, prim[1].r, prim[1].u
			);
		}
	}
}
```

Рисунок 2.2.4, лист 2 – Листинг 13 (продолжение)

– *bnd(...)* – вычисление значений согласно граничным условиям (рисунок 2.2.5);

```
void MethodGas::bnd(Edge *e, Param p1, Param &p2)
{
	switch (e->type) {
		case 1: // вытекание
			p2 = p1;
			break;
		case 2: // втекание
			p2.r = 0.647432;
			p2.u = 220.8633;
			p2.v = 5.741077;
			p2.p = 46066.16;
			p2.e = p2.p / p2.r / (GAM - 1.0);
			p2.T = 248.0;
			p2.M = 0.7;
			break;
		case 3: // отражение
			p2 = p1;
			double Un = p1.u*e->n.x + p1.v*e->n.y;
			Vector V;
			V.x = e->n.x*Un*2.0;
			V.y = e->n.y*Un*2.0;
			p2.u = p1.u - V.x;
			p2.v = p1.v - V.y;
			break;
	}
}
```

Рисунок 2.2.5 – Листинг 14

– *init()* – процедура инициализации, в которой выделяется память для массивов, которые с помощью метода *initValues()* (реализация на рисунке 2.2.2) заполняются начальными данными, а затем происходит сохранение данных на начальный момент времени (рисунок 2.2.6);

```
void MethodGas::init()
{
	mesh = new Mesh();
	mesh->initFromFile((char*)"naca0012");

	ro = new double[mesh->cCount];
	ru = new double[mesh->cCount];
	rv = new double[mesh->cCount];
	re = new double[mesh->cCount];

	initValues();
	saveVTK(0);
```

Рисунок 2.2.6, лист 1 – Листинг 15

```
	int_ro = new double[mesh->cCount];
	int_ru = new double[mesh->cCount];
	int_rv = new double[mesh->cCount];
	int_re = new double[mesh->cCount];

	TMAX = 10.0;

	TAU  = 1.e-5;
}
```

Рисунок 2.2.6, лист 2 – Листинг 15 (продолжение)

– *Param* – структура для представления физических параметров в каждой ячейки сетки. В методе *convertToParam(…)*, учитывая уравнение состояния идеального газа (формула 1.1.2), заполняется структура *Param* значениями из массивов (рисунок 2.2.7);

```
struct Param
{
	double r;	//плотность
	double p;	//давление

	double u;	//первая компонента вектора скорости
	double v;	//вторая компонента вектора скорости

	double e;	//внутренняя энергия
	double T;	//температура
	double M;	//число Маха

	inline double magU() { return u*u+v*v; };
};


void MethodGas::convertToParam(int i, Param& p)
{
	p.r = ro[i];
	p.u = ru[i]/p.r;
	p.v = rv[i]/p.r;
	p.e = re[i] / p.r - p.magU() / 2.;
	p.p = p.r*p.e*(GAM-1.0);
	p.T = 0.0;
	p.M = p.magU()/sqrt(GAM*p.p/p.r);
}
```

Рисунок 2.2.7 – Листинг 16

– *run()* – основной вычислительный метод (рисунок 2.2.8). Он состоит из двух вложенных циклов (цикл по времени и цикл по ребрам сетки).

```
void MethodGas::run()
{
	double t = 0.0;
	int step = 0;
	int k = 1;
	while (t < TMAX) {
		t += TAU;
		step++;

		memset(int_ro, 0, sizeof(double)*mesh->cCount);
		memset(int_ru, 0, sizeof(double)*mesh->cCount);
		memset(int_rv, 0, sizeof(double)*mesh->cCount);
		memset(int_re, 0, sizeof(double)*mesh->cCount);
		for (int ie = 0; ie < mesh->eCount; ie++) {
			Edge &e = mesh->edges[ie];
			int c1 = e.c1;
			Cell &cell1 = mesh->getCell(c1);

			Param p1, p2;
			convertToParam(c1, p1);
			if (e.c2 >= 0) {
				convertToParam(e.c2, p2);
			}
			else {
				bnd(&e, p1, p2);
			}

			double fr, fu, fv, fe;
			flux(p1, p2, e.n, fr, fu, fv, fe);

			fr *= e.l;
			fu *= e.l;
			fv *= e.l;
			fe *= e.l;
			int_ro[c1] -= fr;
			int_ru[c1] -= fu;
			int_rv[c1] -= fv;
			int_re[c1] -= fe;
			if (e.c2 >= 0) {
				int_ro[e.c2] += fr;
				int_ru[e.c2] += fu;
				int_rv[e.c2] += fv;
				int_re[e.c2] += fe;
			}
		}

		for (int i = 0; i < mesh->cCount; i++) {
			double CFL = TAU/mesh->cells[i].S;
			ro[i] += int_ro[i]*CFL;
			ru[i] += int_ru[i]*CFL;
			rv[i] += int_rv[i]*CFL;
			re[i] += int_re[i]*CFL;
		}

		if (step % 1000 == 0)
		{
			saveVTK(step);
			printf("Calculation results for step %d are saved.\n", step);
		}
	}
}
```

Рисунок 2.2.8 – Листинг 17

2.3 Формирование результата и его визуализация

2.3.1 ParaView. Формат VTK

Для хранения, вывода, визуализации и обработки результатов расчетов будем использовать открытый графический кроссплатформенный пакет *ParaView*, имеющий собственный формат файлов *.vtk* [14].

Файлы *VTK* состоят из пяти основных частей.

1. Первая часть – это версия файла и идентификатор. Эта часть содержит одну строку: *# vtk DataFile Version x.x*, где *x.x* – номер версии.
2. Вторая часть – это заголовок, который состоит из символьной строки, завершенной символом конца строки *\n*. Заголовок имеет максимум 256 символов. Он используется для описания данных.
3. Третья часть – это формат файла, описывающий тип файла (двоичный или ASCII).
4. Следующая часть является структурой, которая описывает геометрию набора данных. Она начинается с ключевого слова *DATASET*, за которым указывают тип набора данных, в зависимости от которого другие комбинации ключевых слов определяют фактические данные.
5. Заключительная часть содержит атрибуты набора данных. Данная часть начинается с ключевых слов *POINT_DATA* или *CELL_DATA*, после которых расположено целое число, определяющее количество точек или ячеек, соответственно. Другие комбинации ключевых слов описывают атрибут набора данных (скаляры, векторы, нормали, координаты текстуры и т.д.).

Для определения неструктурированной сетки достаточно точек (*POINTS*), ячеек (*CELLS*) и тип ячеек (*CELLS_TYPES*). Ключевое слово *CELLS* требует два параметра: количество ячеек *n* и размер списка ячеек. А для ключевое слово

CELL_TYPES необходимо указать всего один параметр – количество ячеек *n*. Это значение должно соответствовать значению, указанному ключевым словом *CELLS*. Данные типов ячеек представляют собой одно целое значение для каждой ячейки, в которой указан тип ячейки.

Некоторые возможные типы ячеек:

1. *VTK_VERTEX* (1) – точка;
2. *VTK_LINE* (3) – отрезок;
3. *VTK_TRIANGLE* (5) – треугольник;
4. *VTK_QUAD* (9) – четырехугольник;
5. *VTK_HEXAHEDRON* (12) – шестигранник.

2.3.2 Запись результата

На рисунке 2.3.2.1 приведен код метода *saveVTK(int step)*, который реализует вывод в файл результатов вычислений на шаге *step*.

```
void Method::saveVTK(int step) {
	char fName[50];

	sprintf(fName, "res_%010d.vtk", step);
	FILE * fp = fopen(fName, "w");
	if (!fp) {
		fprintf(stderr, "Can not open file '%s'\n", fName);
		exit(1);
	}
	fprintf(fp, "# vtk DataFile Version 2.0\n");
	fprintf(fp, "GASDIN data file\n");
	fprintf(fp, "ASCII\n");
	fprintf(fp, "DATASET UNSTRUCTURED_GRID\n");
	fprintf(fp, "POINTS %d float\n", mesh->nCount);
	for (int i = 0; i < mesh->nCount; i++)
	{
		fprintf(fp, "%f %f %f  ", mesh->nodes[i].x,  mesh->nodes[i].y, 0.0);
		if (i+1 % 8 == 0) fprintf(fp, "\n");
	}
	fprintf(fp, "\n");
	fprintf(fp, "CELLS %d %d\n", mesh->cCount, 4*mesh->cCount);
	for (int i = 0; i < mesh->cCount; i++)
	{
		fprintf(fp, "3 %d %d %d\n", mesh->cells[i].nodesInd[0], mesh-
>cells[i].nodesInd[1], mesh->cells[i].nodesInd[2]);
	}
	fprintf(fp, "\n");
```

Рисунок 2.3.2.1, лист 1 – Листинг 18

```
	fprintf(fp, "CELL_TYPES %d\n", mesh->cCount);
	for (int i = 0; i < mesh->cCount; i++) {
		fprintf(fp, "5\n");
	}
	fprintf(fp, "\n");

	fprintf(fp, "CELL_DATA %d\nSCALARS Density float 1\nLOOKUP_TABLE default\n", mesh-
>cCount);
	for (int i = 0; i < mesh->cCount; i++)
	{
		Param p;
		convertToParam(i, p);
		fprintf(fp, "%25.16f ", p.r);
		if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
	}

	fprintf(fp, "SCALARS Pressure float 1\nLOOKUP_TABLE default\n");
	for (int i = 0; i < mesh->cCount; i++)
	{
		Param p;
		convertToParam(i, p);
		fprintf(fp, "%25.16f ", p.p);
		if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
	}

	fprintf(fp, "SCALARS Temperature float 1\nLOOKUP_TABLE default\n");
	for (int i = 0; i < mesh->cCount; i++)
	{
		Param p;
		convertToParam(i, p);
		fprintf(fp, "%25.16f ", p.T);
		if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
	}

	fprintf(fp, "VECTORS Velosity float\n");
	for (int i = 0; i < mesh->cCount; i++)
	{
		Param p;
		convertToParam(i, p);
		fprintf(fp, "%25.16f %25.16f %25.16f ", p.u, p.v, 0.0);
		if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
	}

