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
25 июня 2020

БАКАЛАВРСКАЯ РАБОТА

МОДЕЛИРОВАНИЕ АКУСТИЧЕСКИХ ПОЛЕЙ ПРИ ОБТЕКАНИИ ТЕЛ ПОТОКОМ ГАЗА

Автор бакалаврской работы 12.06.2020 А. Р. Багапов

Обозначение бакалаврской работы БР–02069964–01.03.02–02–20

Направление 01.03.02 Прикладная математика и информатика

Руководитель работы

канд. физ.-мат. наук, доц. 12.06.2020 Р. В. Жалнин

Нормоконтролер

канд. физ.-мат. наук, доц. 18.06.2020 Д. К. Егорова

Саранск
2020

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
20 января 2020

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Багапов Ариф Ренатович, 403 группа
1 Тема Моделирование акустических полей при обтекании тел потоком газа
Утверждена приказом № 286-с от 20.01.2020
2 Срок представления работы к защите 27.06.2020
3 Исходные данные для научного исследования (проектирования) Литература по газовой динамике, акустике, численным методам, программированию
4 Содержание выпускной квалификационной работы
4.1 Алгоритм моделирования течения газа
4.2 Расчёт акустических характеристик
4.3 Моделирование акустических полей при обтекании различных тел
5 Приложения
5.1 Приложение А (обязательное) Листинг программы

Руководитель работы 20.01.2020 Р. В. Жалнин

Задание принял к исполнению 20.01.2020 А. Р. Багапов

РЕФЕРАТ

Бакалаврская работа содержит 62 страницы, 20 рисунков, 2 таблицу, 12 использованных источников, 1 приложение.

ГАЗОВАЯ ДИНАМИКА, HLLC, ENO, WENO, PARAVIEW, ОБТЕКАНИЕ, АКУСТИЧЕСКОЕ ПОЛЕ, OASPL, КУБИЧЕСКИЙ ОБЪЕКТ, КАСКАД, СТУПЕНЬКА, КАВЕРНА.

Цель работы – создать математическую модель акустического поля, возникающего при обтекании объектов и тел газом.

При решении задачи использовалась среда разработки Microsoft Visual Studio 2019, язык программирования C++, пакет интерактивной визуализации Paraview 5.6.0, программа Microsoft Office Excel.

Степень внедрения – частичная.

Область применения – расчёты в области газовой динамики, акустики.

Эффективность – данная работа направлена на изучение численных методов высокого порядка точности, а также на понимание процессов, происходящих при обтекании различных объектов потоками газа.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5
1. Алгоритм моделирования течения газа 6
1.1 Разностная схема для расчёта газодинамических течений 6
1.2 Вычисление дискретных потоков 8
1.3 Алгоритм реконструкции газодинамических параметров 12
1.3.1 ENO алгоритм 12
1.3.2 WENO алгоритм 16
1.4 Дискретизация по времени 20
2. Расчёт акустических характеристик 21
2.1 Анализ полученной модели с помощью пакета Paraview 21
2.2 Анализ колебаний давления 22
3. Моделирование акустических полей при обтекании различных тел 25
3.1 Один кубический блок 25
3.2 Каскад из двух кубических блоков 29
3.3 Ступенька 34
3.4 Каверна (выемка) 39
ЗАКЛЮЧЕНИЕ 44
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 45
ПРИЛОЖЕНИЕ А (обязательное) Листинг программы 47

ВВЕДЕНИЕ

В настоящее время методы математического моделирования повсеместно внедряются в производство, численный расчёт становится мощным способом повысить надёжность и точность инженерных исследований. Это касается, прежде всего, тех областей, в которых конструкторские ошибки стоят очень дорого: машиностроение, архитектура, освоение космоса и пр.. В частности, одной из сфер, в которых широко применяются численные методы моделирования, является авиация. Расчёт и анализ аэродинамических и акустических параметров воздушных судов полезен как в мирных, так и в военных целях.

Расчёт данных параметров однозначно приводит к уравнениям газовой динамики, решение которых с помощью ЭВМ наиболее целесообразно в силу сложности и нелинейности этих уравнений. При решении практических задач зачастую приходится сталкиваться с газодинамическими течениями, характеризующимися нестационарностью, нелинейностью происходящих процессов, разнохарактерным и сложным механизмом взаимодействия, для моделирования которых необходимо использовать численные методы повышенного порядка точности, чтобы получить максимально близкие к реальным результатам.

В данной работе рассматривается задача моделирования акустических полей при обтекании тел потоком газа. Для решения задачи использовались алгоритмы WENO для реконструкции газодинамических параметров, программный комплекс Paraview для сбора данных с полученной модели, программа MS Excel для анализа собранных данных при помощи формул, позволяющих интерпретировать колебания давления как акустические.