	fclose(fp);
}
```

Рисунок 2.3.2.1, лист 2 – Листинг 18 (продолжение)

3 Решение задачи

3.1 Постановка задачи

Рассчитать 2D течение невязкого сжимаемого газа в окрестности классического симметричного аэродинамического профиля *NACA0012* (рисунок 3.1.1) и определить его основные аэродинамические характеристики, используя схему Лакса-Фридрикса и схему *HLLC*. Построить зависимости аэродинамических характеристик при разных углах атаки и сравнить с экспериментальными данными.

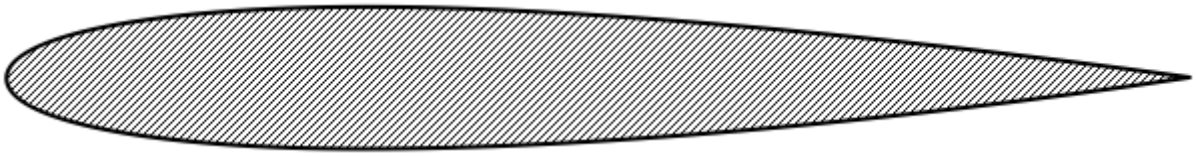

Рисунок 3.1.1

Геометрия профиля задается формулой 3.1.1 [15]:

$$y = \frac{t}{0.2} c \left[\begin{array}{c} 0.2969 \sqrt{\frac{x}{c}} - 0.1260 \left(\frac{x}{c}\right) - \\ - 0.3516 \left(\frac{x}{c}\right)^2 + 0.2843 \left(\frac{x}{c}\right)^3 - \ 0.1036 \left(\frac{x}{c}\right)^4 \end{array} \right], \qquad (3.1.1)$$

где c – длина хорды,

x – положение вдоль хорды от 0 до c,

y – половина толщины при заданном значении x,

t – максимальная толщина профиля.

Для профиля *NACA0012*: $t = 0.012$ и в этой задаче длина хорды $c = 1$м (рисунок 3.1.2).

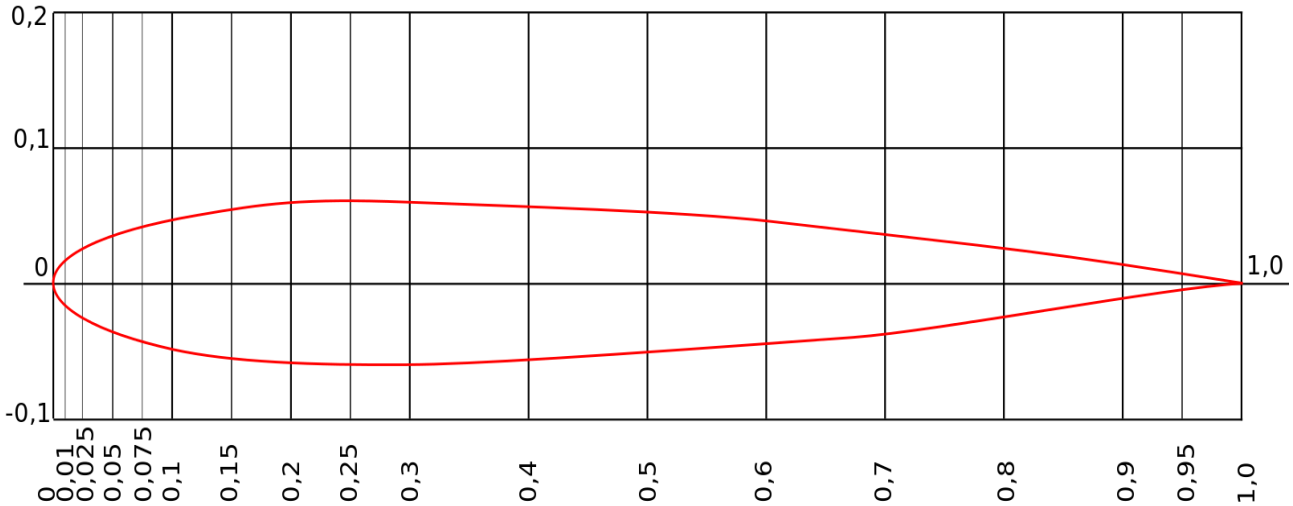

Рисунок 3.1.2

Задача обтекания невязким газом профиля *NACA0012* решается в прямоугольной расчетной области, границы которой располагаются довольно далеко от профиля, с использованием расчетной сетки (в данном случае двумерной неструктурированной треугольной сетки).

Начальные данные:

1. плотность – $\rho = 0.02898$ кг/м^3;
2. давление – $p = 46066.16$ Па;
3. температура – $T = 248$ К;
4. число Маха – $M = 0.7$;
5. показатель адиабаты $\gamma = 1.4$.

Список углов атаки α, и, соответственно, компонент скорости (первая – u, вторая – v), при которых необходимо произвести вычисления:

1. $\alpha = 1.489°$;
    a. $u = 220.8633$ м/с;
    b. $v = 5.741077$ м/с;
2. $\alpha = 3.0462°$;
    a. $u = 220.7004$ м/с;
    b. $v = 11.74487$ м/с;

3. $\alpha = 3.9897°$;
    a. $u = 220.4771$ м/с;
    b. $v = 15.37743$ м/с;
4. $\alpha = 4.841°$;
    a. $u = 220.2243$ м/с;
    b. $v = 18.65146$ м/с.

Необходимо построить график распределения коэффициента давления по поверхности профиля, используя формулу 3.1.2:

$$C_p = \frac{p}{(\rho_\infty u_\infty^2)/2}, \quad (3.1.2)$$

где ρ_∞ и u_∞ – параметры набегающего потока.

Также необходимо вычислить коэффициент подъемной силы C_y и построить график зависимости C_y от угла атаки, который вычисляется по следующей формуле (3.1.3):

$$C_y = \frac{Y}{(\rho_\infty u_\infty^2) l/2}, \quad (3.1.3)$$

где Y –подъемная сила, l – периметр профиля крыла. Подъемная сила считается по формуле 3.1.4:

$$Y = \sum p_i n_{y_i} l_i, \quad (3.1.4)$$

где p_i – давление в i-й ячейки расчетной сетки профиля крыла, n_{y_i} – проекция нормали i-й ячейки на соответствующую ось координат, l_i – длина ячейки.

3.2 Генерация сетки

С помощью генератора сеток *Gmsh* была построена расчетная сетка, которая является двумерной неструктурированной треугольной сеткой, состоящей из более 12000 ячеек и сильно сгущающейся около поверхности профиля крыла (рисунок 3.2.1 – рисунок 3.2.2).