1. Алгоритм моделирования течения газа

1.1 Разностная схема для расчёта газодинамических течений

Математическая модель изучаемых в работе процессов представляет собой закон сохранения масс, энергии и импульса [4]:

$$\frac{\partial \rho}{\partial t} + \mathrm{div}(\rho\vec{v}) = 0,$$
$$\frac{\partial \rho\vec{v}}{\partial t} + \mathrm{div}\overrightarrow{\Pi} = 0, \qquad (1.1.1)$$
$$\frac{\partial e}{\partial t} + \mathrm{div}(\vec{v}e) = -\mathrm{div}(\rho\vec{v}) + Q + \vec{F}\cdot\vec{v} - \mathrm{div}(\overrightarrow{W}),$$
$$p = p(\rho, T), \qquad \varepsilon = \varepsilon(\rho, T).$$

Здесь ρ – плотность газа,

p – давление,

$\vec{v} = \{u, v\}$ – вектор скорости частицы,

ε – удельная внутренняя энергия на единицу массы,

$e = \rho\left(\varepsilon + \frac{|\vec{v}|^2}{2}\right)$ – полная энергия,

T – температура газа,

$\Pi = \rho\vec{v}\otimes\vec{v} + p\mathbf{I}$ – тензор плотности потока импульса (**I** – единичный тензор),

Q – мощность тепловых источников,

$\vec{F}$ – некоторая сила, действующая извне (например, сила тяжести),

$\overrightarrow{W}$ – вектор плотности теплового потока. В качестве уравнения состояния принимается уравнение состояния идеального газа с показателем адиабаты γ:

$$p = (\gamma - 1)\varepsilon\rho. \tag{1.1.2}$$

В векторной форме система уравнений (1.1.1) без членов, которыми в исследовании пренебрегаем, приобретает вид:

$$\frac{\partial U}{\partial t} + \frac{\partial F^{(1)}U}{\partial x} + \frac{\partial F^{(2)}U}{\partial y} = 0, \tag{1.1.3}$$

где

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix}, \; F^{(1)}U = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \end{pmatrix}, \; F^{(2)}U = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(e+p) \end{pmatrix}. \tag{1.1.4}$$

При рассмотрении конкретной модели также необходимо задать начальные и граничные условия для полного описания решаемой задачи. Начальные условия показывают состояние системы в начальный момент времени, а граничные задают состояние системы на её границе: это могут быть условия непротекания, неприлипания, неограниченного втекания или вытекания.

Для построения дискретной модели область непрерывного изменения аргумента накроем сеткой, равномерной по каждому направлению:

$$\omega_\Delta = \omega_{\Delta_x} \times \omega_{\Delta_y},$$

где

$$\begin{aligned} \omega_{\Delta_x} &= \{\Delta_i, i = 1, \dots, N_x, \Delta_i = x_i - x_{i-1}, |\Delta_i| = h_x, h_x N_x = L_x\}, \\ \omega_{\Delta_y} &= \{\Delta_j, j = 1, \dots, N_y, \Delta_j = y_j - y_{j-1}, |\Delta_j| = h_y, h_y N_y = L_y\}. \end{aligned} \tag{1.1.5}$$

Здесь L_x, L_y–размеры исследуемой области по соответствующим направлениям. Все газодинамические функции будем задавать как средние значения в ячейках сетки.

Для аппроксимации системы (1.1.3) использовалась нелинейная консервативная квазимонотонная дифференциально-разностная схема [5]:

$$\frac{dU_{ij}}{dt} + \frac{H^{(1)}_{i+\frac{1}{2}j} - H^{(1)}_{i-\frac{1}{2}j}}{h_x} + \frac{H^{(2)}_{ij+\frac{1}{2}} - H^{(2)}_{ij-\frac{1}{2}}}{h_y} = 0, \qquad (1.1.6)$$

где $H^{(1)}_{i+1/2j}$, $H^{(2)}_{ij+1/2}$ – дискретные потоки, вычисляемые согласно пункту 1.2.

1.2 Вычисление дискретных потоков

Дискретные потоки на границах между ячейками вычислялись с помощью схемы HLLC (Harten-Lax-van Leer-Contact). Впервые она была представлена Э. Торо в 1994 году. Данная схема является модификацией схемы HLL, заключающейся в учёте промежуточных волн. Основная идея обеих схем состоит в том, чтобы искать числовой поток $H_{i+1/2}$ напрямую, а не как в исходном методе Годунова, где сначала ищется решение $Q_{i+1/2}$ для задачи Римана, которое используется для оценки интеграла по времени от потока и, отсюда, находится числовой поток.

Для определения числового потока для двумерных уравнений Эйлера, который является нормальным для внутренности ячейки в силу вращательной инвариантности, достаточно рассмотреть расширенные одномерные уравнения Эйлера, выровненные в направлении нормали. Предположим, без ограничения общности, что нормальным является направление по оси *Ox*. Поэтому интересующие нас уравнения примут вид:

$$\frac{\partial Q}{\partial t} + \frac{\partial F(Q)}{\partial x} = 0, \tag{1.2.1}$$

где Q и $F(Q)$ есть U и $F^{(1)}U$ из (1.1.4) соответственно.

Известно [6], что вектор-столбец собственных значений соответствующего якобиана представляется следующим образом:

$$\lambda(Q) = \begin{pmatrix} u - c \\ u \\ u \\ u + c \end{pmatrix}, \tag{1.2.2}$$

где $c = \sqrt{\frac{p\gamma}{\rho}}$ – скорость звука.

В каждой ячейке мы рассматриваем локальную задачу Римана [11]:

$$\partial_t Q + \partial_x F(Q) = 0,$$
$$Q(x,0) = \begin{cases} Q_L \equiv Q_i^n, \text{если } x < 0, \\ Q_R \equiv Q_{i+1}^n, \text{если } x > 0. \end{cases} \tag{1.2.3}$$

Чтобы получить поток HLLC, рассмотрим волновую картину на рисунке 1, где присутствует промежуточная волна скорости S*.

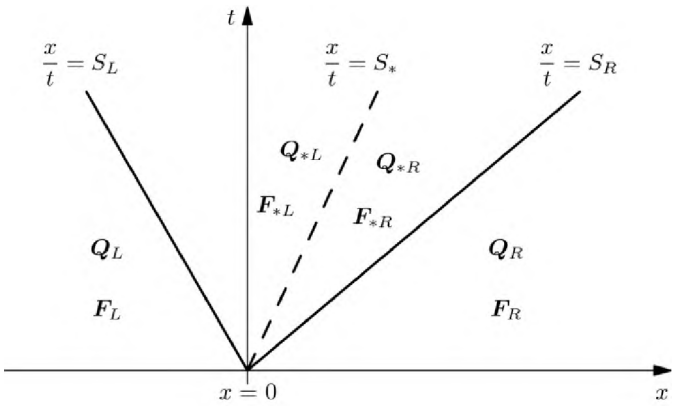


Рисунок 1

Предположим, что оценки скорости S_L, S_*, S_R для трёх семейств волн нам известны (они будут описаны в работе ниже). Применение интегральной формы законов сохранения в области $[x_L,0]\times[0,T]$ и $[x_R,0]\times[0,T]$ даёт, соответственно:

$$\begin{aligned} F_{*L} &= F_L + S_L(Q_{*L} - Q_L), \\ F_{*R} &= F_R + S_R(Q_{*R} - Q_R), \end{aligned} \tag{1.2.4}$$

где

$$Q_{*K} = [\rho_{*K}, \rho_{*K}u, \rho_{*K}v, \rho_{*K}e]^T, \tag{1.2.5}$$

где $K = L, R$.

Это приводит к алгебраической системе с количеством неизвестных большим, чем количество уравнений. Один из способов решения этой проблемы состоит в том, чтобы ввести ряд предположений, согласующихся с точным решением задачи Римана (1.2.3), что приводит к следующим выражениям для векторов промежуточного состояния в (1.2.4):

$$Q_{*K} = \rho_K \left(\frac{S_K - u_K}{S_K - S_*}\right) \begin{bmatrix} 1 \\ S_* \\ v_K \\ \frac{e_K}{\rho_K} + (S_* - u_K)\left[S_* + \frac{p_K}{\rho_K(S_K - u_K)}\right] \end{bmatrix}, \tag{1.2.6}$$

где $K = L, R$. Тогда промежуточные потоки F_{*L} и F_{*R} в (1.2.4) полностью определены, и числовой поток имеет следующий вид:

$$F^{HLLC}_{i+\frac{1}{2}} = \begin{cases} F_L, \text{если } 0 \le S_L, \\ F_{*L}, \text{если } S_L \le 0 \le S_*, \\ F_{*R}, \text{если } S_* \le 0 \le S_R, \\ F_R, \text{если } 0 \ge S_R. \end{cases} \tag{1.2.7}$$