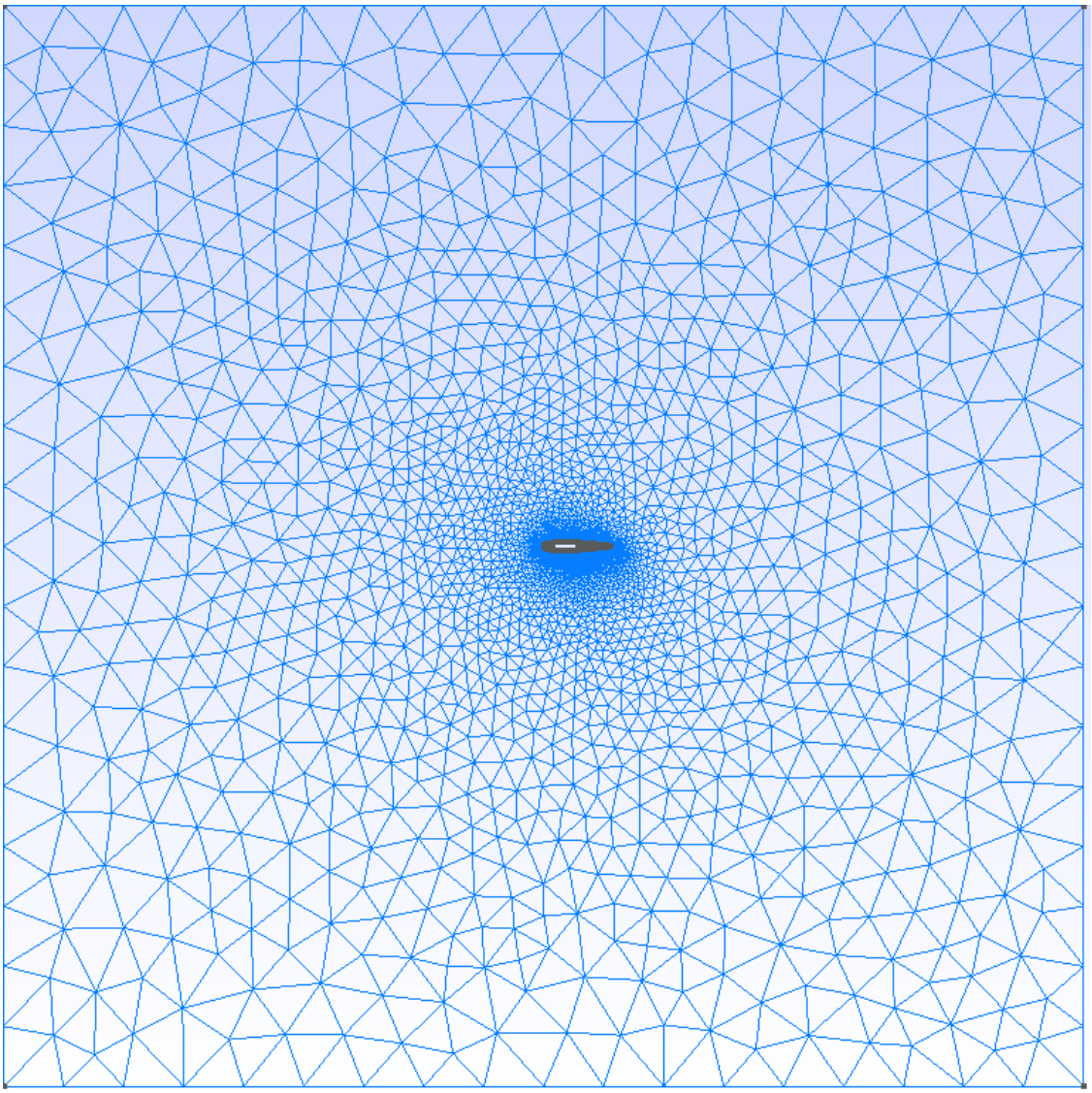

Рисунок 3.2.1

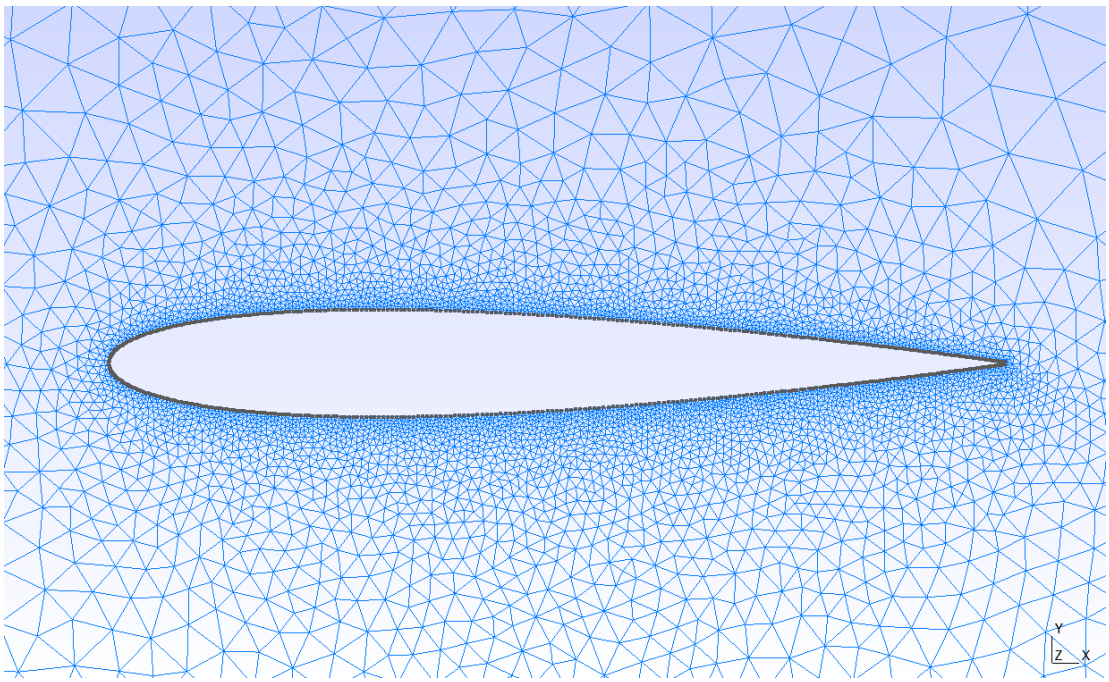

Рисунок 3.2.2

3.3 Результаты вычислений

В результате выполнения программы записываются файлы в формате *.vtk* через определенное количество шагов по времени. Для их просмотра необходимо воспользоваться программой *ParaView* [12]. На рисунке 3.3.1 представлена визуализация результата при $\alpha = 1.489$.

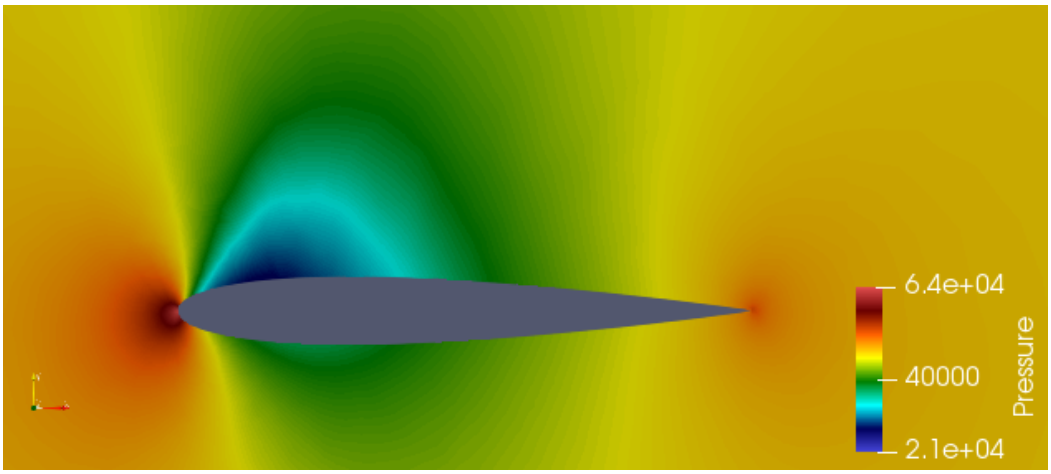

Рисунок 3.3.1

Следующий шаг – построение графика распределения коэффициента давления по поверхности профиля с помощью встроенного калькулятора программы *ParaView*, используя формулу 3.1.2. На рисунке 3.3.2 изображен график распределения коэффициента давления по результатам полученным с помощью схемы Лакса-Фридрихса, а на рисунке 3.3.3 – с помощью схемы *HLLC*.

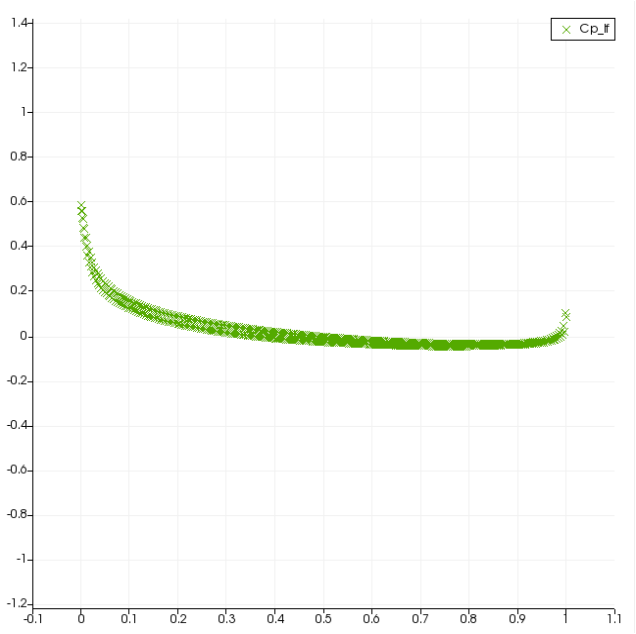

Рисунок 3.3.2

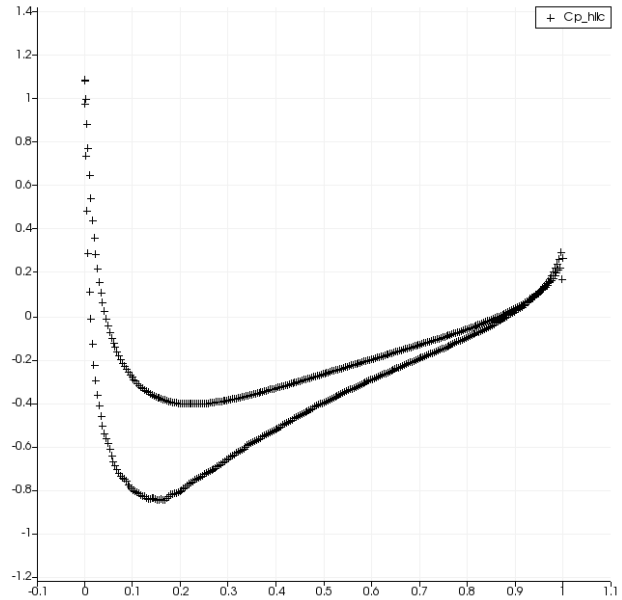

Рисунок 3.3.3

В том же окне программы *ParaView* откроем файл *experiment.csv* (рисунок 3.3.4), содержащий результаты эксперимента при аналогичном угле атаки и сравним полученные графики (рисунок 3.3.5) [11].

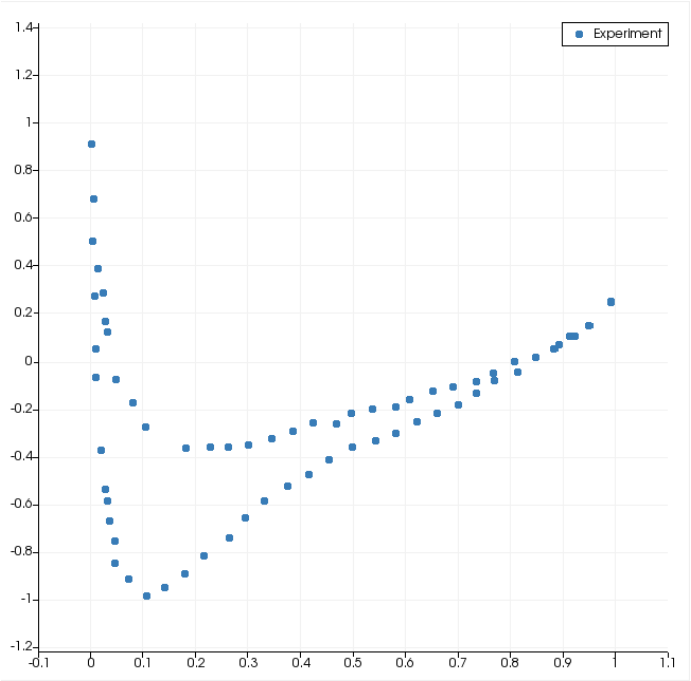

Рисунок 3.3.4

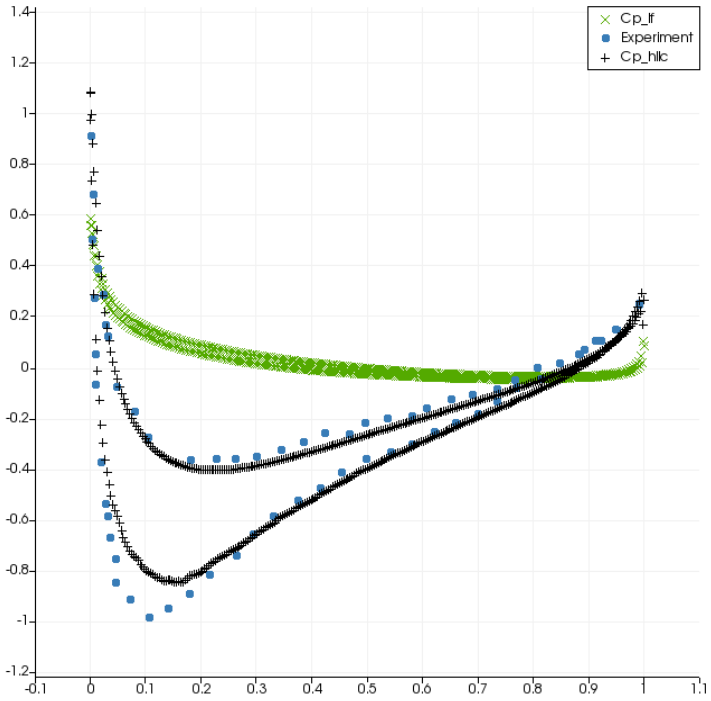

Рисунок 3.3.5

На графике (рисунок 3.3.5) видно, что результаты экспериментальных и нами полученных данных с помощью схемы *HLLC* почти совпадают, а это значит, что данная математическая модель вполне адекватна. Небольшие отличия объясняются тем, что в данной программе не учитывается вязкость газа, поток является ламинарным, то есть не учитываются возникающие турбулентности. А результаты, полученные с помощью схемы Лакса-Фридрихса сильно отстают судя по данному графику, но это объясняется тем, что данный метод "размазывает" поток и надо использовать существенно более мелкую сетку, поэтому в дальнейших вычислениях будем использовать метод *HLLC*.

Для сравнения сгенерируем сетку содержащую всего чуть более 2000 ячеек и рассчитаем распределение коэффициента давления по поверхности профиля (рисунок 3.3.6). После чего сравним результаты, полученные ранее, на изначальной сетке, включающей более 12000 ячеек (рисунок 3.3.7).

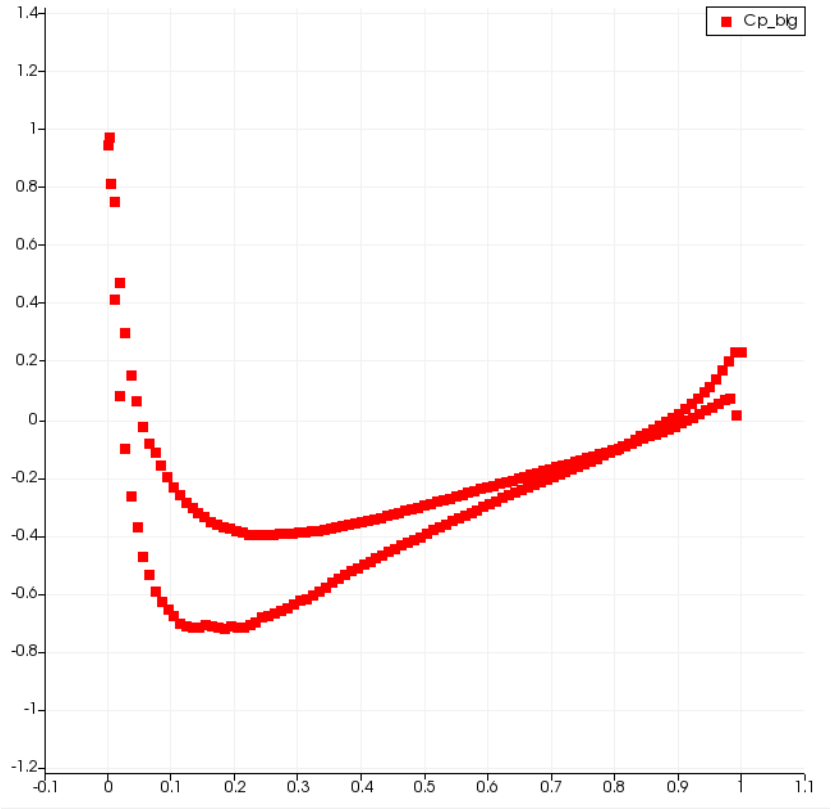

Рисунок 3.3.6

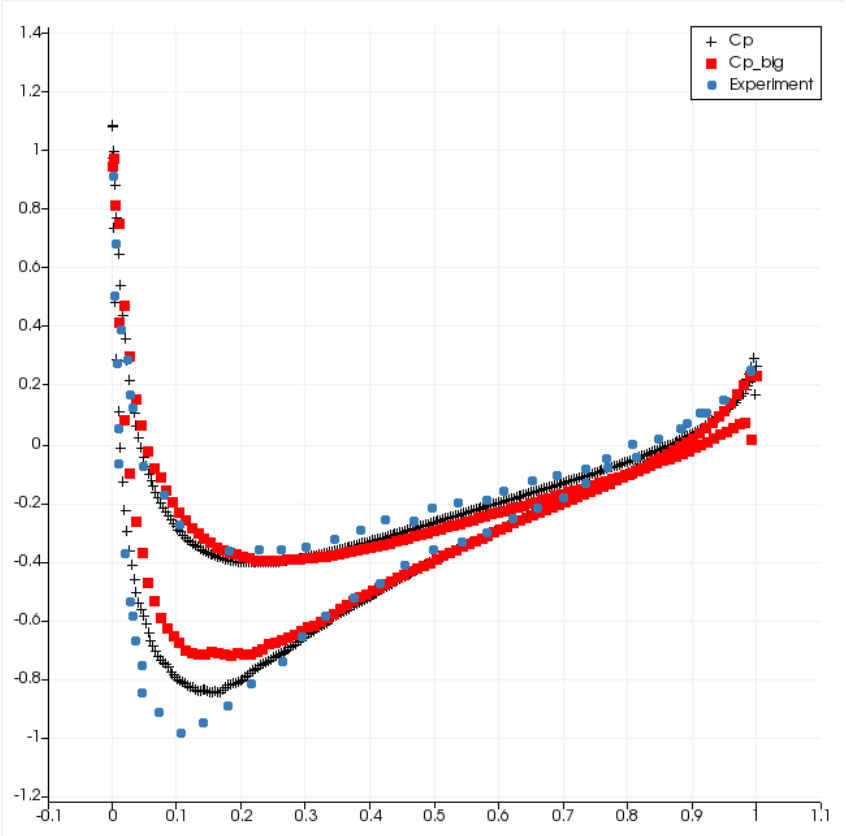

Рисунок 3.3.7

Из данного сравнения (рисунок 3.3.7) видно, что чем крупнее сетка, тем менее точен результат.

На рисунке 3.3.8 приведена визуализация результатов при различных углах атаки.

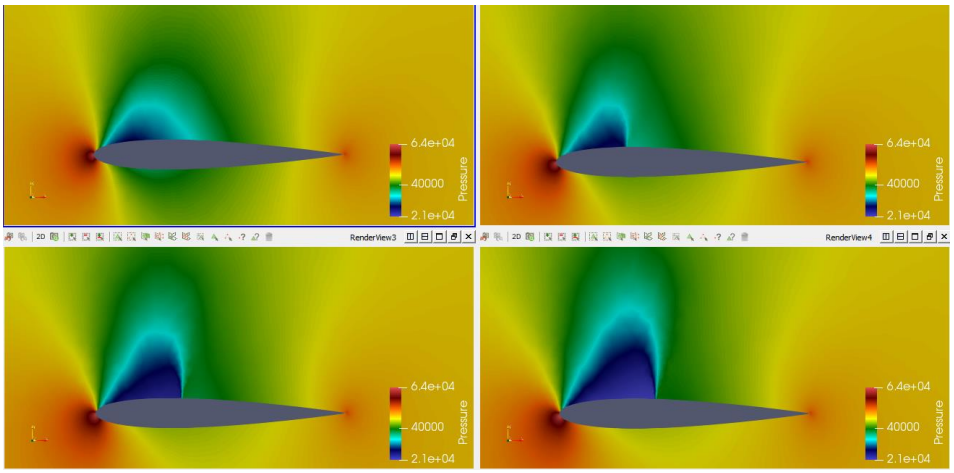

Рисунок 3.3.8

Для различных расчетов летных характеристик крыла важно знать изменение коэффициента подъемной силы C_y при различных углах атаки (формула 3.1.3). Для этой цели строится график зависимости коэффициента C_y от α (рисунок 3.3.9).

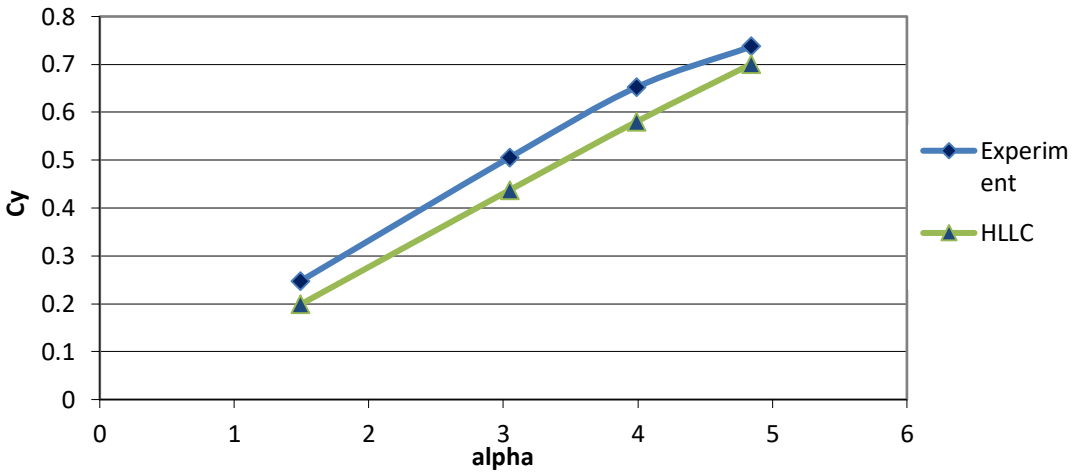

Рисунок 3.3.9