Для схемы HLLC необходимы оценки S_L, S_R и S_*. Следующий выбор оценки, основанный на оценке давления в так называемой «звёздочной» области (области между S_L и S_R) оказался полезным. Для начала необходимо найти оценки S_L и S_R:

$$\begin{aligned} S_L &= u_L - c_L q_L, \\ S_R &= u_R + c_R q_R, \end{aligned} \tag{1.2.8}$$

где

$$q_K = \begin{cases} 1, \text{если } \bar{p}_* \le p_K, \\ \left[1 + \frac{\gamma+1}{2\gamma}\left(\frac{\bar{p}_*}{p_K} - 1\right)\right]^{\frac{1}{2}}, \text{если } \bar{p}_* > p_K. \end{cases} \tag{1.2.9}$$

Здесь $\bar{p}_*$ – оценка давления p_* внутри «звёздочной» области, а $K = L, R$. Обратим внимание, что в (1.2.9) проводится различие между разрежением и ударом. То есть, если волна является волной разрежения ($\bar{p}_* \le p_K$), то скорость S_K будет соответствовать характеристической скорости разрежения, которая является скоростью самой быстрого сигнала. Если имеется ударная волна ($\bar{p}_* > p_K$), то скорость находится как приближение истинной скорости удара[10].

Что касается $\bar{p}_*$, то надёжным выбором является двухразреженное приближение:

$$\bar{p}_* = \left[\frac{c_L + c_R - \frac{\gamma-1}{2}(u_R - u_L)}{\frac{c_L}{p_L^z} + \frac{c_R}{p_R^z}}\right]^{\frac{1}{z}}, z = \frac{\gamma-1}{2\gamma}. \tag{1.2.10}$$

Это приближение получается, если априори предположить, что две волны давления (самые быстрые волны в решении задачи Римана) являются волнами разрежения.

После нахождения оценок S_L и S_R можно дать оценку S_*:

$$S_* = \frac{p_R - p_L + \rho_L u_L (S_L - u_L) - \rho_R u_R (S_R - u_R)}{\rho_L (S_L - u_L) - \rho_R (S_R - u_R)}. \quad (1.2.11)$$

Как правило, предложенные оценки скорости были очень надёжными на протяжении многих лет [11].

1.3 Алгоритм реконструкции газодинамических параметров

1.3.1 ENO алгоритм

Алгоритм ENO (Essentially Non-Oscillatory – существенно неосциллирующие) описывает построение и локальный выбор аппроксимационной формулы с наименьшими осцилляционными свойствами (то есть наименьшей величиной колебаний) из набора формул того же порядка.

Пусть задана равномерная сетка с шагом Δx:

$$x_{\frac{1}{2}} < \ldots < x_{i-\frac{1}{2}} < \ldots < x_{i+\frac{1}{2}} x_{N+\frac{1}{2}}, \quad (1.3.1.1)$$

будем считать, что эту сетку, при необходимости, мы можем продолжить вправо и влево, добавив элементы справа или слева соответственно. Пусть заданы средние значения некоторой функции $v(x)$ в ячейках сетки:

$$\bar{v}_1 < \ldots < \bar{v}_i < \ldots < \bar{v}_N, \quad (1.3.1.2)$$

где

$$\bar{v}_i = \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} v(x)dx.$$

Введём обозначение I_i ячейки $\left(x_{i-1/2}, x_{i+1/2}\right)$ сетки. Для каждой ячейки I_i $(i = \overline{1,N})$ построим полином *p(x)* степени не большей *K–1*, который интерполирует функцию $v(x)$ с порядком точности *K* в пределах данной ячейки, т.е.

$$p(x) = v(x) + O(\Delta x^K), \tag{1.3.1.3}$$

при $x \in I_i$ и

$$\frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} p(x)dx = \bar{v}_i. \tag{1.3.1.4}$$

Тогда, обозначив $v^+_{i-1/2} = p\left(x_{i-1/2}\right)$ и $v^-_{i+1/2} = p\left(x_{i+1/2}\right)$, получим

$$\begin{aligned} v^+_{i-\frac{1}{2}} &= v\left(x_{i-\frac{1}{2}}\right) + O(\Delta x^K), \\ v^-_{i+\frac{1}{2}} &= v\left(x_{i+\frac{1}{2}}\right) + O(\Delta x^K). \end{aligned} \tag{1.3.1.5}$$

Такой полином получается [9], если рассмотреть первообразную *V(x)* функции $v(x)$:

$$V(x) = \int_{-\infty}^{x} v(\xi)d\xi,$$

и построить полином *P(x)* степени *K*, который будет интерполировать функцию *V(x)* на границах ячеек сетки. Положим $p(x) = P'(x)$.