На графике из рисунка 3.3.9 видно, что результаты, полученные с помощью схемы *HLLC* и экспериментальные данные, немного различаются, что и было обосновано при сравнении из графиков на рисунке 3.3.6.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была составлена адекватная математическая модель и реализована программа для исследования аэродинамических характеристик профиля крыла, результаты которой вполне сравнимы с реальными экспериментами.

При использовании более мелкой сетки получаются более точные результаты, что говорит о сходимости методики. Следовательно, мы можем улучшить наши вычисления, используя сетку с большим количеством ячеек, но для этого потребуется более мощное аппаратное обеспечение, чем обычный персональный компьютер.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вакджира М. Б. Исторический аспект математического моделирования / М. Б. Вакджира. – М. : 2014. – 66 с.
2. Волков К. Н. Методы ускорения газодинамических расчетов на неструктурированных сетках / К. Н. Волков, Ю. Н. Дерюгин, В. Н. Емельянов, А. Г. Карпенко, А. С. Козелков, И. В. Тетерина. – М. : Физматлит, 2013. – 415 с.
3. Волков К. Н. Течения и теплообмен в каналах и вращающихся полостях / К. Н. Волков, В. Н. Емельянов. – М. : Физматлит, 2009. – 463 с.
4. Голованов Н. Н. Геометрическое моделирование / Н. Н. Голованов. – М. : Физматлит, 2002. – 472 с.
5. Железнякова А. Л. Анализ эффективности современных численных схем решения задачи о распаде произвольного разрыва в рамках метода расщепления по физическим процессам для расчета гиперзвуковых течений / А. Л. Железнякова. – М. : Институт проблем механики им. А. Ю. Ишлинского Российской академии наук, 2014. – 22 с.
6. Луцкий А. Е. GridMan3D – библиотека подпрограмм для параллельных вычислений на нерегулярных сетках / А. Е. Луцкий, А. В. Северин. – М. : Институт прикладной математики им.М. В. Келдыша, 2014. – 4 с.
7. Капустина Е. С., Альбиков Р. Р., Кулягин А. И., Фролов Т. Е., Чалдаев А. И. Решение задач газовой динамики разрывным методом Галеркина в криволинейной системе координат [Электронный ресурс] // Огарев-online (принято к печати)
8. Колесов А. Е. Построение геометрии и генерация сеток / А. Е. Колесов, М. В. Васильева – Якутск, центр вычислительных технологий Северо-Восточного федерального университета, 2017. – 16 с.
9. Куликовский А. Г. Математические вопросы численного решения гиперболических систем уравнений / А. Г. Куликовский, Н. В. Погорелов, А. Ю. Семенов – М. : Физматлит, 2001. – 608 с.

10. Gmsh 4.3.0 [Электронный ресурс]. – Режим доступа: http://gmsh.info/doc/texinfo/gmsh.html
11. Harris C. D. Two-Dimensional Aerodynamic Characteristics of the NACA 0012 Airfoil in the Langley 8-Foot Transonic Pressure Tunnel / C. D. Harris. – WA. : NASA, 1981. – 141 p.
12. ParaView guide [Электронный ресурс]. – Режим доступа: https://www.paraview.org/paraview-guide.html
13. Toro E. Riemann Solvers and Numerical Methods for Fluid Dynamics / E. Toro. – NY. : Springer, 2009.– 749 p.
14. VTK format [Электронный ресурс]. – Режим доступа: https://www.vtk.org/Wiki/VTK.html
15. Wikipedia – NACA airfoil [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/NACA_airfoil.html

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

mesh.h

```
struct Point
{
    double x;
    double y;

    Point() : x(0.0), y(0.0) {}
    Point(double ax, double ay) : x(ax), y(ay) {}

    inline void operator  = (double q) { x = q;  y = q; }
    inline void operator  = (Point p) { x = p.x;  y = p.y; }
};

typedef Point Vector;

class Cell
{
public:
    Cell(): nodesInd(0), edgesInd(0), nCount(0), eCount(0) {};
    ~Cell();

    int   nCount;
    int   eCount;
    int*  nodesInd;
    int*  edgesInd;
    int   neigh[3];
    int   type;
    char  typeName[64];
    double  S;
    Point c;
    double HX;
    double HY;
    unsigned int flag;

    friend class Grid;
};


class Edge
{
public:
    Edge(): c(0), cCount(0) {};
    ~Edge();

    int      n1;        // узел в начале
    int      n2;        // узел в конце
    int      c1;        // ячейка слева
    int      c2;        // ячейка справа, нормаль из c1 в c2
    Vector   n;         // нормаль к грани
    double   l;         // длина грани
    int      cCount;    // количество точек на грани
    Point*   c;         // точки на грани
```

```
    int      type;      // тип грани (внутр., гранич.)
    friend class Grid;
public:
    static const int TYPE_INNER          = 0;
    static const int TYPE_OUTLET  = 1;
    static const int TYPE_INLET          = 2;
    static const int TYPE_WALL           = 3;

    static const int TYPE_NAMED = 0x100;
};

class Mesh
{
public:
    Mesh(): nodes(0), cells(0), edges(0),
            nCount(0), cCount(0), eCount(0) {};
    ~Mesh();

    void initFromFile(char* fName);
    inline Point& getNode(int i) { return nodes[i]; };
    inline Cell&  getCell(int i) { return cells[i]; };
    inline Edge&  getEdge(int i) { return edges[i]; };

    Point* nodes;
    Cell*  cells;
    Edge*  edges;
    int    nCount;  //количество nodes.
    int    cCount;  //количество ячеек(cells).
    int    eCount;  //количество ребер(edges).
};
```

method.h

```
#include "mesh.h"

#define METHOD_CODE_HEAT 1
#define METHOD_CODE_GAS  2

struct Param
{
    double r;        //!< плотность
    double p;        //!< давление
    double u;        //!< первая  компонента  вектора  скорости
    double v;        //!< вторая  компонента  вектора  скорости
    double e;        //!< внутренняя   энергия
    double T;        //!< температура
    double M;        //!< число  Маха
    double tau;
    inline double magU() { return u*u+v*v; };
};

class Method
{
protected:
    Mesh * mesh;
public:
    virtual void convertToParam(int, Param&) = 0;
    void saveVTK(int step);
    virtual void init() = 0;
    virtual void run() = 0;
```

```
    static Method* create();
};
```

MethodGas.h

```
#pragma once

#include "method.h"

class MethodGas : public Method
{
public:
    MethodGas():GAM(1.4){};
    virtual void convertToParam(int, Param&);
    virtual void init();
    virtual void run();

    ~MethodGas();
protected:

    void initValues();
    void flux_lf(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv, double
&fe);
    void flux_hllc(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv, double
&fe);
    void bnd(Edge *e, Param p1, Param &p2);
    void calcTau();
    void _flux_hllc_x_1(Param prim[2], double* qr, double* qu, double* qv, double* qe);

    double *ro, *ru, *rv, *re, *tau;
    double *int_ro, *int_ru, *int_rv, *int_re, *v_max;

    double TMAX;
    double TAU;
       double CFL;

    double GAM;
};
```

mesh.cpp

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include "mesh.h"
#include <utility>
#include <vector>
#include <map>

#define __max__(x,y) ((x)>=(y) ? (x) : (y))
#define _max_(a,b,c) (__max__(__max__((a),(b)), (c)))

typedef std::pair<int, int> pointPair;

void CreateFileEdges(int cCount, Cell* cells, char* nameFile);

// загрузка сетки из файла fName.
void Mesh::initFromFile(char* fName)
{
```

```
	char str[50];
	FILE *fp;
	int tmp, n, n1, n2, n3, n4;
	double ftmp;
	char stmp[50];

	// читаем данные с файла
	sprintf(str, "%s.msh", fName);
	fp = fopen(str, "r");
	if (!fp) {
		fprintf(stderr, "Can not open file '%s'\n", str);
		exit(1);
	}

	fscanf(fp, "%s", &stmp);		//skip $MeshFormat
	fscanf(fp, "%lf %d %d", &ftmp, &tmp, &tmp);
	fscanf(fp, "%s", &stmp);		//skip $EndMeshFormat
	fscanf(fp, "%s", &stmp);		//skip $PhysicalNames
	fscanf(fp, "%d", &n);

	for (int i = 0; i < n; i++)
	{
		fscanf(fp, "%d %d %s", &tmp, &tmp, &stmp);
	}

	fscanf(fp, "%s", &stmp);		//skip $EndPhysicalNames
	fscanf(fp, "%s", &stmp);		//skip $Entities
	fscanf(fp, "%d %d %d %d", &n1, &n2, &n3, &n4);
	n = n1 + n2 + n3 + n4;
	for (int i = 0; i < n1; i++)
	{
		fscanf(fp, "%d %lf %lf %d %d", &tmp, &ftmp, &ftmp, &tmp, &tmp);
	}

	for (int i = 0; i < n2; i++)
	{
		fscanf(fp, "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf", &ftmp, &ftmp,
&ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp);
	}

	for (int i = 0; i < n3; i++)
	{
		fscanf(fp, "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
&ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp, &ftmp,
&ftmp, &ftmp, &ftmp, &ftmp);
	}

	fscanf(fp, "%s", &stmp);		//skip $EndEntities
	fscanf(fp, "%s", &stmp);		//skip $Nodes

	printf("%s\n", stmp);			//проверка, на правильность пропуска всех
предыдущих строчек (должно вывести - $Nodes)

	int countBlocks;			//numEntityBlocks(size_t)
	int countNodesBlock;			//numNodesBlock(size_t)

	fscanf(fp, "%d %d %d %d", &countBlocks, &nCount, &tmp, &tmp);

	nodes = new Point[nCount];

	int k = 0;				//индекс для считыввания координат
```

```
	for (int i = 0; i < countBlocks; i++) {
		fscanf(fp, "%d %d %d %d", &tmp, &tmp, &tmp, &countNodesBlock);
		for (int j = 0; j < countNodesBlock; j++)
			fscanf(fp, "%d", &tmp);

		for (int j = 0; j < countNodesBlock; j++) {
			fscanf(fp, "%lf %lf %lf", &(nodes[k].x), &(nodes[k].y), &ftmp);
			k++;
		}
	}

	fscanf(fp, "%s", &stmp);				//skip $EndNodes
	fscanf(fp, "%s", &stmp);				//skip $Elements

	int countElements;					//numElements(size_t)
	int countElementsBlock;					//numElementsBlock(size_t)
	int elementType;					//elementType(int)

	fscanf(fp, "%d %d %d %d", &countBlocks, &countElements, &tmp, &tmp);

	Cell* cellsTmp = new Cell[countElements];
	int cCountTmp = 0;

	for (int i = 0; i < countBlocks; i++) {
		fscanf(fp, "%d %d %d %d", &elementType, &tmp, &tmp, &countElementsBlock);
		elementType++;
		if (elementType == 1) {
			for (int j = 0; j < countElementsBlock; j++) {
				fscanf(fp, "%d %d", &tmp, &tmp);
			}
		}
		else if (elementType == 2) {
			for (int j = 0; j < countElementsBlock; j++) {
				fscanf(fp, "%d %d %d", &tmp, &tmp, &tmp);
			}
		}
		else if (elementType == 3) {
			for (int j = 0; j < countElementsBlock; j++) {
				cellsTmp[cCountTmp].nCount = 3;
				cellsTmp[cCountTmp].nodesInd = new
int[cellsTmp[cCountTmp].nCount];
				fscanf(fp, "%d %d %d %d", &tmp,
&(cellsTmp[cCountTmp].nodesInd[0]), &(cellsTmp[cCountTmp].nodesInd[1]),
&(cellsTmp[cCountTmp].nodesInd[2]));
				cellsTmp[cCountTmp].nodesInd[0]--;
				cellsTmp[cCountTmp].nodesInd[1]--;
				cellsTmp[cCountTmp].nodesInd[2]--;
				cellsTmp[cCountTmp].c.x =
(nodes[cellsTmp[cCountTmp].nodesInd[0]].x + nodes[cellsTmp[cCountTmp].nodesInd[1]].x +
nodes[cellsTmp[cCountTmp].nodesInd[2]].x) / 3.0;
				cellsTmp[cCountTmp].c.y =
(nodes[cellsTmp[cCountTmp].nodesInd[0]].y + nodes[cellsTmp[cCountTmp].nodesInd[1]].y +
nodes[cellsTmp[cCountTmp].nodesInd[2]].y) / 3.0;
				cellsTmp[cCountTmp].HX =
_max_(fabs(nodes[cellsTmp[cCountTmp].nodesInd[0]].x -
nodes[cellsTmp[cCountTmp].nodesInd[1]].x),
					fabs(nodes[cellsTmp[cCountTmp].nodesInd[1]].x -
nodes[cellsTmp[cCountTmp].nodesInd[2]].x),
					fabs(nodes[cellsTmp[cCountTmp].nodesInd[0]].x -
nodes[cellsTmp[cCountTmp].nodesInd[2]].x));
```

```
                        cellsTmp[cCountTmp].HY =
_max_(fabs(nodes[cellsTmp[cCountTmp].nodesInd[0]].y -
nodes[cellsTmp[cCountTmp].nodesInd[1]].y),
                                fabs(nodes[cellsTmp[cCountTmp].nodesInd[1]].y -
nodes[cellsTmp[cCountTmp].nodesInd[2]].y),
                                fabs(nodes[cellsTmp[cCountTmp].nodesInd[0]].y -
nodes[cellsTmp[cCountTmp].nodesInd[2]].y));
                        cellsTmp[cCountTmp].eCount = 3;
                        cellsTmp[cCountTmp].edgesInd = new
int[cellsTmp[cCountTmp].eCount];
                        for (int k = 0; k < 3; k++)
                                cellsTmp[cCountTmp].edgesInd[k] = -1;
                        cCountTmp++;
                }
        }
    }

    fclose(fp);
    cCount = cCountTmp;
    cells = new Cell[cCount];
    for (int i = 0; i < cCount; i++) {
        cells[i] = cellsTmp[i];
    }

    for (int i = 0; i < cCount; i++)
    {
        double a = sqrt(pow(nodes[cells[i].nodesInd[0]].x -
nodes[cells[i].nodesInd[1]].x, 2) + pow(nodes[cells[i].nodesInd[0]].y -
nodes[cells[i].nodesInd[1]].y, 2));
        double b = sqrt(pow(nodes[cells[i].nodesInd[2]].x -
nodes[cells[i].nodesInd[1]].x, 2) + pow(nodes[cells[i].nodesInd[2]].y -
nodes[cells[i].nodesInd[1]].y, 2));
        double c = sqrt(pow(nodes[cells[i].nodesInd[0]].x -
nodes[cells[i].nodesInd[2]].x, 2) + pow(nodes[cells[i].nodesInd[0]].y -
nodes[cells[i].nodesInd[2]].y, 2));
        double p = (a + b + c) / 2.0;
        cells[i].S = sqrt(p*(p - a)*(p - b)*(p - c));
    }


    sprintf(str, "%s_edges.txt", fName);
    fp = fopen(str, "r");
    if (!fp) {
        printf("file creation: '%s'\n", str);
        CreateFileEdges(cCount, cells, fName);
        fp = fopen(str, "r");
        if (!fp) {
            fprintf(stderr, "Can not open file '%s'\n", str);
            exit(1);
        }
        else {
            printf("file created: '%s'\n", str);
        }
    }

    fscanf(fp, "%d", &eCount);
    edges = new Edge[eCount];
    int typeEdge, n1_tmp, n2_tmp, kTmp;
    for (int i = 0; i < eCount; i++) {
        fscanf(fp, "%d", &typeEdge);
        if (typeEdge == 1) {
            fscanf(fp, "%d %d %d", &n1_tmp, &n2_tmp, &edges[i].c1);
```

```
                        edges[i].n1 = n1_tmp;
                        edges[i].n2 = n2_tmp;
                        edges[i].c2 = -1;

                        kTmp = 0;
                        while (cells[edges[i].c1].edgesInd[kTmp] > -1) kTmp++;
                        cells[edges[i].c1].edgesInd[kTmp] = i;

                        if ((nodes[n1_tmp].x == -8.0) && (nodes[n2_tmp].x == -8.0)) {
                                edges[i].type = Edge::TYPE_INLET;
                        }
                        else if ((nodes[n1_tmp].x == 8.0) && (nodes[n2_tmp].x == 8.0)) {
                                edges[i].type = Edge::TYPE_INLET;
                        }
                        else if ((nodes[n1_tmp].y == 8.0) && (nodes[n2_tmp].y == 8.0)) {
                                edges[i].type = Edge::TYPE_INLET;
                        }
                        else if ((nodes[n1_tmp].y == -8.0) && (nodes[n2_tmp].y == -8.0)) {
                                edges[i].type = Edge::TYPE_INLET;
                        }
                        else {
                                edges[i].type = Edge::TYPE_WALL;
                        }
                }
                else {
                        fscanf(fp, "%d %d %d %d", &edges[i].n1, &edges[i].n2, &edges[i].c1,
&edges[i].c2);
                        edges[i].type = Edge::TYPE_INNER;
                        kTmp = 0;
                        while (cells[edges[i].c1].edgesInd[kTmp] > -1) kTmp++;
                        cells[edges[i].c1].edgesInd[kTmp] = i;
                        kTmp = 0;
                        while (cells[edges[i].c2].edgesInd[kTmp] > -1) kTmp++;
                        cells[edges[i].c2].edgesInd[kTmp] = i;
                }
        }
        fclose(fp);
        for (int iEdge = 0; iEdge < eCount; iEdge++) {
                edges[iEdge].cCount = 3;
                edges[iEdge].c = new Point[edges[iEdge].cCount];
                double _sqrt3 = 1.0 / sqrt(3.0);
                // центр ребра
                edges[iEdge].c[0].x = (nodes[edges[iEdge].n1].x + nodes[edges[iEdge].n2].x) /
2.0;
                edges[iEdge].c[0].y = (nodes[edges[iEdge].n1].y + nodes[edges[iEdge].n2].y) /
2.0;
                edges[iEdge].n.x = nodes[edges[iEdge].n2].y - nodes[edges[iEdge].n1].y;
                edges[iEdge].n.y = nodes[edges[iEdge].n1].x - nodes[edges[iEdge].n2].x;
                edges[iEdge].l = sqrt(edges[iEdge].n.x*edges[iEdge].n.x +
edges[iEdge].n.y*edges[iEdge].n.y);
                edges[iEdge].n.x /= edges[iEdge].l;
                edges[iEdge].n.y /= edges[iEdge].l;

                Edge &edge = edges[iEdge];
                Cell &cell1 = cells[edge.c1];
                Vector nc1(cell1.c.x-edge.c[0].x, cell1.c.y-edge.c[0].y);
                double scal = nc1*edge.n;
                if (scal > 0) {
                        edge.n *= -1.;
                }
        }
}
```