Представив полином $P(x)$ в виде интерполяционного полинома Лагранжа [3]:

$$P(x) = \sum_{m=0}^{K} V(x_{i-r+m-\frac{1}{2}}) \prod_{\substack{l=0 \\ l\neq m}}^{K} \frac{x - x_{i-r+l-1/2}}{x_{i-r+m-1/2} - x_{i-r+l-1/2}},$$

при $x = x_{i+1/2}$ получим

$$p(x_{i+1/2}) = \sum_{j=0}^{K-1} \left(\sum_{m=j+1}^{K} \frac{\sum_{\substack{l=0 \\ l\neq 0}}^{K} \prod_{\substack{q=0 \\ q\neq m,l}}^{K} (x_{i+1/2} - x_{i-r+q-1/2})}{\prod_{\substack{l=0 \\ l\neq 0}}^{K} (x_{i-r+m-1/2} - x_{i-r+l-1/2})} \right) \bar{v}_{i-r+j} \Delta x_{i-r+j}.$$

Далее на равномерной сетке с шагом Δx для искомых значений заданной функции $v(x)$ в точках $x_{i-1/2}$ *и* $x_{i+1/2}$ получаем выражения:

$$\begin{aligned} v_{i-1/2}^{+} &= \sum_{j=0}^{K-1} \tilde{c}_{rj} \bar{v}_{i-r+j}, \\ v_{i+1/2}^{-} &= \sum_{j=0}^{K-1} c_{rj} \bar{v}_{i-r+j}, \end{aligned} \qquad (1.3.1.6)$$

где

$$c_{rj} = \sum_{m=j+1}^{K} \frac{\sum_{\substack{l=0 \\ l\neq m}}^{K} \prod_{\substack{q=0 \\ q\neq m,l}}^{K} (r-q+1)}{\prod_{\substack{l=0 \\ l\neq m}}^{K} m-l}. \qquad (1.3.1.7)$$

Значения c_{ij} для K = 0,1,2,…,5 приведены в таблице 1.

Т а б л и ц а 1 – Значения коэффициентов c_{rj}.

K	r	j=0	j=1	j=2	j=3	j=4
1	-1 0	1 1				
2	-1 0 1	3/2 1/2 -1/2	-1/2 1/2 3/2			
3	-1 0 1 2	11/6 1/3 -1/6 1/3	-7/6 5/6 5/6 -7/6	1/3 -1/6 1/3 11/6		
4	-1 0 1 2 3	25/12 1/4 -1/12 1/12 -1/4	-23/12 13/12 7/12 -5/12 13/12	13/12 -5/12 7/12 13/12 -23/12	-1/4 1/12 -1/12 1/4 25/12	
5	-1 0 1 2 3 4	137/60 1/5 -1/20 1/30 -1/20 1/5	-163/60 77/60 9/20 -13/60 17/60 -21/20	137/60 -43/60 47/60 47/60 -43/60 137/60	-21/20 17/60 -13/60 9/20 77/60 -163/60	1/5 -1/20 1/30 -1/20 1/5 137/60

Таким способом можно получить K полиномов $p_r(x), r = 0, \dots, K-1$, удовлетворяющих условиям (1.3.1.3), (1.3.1.4). Каждый полином будет соответствовать шаблону

$$S_r = \left\{ x_{i-r-\frac{1}{2}}, \dots, x_{i-r+K+\frac{1}{2}} \right\}, \qquad (1.3.1.8)$$

где $r = 0, \dots, K-1$.

Суть алгоритма ENO заключается в том, что из всех K возможных полиномов выбирается наиболее гладкий на отрезке $[x_{i-1/2}, x_{i+1/2}]$. Выбор осуществляется следующим образом [9]. Сначала рассматривается шаблон, состоящий из границ одной ячейки:

$$\{x_{i-1/2}, x_{i+1/2}\},$$

далее, если

$$\left|V\left[x_{i-3/2}, x_{i-1/2}, x_{i+1/2}\right]\right| < \left|V\left[x_{i-1/2}, x_{i+1/2}, x_{i+3/2}\right]\right|,$$

добавляется точка $x_{i-3/2}$, в противном случае $x_{i+3/2}$. И так далее, пока в шаблон не будут включены K+1 точек. Здесь

$$V\left[x_{i-1/2}, \dots, x_{i+s+1/2}\right] = \frac{V\left[x_{i+1/2}, \dots, x_{i+s+1/2}\right] - V\left[x_{i-1/2}, \dots, x_{i+s-1/2}\right]}{x_{i+s+1/2} - x_{i-1/2}},$$

$$V[x_{i-1/2}] = V(x_{i-1/2}).$$

Такой способ выбора шаблона обусловлен представлением искомого полинома в виде интерполяционного полинома Ньютона:

$$P(x) = V(x_{i-r-1/2}) + V[x_{i-r-1/2}, x_{i-r+1/2}](x - x_{i-r-1/2}) + \cdots$$
$$+ V[x_{i-r-1/2}, \dots, x_{i+s+1/2}](x - x_{i-r-1/2}) \dots (x - x_{i+s+1/2}).$$

Однако, более точные результаты даёт использование WENO алгоритма.