Продолжение ПРИЛОЖЕНИЯ А

```
void CreateFileEdges(int cCount, Cell* cells, char* nameFile) {
	std::vector <pointPair> edgeTmp;				//вектор из ребер
	std::map <pointPair, std::vector<int>> m;		// соответствие ребро - треугольники, включающие это ребро
	int flag_p;							// 0 - пара не нашлась в списке, 1 - пара нашлась, 2 - нашлась симметричная пара
	int p1, p2;

	for (int i = 0; i < cCount; i++)
	{
		for (int j = 0; j < 3; j++) {
			flag_p = 0;

			switch (j) {
			case 0:
				p1 = 0;
				p2 = 1;
				break;
			case 1:
				p1 = 1;
				p2 = 2;
				break;
			case 2:
				p1 = 0;
				p2 = 2;
				break;
			}

			pointPair p(cells[i].nodesInd[p1], cells[i].nodesInd[p2]);
			pointPair p_sim(cells[i].nodesInd[p2], cells[i].nodesInd[p1]);

			for (int k = 0; k < edgeTmp.size(); k++) {
				pointPair pointTmp = edgeTmp[k];
				if (flag_p == 0) {
					if (p == pointTmp)  flag_p = 1;
					else if (p_sim == pointTmp) flag_p = 2;
				}
				else break;
			}

			switch (flag_p) {
			case 0:
				edgeTmp.push_back(p);
				m[p].push_back(i);
				break;
			case 1:
				m[p].push_back(i);
				break;
			case 2:
				m[p_sim].push_back(i);
				break;
			}
		}
	}

	std::map <pointPair, std::vector<int>>::iterator it;
	printf("size: %d \n", m.size());

	FILE *file;
	char str[50];

	sprintf(str, "%s_edges.txt", nameFile);
```

```
	file = fopen(str, "w");
	if (!file){
		fprintf(stderr, "Can not open file '%s'\n", str);
		exit(1);
	}

	fprintf(file, "%d\n", m.size());
	for (it = m.begin(); it != m.end(); it++)
	{
		fprintf(file, "%d\n", (*it).second.size());
		fprintf(file, "%d %d", (*it).first.first, (*it).first.second);
		for (int i = 0; i < (*it).second.size(); i++)
			fprintf(file, " %d", (*it).second[i]);
		fprintf(file, "\n");
	}
	fclose(file);
}

Mesh::~Mesh() { delete[] edges, cells, nodes; }

Cell::~Cell() { delete[] nodesInd, edgesInd; }


Edge::~Edge() { delete[] c; }
```

method.cpp

```
#include "method.h"
#include "MethodGas.h"
#include <cstdio>
#include <cstdlib>

Method* Method::create() {
	return new MethodGas();
}

void Method::saveVTK(int step) {
    char fName[50];

    sprintf(fName, "res_%010d.vtk", step);
    FILE * fp = fopen(fName, "w");
    if (!fp) {
        fprintf(stderr, "Can not open file '%s'\n", fName);
        exit(1);
    }
    fprintf(fp, "# vtk DataFile Version 2.0\n");
    fprintf(fp, "GASDIN data file\n");
    fprintf(fp, "ASCII\n");
    fprintf(fp, "DATASET UNSTRUCTURED_GRID\n");
    fprintf(fp, "POINTS %d float\n", mesh->nCount);
    for (int i = 0; i < mesh->nCount; i++)
    {
        fprintf(fp, "%f %f %f  ", mesh->nodes[i].x,  mesh->nodes[i].y, 0.0);
        if (i+1 % 8 == 0) fprintf(fp, "\n");
    }
    fprintf(fp, "\n");
    fprintf(fp, "CELLS %d %d\n", mesh->cCount, 4*mesh->cCount);
    for (int i = 0; i < mesh->cCount; i++)
    {
```

```
            fprintf(fp, "3 %d %d %d\n", mesh->cells[i].nodesInd[0], mesh-
>cells[i].nodesInd[1], mesh->cells[i].nodesInd[2]);
    }
    fprintf(fp, "\n");

    fprintf(fp, "CELL_TYPES %d\n", mesh->cCount);
       for (int i = 0; i < mesh->cCount; i++) {
            fprintf(fp, "5\n");
       }
    fprintf(fp, "\n");

    fprintf(fp, "CELL_DATA %d\nSCALARS Density float 1\nLOOKUP_TABLE default\n", mesh-
>cCount);
    for (int i = 0; i < mesh->cCount; i++)
    {
        Param p;
        convertToParam(i, p);
        fprintf(fp, "%25.16f ", p.r);
        if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
    }

    fprintf(fp, "SCALARS Pressure float 1\nLOOKUP_TABLE default\n");
    for (int i = 0; i < mesh->cCount; i++)
    {
        Param p;
        convertToParam(i, p);
        fprintf(fp, "%25.16f ", p.p);
        if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
    }

    fprintf(fp, "SCALARS Temperature float 1\nLOOKUP_TABLE default\n");
    for (int i = 0; i < mesh->cCount; i++)
    {
        Param p;
        convertToParam(i, p);
        fprintf(fp, "%25.16f ", p.T);
        if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
    }

    fprintf(fp, "SCALARS tau float 1\nLOOKUP_TABLE default\n");
    for (int i = 0; i < mesh->cCount; i++)
    {
        Param p;
        convertToParam(i, p);
        fprintf(fp, "%25.16f ", p.tau);
        if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
    }

    fprintf(fp, "VECTORS Velosity float\n");
    for (int i = 0; i < mesh->cCount; i++)
    {
        Param p;
        convertToParam(i, p);
        fprintf(fp, "%25.16f %25.16f %25.16f ", p.u, p.v, 0.0);
        if (i+1 % 8 == 0 || i+1 == mesh->cCount) fprintf(fp, "\n");
    }

    fclose(fp);
}
```

Продолжение ПРИЛОЖЕНИЯ А

MethodGas.cpp

```
#include "MethodGas.h"
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>

void MethodGas::convertToParam(int i, Param& p)
{
	p.r = ro[i];
	p.u = ru[i] / p.r;
	p.v = rv[i] / p.r;
	p.e = re[i] / p.r - p.magU() / 2.;
	p.p = p.r * p.e * (GAM - 1.0);
	p.T = 0.0;
	p.M = p.magU() / sqrt(GAM*p.p / p.r);

	p.tau = tau[i];
	if (p.M * 0 != 0) {
		printf("");
	}
}

void MethodGas::init()
{
	mesh = new Mesh();
	mesh->initFromFile((char*)"naca0012");

	ro = new double[mesh->cCount];
	ru = new double[mesh->cCount];
	rv = new double[mesh->cCount];
	re = new double[mesh->cCount];

	tau = new double[mesh->cCount];

	initValues();

	int_ro = new double[mesh->cCount];
	int_ru = new double[mesh->cCount];
	int_rv = new double[mesh->cCount];
	int_re = new double[mesh->cCount];

	v_max = new double[mesh->cCount];

	TMAX = 10.0;
	TAU = 1.e-8;
	CFL = 0.1;
}

int alpha = 1;			//1 -1.489
				//2 - 3.0462
				//3 - 3.9897
				//4 - 4.841

void MethodGas::initValues()
{
	for (int i = 0; i < mesh->cCount; i++) {
		ro[i] = 0.647432;
		switch (alpha)
		{
		case 1:
```

```
                        ru[i] = 220.8633*ro[i];
                        rv[i] = 5.741077*ro[i];
                        break;
                case 2:
                        ru[i] = 220.7004*ro[i];
                        rv[i] = 11.74487*ro[i];
                        break;
                case 3:
                        ru[i] = 220.4771*ro[i];
                        rv[i] = 15.37743*ro[i];
                        break;
                case 4:
                        ru[i] = 220.2243*ro[i];
                        rv[i] = 18.65146*ro[i];
                        break;
                }
                re[i] = (46066.16) / (GAM - 1.0) + 0.5*(ru[i] * ru[i] + rv[i] * rv[i]) /
ro[i];
        }
}
#define _MAX_(a,b) ((a)>(b) ? (a) : (b))

void MethodGas::calcTau() {

    for (int i = 0; i < mesh->cCount; i++) {
        tau[i] = TAU;
    }
    return;

    memset(v_max, 0, sizeof(double) * mesh->cCount);
       for (int ie = 0; ie < mesh->eCount; ie++) {
                Edge &e = mesh->edges[ie];
                int c1 = e.c1;
                Cell &cell1 = mesh->getCell(c1);

                Param p1, p2;
                convertToParam(c1, p1);
                if (e.c2 >= 0) {
                        convertToParam(e.c2, p2);
                } else {
                        bnd(&e, p1, p2);
                }

                double vmax = _MAX_(p1.magU()+sqrt(GAM*p1.p / p1.r), p2.magU()+sqrt(GAM*p2.p /
p2.r));
                v_max[c1] = _MAX_(v_max[c1], vmax);
                if (e.c2 >= 0) v_max[e.c2] = _MAX_(v_max[e.c2], vmax);
        }
        for (int i = 0; i < mesh->cCount; i++) {
                tau[i] = CFL * mesh->getCell(i).S / v_max[i];
        }
}

void MethodGas::run()
{
        double t = 0.0;
        int step = 0;
        int k = 1;
        while (t < TMAX) {
                //t += TAU;
                step++;
```

```
calcTau();

memset(int_ro, 0, sizeof(double)*mesh->cCount);
memset(int_ru, 0, sizeof(double)*mesh->cCount);
memset(int_rv, 0, sizeof(double)*mesh->cCount);
memset(int_re, 0, sizeof(double)*mesh->cCount);
for (int ie = 0; ie < mesh->eCount; ie++) {
	Edge &e = mesh->edges[ie];
	int c1 = e.c1;
	Cell &cell1 = mesh->getCell(c1);

	Param p1, p2;
	convertToParam(c1, p1);
	if (e.c2 >= 0) {
		convertToParam(e.c2, p2);
	}
	else {
		bnd(&e, p1, p2);
	}

	double fr, fu, fv, fe;
	flux_hllc(p1, p2, e.n, fr, fu, fv, fe);
	//flux_lf(p1, p2, e.n, fr, fu, fv, fe);

	fr *= e.l;
	fu *= e.l;
	fv *= e.l;
	fe *= e.l;
	int_ro[c1] -= fr;
	int_ru[c1] -= fu;
	int_rv[c1] -= fv;
	int_re[c1] -= fe;
	if (e.c2 >= 0) {
		int_ro[e.c2] += fr;
		int_ru[e.c2] += fu;
		int_rv[e.c2] += fv;
		int_re[e.c2] += fe;
	}
}

for (int i = 0; i < mesh->cCount; i++) {
	double _CFL = tau[i] / mesh->cells[i].S;
	ro[i] += int_ro[i] * _CFL;
	ru[i] += int_ru[i] * _CFL;
	rv[i] += int_rv[i] * _CFL;
	re[i] += int_re[i] * _CFL;
}

if (step % 100000 == 0)
{
	saveVTK(step);
	printf("Calculation results for step %d are saved.\n", step);


	char fName[50];

	sprintf(fName, "cx_%010d.csv", step);
	FILE * fp = fopen(fName, "w");
	if (!fp) {
		fprintf(stderr, "Can not open file '%s'\n", fName);
		exit(1);
	}
```

```
                        fprintf(fp, "p; r; u; v; V2; nx; ny; cx; cy; l \n");
                        for (int ie = 0; ie < mesh->eCount; ie++) {
                                Edge &e = mesh->edges[ie];
                                if (e.type == 3) {
                                        Param p;
                                        convertToParam(e.c1, p);
                                        fprintf(fp, "%25.16f; %25.16f; %25.16f; %25.16f; %25.16f;
%lf; %lf; %lf; %lf; %lf \n", p.p, p.r, p.u, p.v, p.u*p.u+p.v*p.v, e.n.x, e.n.y, e.c[0].x,
e.c[0].y, e.l);
                                }
                        }

                        fclose(fp);
                }
        }
}


void MethodGas::flux_lf(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv,
double &fe)
{
    double rol, rul, rvl, rel;
    double ror, rur, rvr, rer;
    double frl, ful, fvl, fel;
    double frr, fur, fvr, fer;
    double alpha, unl, unr, q1, q2;

    unl = n.x*pl.u + n.y*pl.v;
    unr = n.x*pr.u + n.y*pr.v;

    rol = pl.r;
    rul = pl.r*pl.u;
    rvl = pl.r*pl.v;
    rel = pl.r*(pl.e + 0.5*pl.magU());

    ror = pr.r;
    rur = pr.r*pr.u;
    rvr = pr.r*pr.v;
    rer = pr.r*(pr.e + 0.5*pr.magU());

    frl = pl.r*unl;
    ful = frl * pl.u + pl.p*n.x;
    fvl = frl * pl.v + pl.p*n.y;
    fel = (rel + pl.p)*unl;

    frr = pr.r*unr;
    fur = frr * pr.u + pr.p*n.x;
    fvr = frr * pr.v + pr.p*n.y;
    fer = (rer + pr.p)*unr;

    q1 = sqrt(pl.p*GAM / pl.r) + fabs(pl.magU());
    q2 = sqrt(pr.p*GAM / pr.r) + fabs(pr.magU());

    alpha = (q1 > q2) ? q1 : q2;

    fr = 0.5*((frl + frr) - alpha * (ror - rol));
    fu = 0.5*((ful + fur) - alpha * (rur - rul));
    fv = 0.5*((fvl + fvr) - alpha * (rvr - rvl));
    fe = 0.5*((fel + fer) - alpha * (rer - rel));
}
```

```
void MethodGas::flux_hllc(Param pl, Param pr, Vector n, double &fr, double &fu, double &fv,
double &fe)
{

    Param pn[2];
    int i;
    double _qu, _qv;

    pn[0] = pl; pn[1] = pr;
    pn[0].u =  n.x * pl.u + n.y * pl.v;
    pn[0].v = -n.y * pl.u + n.x * pl.v;
    pn[1].u =  n.x * pr.u + n.y * pr.v;
    pn[1].v = -n.y * pr.u + n.x * pr.v;

    _flux_hllc_x_1(pn, &fr, &_qu, &_qv, &fe); // n = (1,0)

    fu = _qu*n.x-_qv*n.y;
    fv = _qu*n.y+_qv*n.x;

}

#define F_HLLC_U(UK, FK, SK, SS, PK, RK, VK) (((SS)*((SK)*(UK)-(FK)) + (SK)*(
(PK)+(RK)*((SK)-(VK))*((SS)-(VK)) )) / ((SK)-(SS)))
#define F_HLLC_V(UK, FK, SK, SS, PK, RK, VK) (((SS)*((SK)*(UK)-(FK))) / ((SK)-(SS)))
#define F_HLLC_E(UK, FK, SK, SS, PK, RK, VK) (((SS)*((SK)*(UK)-(FK)) + (SK)*(
(PK)+(RK)*((SK)-(VK))*((SS)-(VK)) )*(SS)) / ((SK)-(SS)))


void MethodGas::_flux_hllc_x_1(Param prim[2], double* fr, double* fu, double* fv, double*
fe)
{
    int             i;
    double          sl, sr, p_star, s_star, p_pvrs, ql, qr, tmp, cz[2], E[2];

    cz[0] = sqrt(GAM*prim[0].p/prim[0].r);
    cz[1] = sqrt(GAM*prim[1].p/prim[1].r);

    E[0] = prim[0].e+0.5*prim[0].magU();
    E[1] = prim[1].e+0.5*prim[1].magU();

    p_pvrs = 0.5*(prim[0].p+prim[1].p)-0.5*(prim[1].u-
prim[0].u)*0.25*(prim[0].r+prim[1].r)*(cz[0]+cz[1]);
    p_star = (p_pvrs > 0.) ? p_pvrs : 0.;

    ql = (p_star <= prim[0].p) ? 1 : sqrt(1.+(GAM+1.)*(p_star/prim[0].p-1.)/(2.*GAM));
    qr = (p_star <= prim[1].p) ? 1 : sqrt(1.+(GAM+1.)*(p_star/prim[1].p-1.)/(2.*GAM));

    sl = prim[0].u-cz[0]*ql;
    sr = prim[1].u+cz[1]*qr;

    if (sl>sr) {
        tmp = sl;
        sl = sr;
        sr = tmp;
    }

    s_star  = prim[1].p-prim[0].p;
    s_star += prim[0].r*prim[0].u*(sl-prim[0].u);
    s_star -= prim[1].r*prim[1].u*(sr-prim[1].u);
    s_star /= (prim[0].r*(sl-prim[0].u)-prim[1].r*(sr-prim[1].u));

    if (s_star < sl) s_star = sl;
```

```
    if (s_star > sr) s_star = sr;


    if (!((sl <= s_star) && (s_star <= sr))) {
        printf("HLLC solver: ERROR!\n");
        exit(1);
    }

    if (sl >= 0.) {
        *fr = prim[0].r*prim[0].u;
        *fu = prim[0].r*prim[0].u*prim[0].u+prim[0].p;
        *fv = prim[0].r*prim[0].v*prim[0].u;
        *fe = (prim[0].r*E[0]+prim[0].p)*prim[0].u;
    }
    else if (sr <= 0.) {
        *fr = prim[1].r*prim[1].u;
        *fu = prim[1].r*prim[1].u*prim[1].u+prim[1].p;
        *fv = prim[1].r*prim[1].v*prim[1].u;
        *fe = (prim[1].r*E[1]+prim[1].p)*prim[1].u;
    }
    else {
        if (s_star >= 0) {
            *fr = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[0].r,
                    prim[0].r * prim[0].u,
                    sl, s_star, prim[0].p, prim[0].r, prim[0].u
            );
            *fu = F_HLLC_U( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[0].r * prim[0].u,
                    prim[0].r * prim[0].u * prim[0].u + prim[0].p,
                    sl, s_star, prim[0].p, prim[0].r, prim[0].u
            );
            *fv = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[0].r * prim[0].v,
                    prim[0].r * prim[0].u * prim[0].v,