1.3.2 WENO алгоритм

Алгоритм схемы WENO (Weighted ENO – взвешенные ENO) [9] построения интерполяционного полинома основан на ENO схеме, но в

отличие от неё использует для аппроксимации комбинации всех возможных шаблонов для данной ячейки:

$$v_{i+1/2} = \sum_{r=0}^{K-1} \omega_r v_{i+1/2}^{(r)}, \qquad (1.3.2.1)$$

где

$$v_{i+1/2}^{(r)} = \sum_{j=0}^{K-1} c_{ij} \bar{v}_{i-r+j}, \qquad r = 0, \dots, K-1. \qquad (1.3.2.2)$$

Таким образом, необходимо найти весовые коэффициенты ω_r, $r = K-1$, удовлетворяющие условиям

$$\omega_r > 0, \sum_{r=0}^{K-1} \omega_r = 1, \qquad (1.3.2.3)$$

и сохраняющие все свойства существенно неосциллирующих систем.

Для гладкой на каждом шаблоне S_r, $r = 0, \dots, K-1$, функции $v(x)$ существуют константы $d_r, r = 0, \dots, K-1$ такие, что

$$v_{i+1/2} = \sum_{r=0}^{K-1} d_r v_{i+1/2}^{(r)} = v(x_{i+1/2}) + O(\Delta x^{2K-1}). \qquad (1.3.2.4)$$

Например, для $1 \le K \le 3$ имеем

$$d_0 = 1, \qquad k = 1;$$
$$d_0 = \frac{2}{3}, \; d_1 = \frac{1}{3}, k = 2; \qquad (1.3.2.5)$$

$$d_0 = \frac{3}{10},\ d_1 = \frac{3}{5}, d_2 = \frac{1}{10}, k = 3.$$

Видно, что константы всегда положительны, а их сумма равна единице.

Если

$$\omega_r = d_r + O(\Delta x^{K-1}), r = 0, \dots, K-1, \qquad (1.3.2.6)$$

то получим, что

$$v_{i+1/2} = \sum_{r=0}^{K-1} \omega_r v_{i+1/2}^{(r)} = v(x_{i+\frac{1}{2}}) + O(\Delta x^{2K-1}),\ \ r = 0, \dots, K-1. \qquad (1.3.2.7)$$

Действительно,

$$\sum_{r=0}^{K-1} \omega_r v_{i+1/2}^{(r)} - \sum_{r=0}^{K-1} d_r v_{i+\frac{1}{2}}^{(r)} =$$

$$= \sum_{r=0}^{K-1} \omega_r v_{i+1/2}^{(r)} + v(x_{i+1/2}) \sum_{r=0}^{K-1} (\omega_r - d_r) - \sum_{r=0}^{K-1} d_r v_{i+1/2}^{(r)} =$$

$$= \sum_{r=0}^{K-1} (\omega_r - d_r)(v_{i+1/2}^{(r)} - v(x_{i+1/2})) = \sum_{r=0}^{K-1} O(\Delta x^{K-1}) O(\Delta x^K) = O(\Delta x^{2K-1}).$$

Если на каком-то шаблоне S_{r_0} функция $v(x)$ имеет разрыв, то для сохранения свойств ENO схем необходимо коэффициент ω_{r_0} сделать близким к 0. Поэтому весовые коэффициенты вычисляются следующим образом [8]:

$$\omega_r = \frac{\alpha_r}{\sum_{s=0}^{K-1} \alpha_s}, \qquad r = 0, \dots, K-1, \qquad (1.3.2.8)$$

где

$$\alpha_r = \frac{d_r}{(\varepsilon + \beta_r)^2}. \qquad (1.3.2.9)$$

Здесь β_r – так называемый «индикатор гладкости»,

ε – малая положительная величина, введённая, чтобы избежать деления на 0. Далее полагаем $\varepsilon = 10^{-40}$ [7].