                    sl, s_star, prim[0].p, prim[0].r, prim[0].u
            );
            *fe = F_HLLC_E( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[0].r * E[0],
                    (prim[0].r * E[0] + prim[0].p)*prim[0].u,
                    sl, s_star, prim[0].p, prim[0].r, prim[0].u
            );
        }
        else {
            *fr = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[1].r,
                    prim[1].r * prim[1].u,
                    sr, s_star, prim[1].p, prim[1].r, prim[1].u
            );
            *fu = F_HLLC_U( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[1].r * prim[1].u,
                    prim[1].r * prim[1].u * prim[1].u + prim[1].p,
                    sr, s_star, prim[1].p, prim[1].r, prim[1].u
            );
            *fv = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[1].r * prim[1].v,
                    prim[1].r * prim[1].u * prim[1].v,
                    sr, s_star, prim[1].p, prim[1].r, prim[1].u
            );
            *fe = F_HLLC_E( /*  UK, FK, SK, SS, PK, RK, VK */
                    prim[1].r * E[1],
                    (prim[1].r * E[1] + prim[1].p)*prim[1].u,
```

```
                    sr, s_star, prim[1].p, prim[1].r, prim[1].u
            );
        }
    }
}


void MethodGas::bnd(Edge *e, Param p1, Param &p2)
{
	switch (e->type) {
	case 1: // вытекание
		p2 = p1;
		break;

	case 2: // втекание
		p2.r = 0.647432;
		switch (alpha)
		{
		case 1:
			p2.u = 220.8633;
			p2.v = 5.741077;
			break;
		case 2:
			p2.u = 220.7004;
			p2.v = 11.74487;
			break;
		case 3:
			p2.u = 220.4771;
			p2.v = 15.37743;
			break;
		case 4:
			p2.u = 220.2243;
			p2.v = 18.65146;
			break;
		}
		p2.p = 46066.16;
		p2.e = p2.p / p2.r / (GAM - 1.0);
		p2.T = 248.0;
		p2.M = 0.7;
		p2.tau = p1.tau;
		break;

	case 3: // отражение
		p2 = p1;
		double Un = p1.u * e->n.x + p1.v * e->n.y;
		Vector V;
		V.x = e->n.x * Un*2.0;
		V.y = e->n.y * Un*2.0;
		p2.u = p1.u - V.x;
		p2.v = p1.v - V.y;
		break;
	}
}


MethodGas::~MethodGas()
{
	delete mesh;
	delete[] ro, ru, rv, re, tau;
	delete[] int_ro, int_ru, int_rv, int_re, v_max;
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Диск с магистерской диссертацией

Диск содержит следующие файлы:

— Магистерская диссертация (Фролов Т.Е.).pdf;

— Презентация (Фролов Т.Е.).pptx;

— program.rar.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий
Кафедра прикладной математики, дифференциальных уравнений и теоретической механики

ОТЗЫВ

Руководителя магистерской диссертации Тишкина В. Ф., д-ра физ.-мат. наук, проф., чл.-кор. РАН
(Фамилия, и. о., звание, должность)
на магистерскую диссертацию студента Фролова Тимура Евгеньевича
(Ф. И. О.)

Данная работа посвящена газовой динамики – одной из наиболее важных и актуальных сфер в области современной науки и техники.

Цель магистерской диссертации Фролова Т. Е. – разработка и реализация программного обеспечения для исследования аэродинамических характеристик профилей крыльев.

Для достижения данной цели Фролов Т. Е. решил следующие задачи:

- рассмотрел схемы решения задач газовой динамики;
- разработал методику решения двухмерных задач газовой динамики с использованием неструктурированных сеток;
- провел расчеты на конкретной задаче и сравнил полученные результаты с данными эксперимента для оценки работы методики.

При выполнении магистерской диссертации Фролов Т. Е. в полной мере продемонстрировал, что владеет знаниями и навыками, необходимыми для решения поставленных задач, а также зарекомендовал себя как специалист, способный самостоятельно решать научные и практические задачи.

Данная магистерская диссертация выполнена в соответствии с требованиями, предъявляемым к выпускным квалификационным работам и заслуживает оценки «отлично».

_____ В. Ф. Тишкин «04» 06 2019 г
(подпись)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.П. ОГАРЁВА"

ОТЗЫВ РЕЦЕНЗЕНТА
О ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Студента Фролова Тимура Евгеньевича
(Фамилия, имя, отчество)
Факультет математики и информационных технологий
Кафедра ПМДУиТМ Группа 201М
Направление подготовки 01.04.02 Прикладная математика и информатика
Квалификация (степень) магистрант
Наименование темы: Численное решение двумерных задач газовой динамики с использованием неструктурированных сеток
Рецензен Бадокина Т. Е., ФГБОУ ВО «НИ МГУ им. Н. П. Огарева» кафедра фундаментальной информатики, кандидат физико-математических наук
(Фамилия И.О., место работы, ученое звание, степень)

ОЦЕНКА ВЫПУСКНОЙ РАБОТЫ

№ п/п	Показатели		Оценка				
			5	4	3	2	0*
1.	Актуальность тематики работы		+				
2.	Степень полноты обзора состояния вопроса и корректность постановки задачи		+				
3.	Уровень и корректность использования в работе методов исследования, математического моделирования		+				
4.	Степень комплексности работы, применение в ней знаний естественнонаучных, социально-экономических, общепрофессиональных и специальных дисциплин			+			
5.	Ясность, четкость, последовательность и обоснованность изложения		+				
6.	Применение современного математического и программного обеспечения, компьютерных технологий в работе		+				
7.	Качество оформления (общий уровень грамотности, стиль изложения, качество иллюстраций, соответствие требованиям стандарта)		+				
8.	Оригинальность и новизна полученных результатов (научных, конструкторских и технологических решений)		+				
9.	Тип работы	фундаментальная с оригинальными результатами					
		Реферативная					
		Прикладная	+				
10.	Рекомендации	к опубликованию	+				
		к внедрению					
ИТОГОВАЯ ОЦЕНКА			отлично				

* – не оценивается (трудно оценить)

Отмеченные достоинства: Поставленная задача полностью решена, цель работы достигнута. В данной работе были рассмотрены схемы численного решения задач газовой динамики и реализована программа для исследования аэродинамических характеристик профиля крыла.

Отмеченные недостатки: В разработанной программе не учитывается вязкость и турбулентное течение газа, что сказывается на точности получаемых результатов.

Заключение: Магистерская диссертация Фролова Тимура Евгеньевича соответствует всем требованиям, предъявляемым к магистерским диссертациям, и заслуживает оценки «отлично».

06. 06. 2019 г. Рецензент _____
(подпись)

ОТЧЕТ
о результатах проверки работы обучающегося
на наличие заимствований

Ф.И.О. автора работы Фролов Тимур Евгеньевич

Тема работы Численное решение двумерных задач газовой динамики с использованием неструктурированных сеток

Руководитель Тишкин В. Ф., д-р физ.-мат. наук, проф., чл.-кор. РАН

Представленная работа прошла проверку на наличие заимствований в системе «Антиплагиат.ВУЗ» (*или иной аналогичной системе, анализа текстов на наличие заимствований, выбранной Университетом*).

Результаты автоматической проверки: оригинальность – 88,52%
цитирования – 0,29%
заимствования – 11,19%

Результаты анализа полного отчета на наличие заимствований:

правомерные заимствования: да, 11,19%, введение понятий и утверждений, известных ранее по теме работы;
да/нет, количество(%), обоснованность

корректные цитирования: да, 0,29%, использование общеупотребительных выражений;
да/нет, количество(%), обоснованность

неправомерные заимствования: нет;
да/нет, количество(%), обоснованность

признаки обхода системы: нет;
(да/нет, описание)

Общее заключение об итоговой оригинальности работы и возможности ее допуска к защите:

Данная работа не содержит элементов неправомерных заимствований и отвечает всем нормам Положения о проверке работ обучающихся ФГБОУ ВО «МГУ им. Н. П. Огарёва» на наличие заимствований.

Руководитель
д-р физ.-мат. наук, проф., чл.-кор. РАН 08.06.19 В. Ф. Тишкин

Заявление о самостоятельном характере выполнения работы

Я, Фролов Тимур Евгеньевич, обучающийся 2 курса, направления подготовки/специальности 01.04.02 Прикладная математика и информатика (код и наименование направления подготовки/специальности), заявляю, что в моей работе на тему «Численное решение двумерных задач газовой динамики с использованием неструктурированных сеток», представленной в Государственную экзаменационную комиссию для публичной защиты, не содержится элементов неправомерных заимствований.

Все прямые заимствования из печатных и электронных источников, а также ранее защищенных письменных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим в Университете Положением о проверке работ обучающихся ФГБОУ ВО «МГУ им. Н. П. Огарёва» на наличие заимствований, в соответствии с которым обнаружение неправомерных заимствований является основанием для отрицательного отзыва руководителя работы.

_____ «8» 06 2019 г.

Работа представлена для проверки в Системе:

«8» 06 2019 г. _____ В. Ф. Тишкин

Заведующему кафедрой прикладной
математики, дифференциальных
уравнений и теоретической
механики
Р. В. Жалнину
студента 2 курса
очной формы обучения
(на бесплатной основе)
направления подготовки
«Прикладная математика и
информатика»
факультета математики и
информационных технологий
Фролова Тимура Евгеньевича

заявление.

Прошу разместить мою выпускную квалификационную работу на тему «Численное решение двумерных задач газовой динамики с использованием неструктурированных сеток» в электронной библиотечной системе университета в полном объеме.

«24» 06 2019 г. _____