На β_r накладываются следующие условия:

$$\beta_r = O(\Delta x^2),$$

если функция $v(x)$ является гладкой на шаблоне S_r, и

$$\beta_r = O(1),$$

если функция $v(x)$ имеет разрыв на шаблоне S_r. Тогда получаем, что

$$\omega_r = O(1),$$

если функция гладкая и

$$\omega_r = O(\Delta x^4),$$

если функция имеет разрыв на шаблоне S_r.

Согласно [8] положим:

$$\beta_r = \sum_{l=0}^{K-1} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Delta x^{2l-1} \left(\frac{\partial^l p_r(x)}{\partial x^l} \right)^2 dx\,, \qquad r = 0, \dots, K-1, \quad (1.3.2.10)$$

где $p_r(x)$ – полином, соответствующий шаблону $S_r, r = 0, \dots, K-1$.

Таким образом, для K = 3 имеем:

$$\beta_0 = \frac{13}{12}(\bar{v}_i - 2\bar{v}_{i+1} + \bar{v}_{i+2})^2 + \frac{1}{4}(3\bar{v}_i - 4\bar{v}_{i+1} + \bar{v}_{i+2})^2,$$
$$\beta_1 = \frac{13}{12}(\bar{v}_{i-1} - 2\bar{v}_i + \bar{v}_{i+1})^2 + \frac{1}{4}(\bar{v}_{i-1} - \bar{v}_{i+1})^2, \qquad (1.3.2.11)$$
$$\beta_2 = \frac{13}{12}(\bar{v}_{i-2} - 2\bar{v}_{i-1} + \bar{v}_i)^2 + \frac{1}{4}(3\bar{v}_{i-2} - 4\bar{v}_{i-1} + 3\bar{v}_i)^2.$$

При этом получаем WENO схему пятого порядка [8].

Положим для дальнейшего исследования K = 3.

1.4 Дискретизация по времени

Для дискретизации по времени для уравнения вида:

$$\frac{\partial U}{\partial t} = L(U), \qquad (1.4.1)$$

где $L(U)$ – пространственный оператор, использовалась TVD–схема Рунге-Кутта третьего порядка [9]:

$$U^* = U^n + \Delta t \cdot L(U^n),$$
$$U^{**} = \frac{3}{4}U^n + \frac{1}{4}U^* + \frac{1}{4}\Delta t \cdot L(U^*), \qquad (1.4.2)$$
$$U^{n+1} = \frac{1}{3}U^n + \frac{2}{3}U^{**} + \Delta t \cdot L(U^{**}).$$

2. Расчёт акустических характеристик

2.1 Анализ полученной модели с помощью пакета Paraview

Описанные выше схемы и подходы позволяют проводить моделирование течения газа около некоего объекта и, таким образом, получать плоскостно-временное распределение газодинамических переменных. Более всего нас будет интересовать давление и его пульсации, именно они и позволяют определить акустическое поле, как в ближней окрестности объекта, так и в любой точке моделируемой области.

Наиболее важными характеристиками акустического поля с точки зрения разработчиков летательных аппаратов являются зависимость общего уровня звукового давления (Overall Sound Pressure Level – OASPL) от направления на точку наблюдения и спектральный состав акустического сигнала в точках наблюдения. Спектральные характеристики сигналов могут быть получены с помощью методов, основанных на дискретном преобразовании Фурье. В данной же работе мы подробно остановимся на расчёте общего уровня звукового давления.

Введём дискретную сетку для времени:

$$\omega_{\Delta_t} = \{\Delta_i, i = 0, \dots, N_t, \Delta_i = t_i - t_{i-1}, |\Delta_i| = \Delta t, \Delta t N_t = T\}, \qquad (2.1.1)$$

здесь $[0,T]$ – промежуток времени, на котором проводится исследование. Отсюда, пульсации давления $p'(R,t)$ в точке R для каждого момента времени находятся по формуле:

$$p'(R,t_i) = p(t_i) - p(t_{i-1}),\ i = 1, \dots N_t, \qquad (2.1.2)$$

Для получения колебаний давления $p(t_i)$ на отрезке времени $[0,T]$ воспользуемся пакетом анализа и визуализации данных Paraview 5.6.0, и, конкретно, инструментом “Plot Selection Over Time”, который позволяет построить график изменения всех используемых переменных в исследуемой точке.

Далее нам необходимо получить численные данные о давлении в исследуемой точке. Для этого, выбрав полученный график, перейдём к виду в форме таблицы (“Spreadsheet view”) и, отметив нужные столбцы (время и давление), экспортируем их в формат *.csv для удобства последующего расчёта в MS Excel.

Аналогичную операцию повторим для некоторого количества точек, равномерно расположенных вокруг исследуемого объекта на равном расстоянии от его центра, чтобы в дальнейшем получить диаграмму направленности общего уровня звукового давления. Увеличение количества точек для рассмотрения сделает диаграмму более подробной, но вместе с этим увеличится и время расчёта необходимых параметров.

2.2 Анализ колебаний давления

Полученный акустический сигнал (последовательность пульсаций давления) для каждой исследуемой точки подвергается дальнейшему анализу для получения интересующих акустических характеристик. Одной из наиболее показательных характеристик является общий уровень пульсаций давления, измеряемый в децибелах. Данный показатель определяется по следующей формуле [12]:

$$OASPL(R) = 10\log_{10}\left(\frac{\langle p'(R)^2\rangle}{p_0^2}\right), \qquad (2.2.1)$$

где $p_0 = 2 \cdot 10^{-5}$ *Па* – минимальный порог слышимости звука,

$\langle p'(R)^2 \rangle$ – среднее значение квадрата колебаний давления в точке R, вычисляемое по формуле:

$$\langle p'(R)^2 \rangle = \frac{\sum_{i=0}^{N_t} p_i'^2}{N_t}. \qquad (2.2.2)$$

С точки зрения инженерных разработок наиболее полезным представлением о картине общего уровня звукового давления является азимутальная диаграмма направленности – график в полярных координатах, на котором отражена зависимость общего уровня звукового давления в некоторой плоскости для некоторого диапазона значений азимутального угла. Как правило, значения угла берутся либо от 0 до π, либо от 0 до 2π. В данной работе исследуется двумерный случай, поэтому выбор плоскости представления ограничен одной.

Для данного исследования вычисление средних значений давления, общего уровня пульсаций давления, а также визуализация полученных данных осуществляется при помощи MS Excel.

Строго говоря, в отличие от аэродинамических характеристик течения, которые довольно точно предсказываются и на малом масштабе, для расчёта шума необходимо использовать расчётную область, размер которой в поперечном направлении совпадает с размером экспериментальной модели. Однако в большинстве случаев это требует очень больших вычислительных ресурсов и в реальности оказывается практически неосуществимым. В связи с этим для расчёта шума, создаваемого при обтекании «квазидвумерных» тел на основе аэродинамических расчётов, выполненных в области исследования размером L, значительно меньшей, чем реальные размеры рассматриваемого в эксперименте тела, предложен ряд специальных поправок.

Простейшая из них базируется на предположении о том, что акустические сигналы от отдельных отрезков тела длиной l, на которые может быть разбито реальное тело, являются некоррелированными между собой. В этом случае суммарная спектральная мощность шума равна сумме спектральных мощностей от отдельных источников длиной l, то есть искомая поправка определяется соотношением

$$\Delta(дБ) = 10\lg(N), \tag{2.2.3}$$

где $\Delta(дБ)$ – величина поправки в децибелах,

$$N = \frac{l_{exp}}{l},$$

где l_{exp} – размер тела в эксперименте [1].

Например, для реального кубического тела со стороной l_{exp} = 10 см, которое в расчётах представлено моделью со стороной l = 0.1 см искомая поправка будет равна

$$\Delta(дБ) = 10\lg(N) = 10\lg\left(\frac{0.1}{0.001}\right) = 20 \text{ дБ}.$$

Как можно понять, это достаточно грубое предположение, которое может привести к существенному занижению шума. Существуют и более точные поправки. Они, тем не менее, в данной работе не описываются из-за относительной трудности расчёта при использовании описанного выше метода анализа.

3. Моделирование акустических полей при обтекании различных тел

3.1 Один кубический блок

Основой для расчётов стала программа, написанная на языке C++, листинг которой приводится в приложении А. В программе задаются начальные и граничные условия, параметры моделируемой области, размер и положение обтекаемых тел, а также условия на их границах. Результаты расчётов выводятся в формате *.vtk и позволяют визуализировать и наблюдать в программном комплексе Paraview все переменные, описанные законами сохранения на протяжении всего времени моделирования.

Практическую часть исследования было решено начать с базового примера обтекания одного блока простой формы. Для моделирования данного процесса и всех следующих были выбраны следующие параметры:

– размеры моделируемой области: 1,78D x 6,5D, где D – некоторый множитель для масштабирования, равный для данной модели 0,05 м. На данной области вводится сетка размерами 89x325.

– газодинамические параметры:

начальное состояние области:

ρ = 1, 20 кг/м^3 – плотность;

(u; v) = (0; 0) м/с – вектор скорости;

p = 101325 Па – давление;

параметры втекающего газа:

ρ = 1, 65 кг/м^3 – плотность;

(u; v) = (0; 114,4) м/с – вектор скорости;

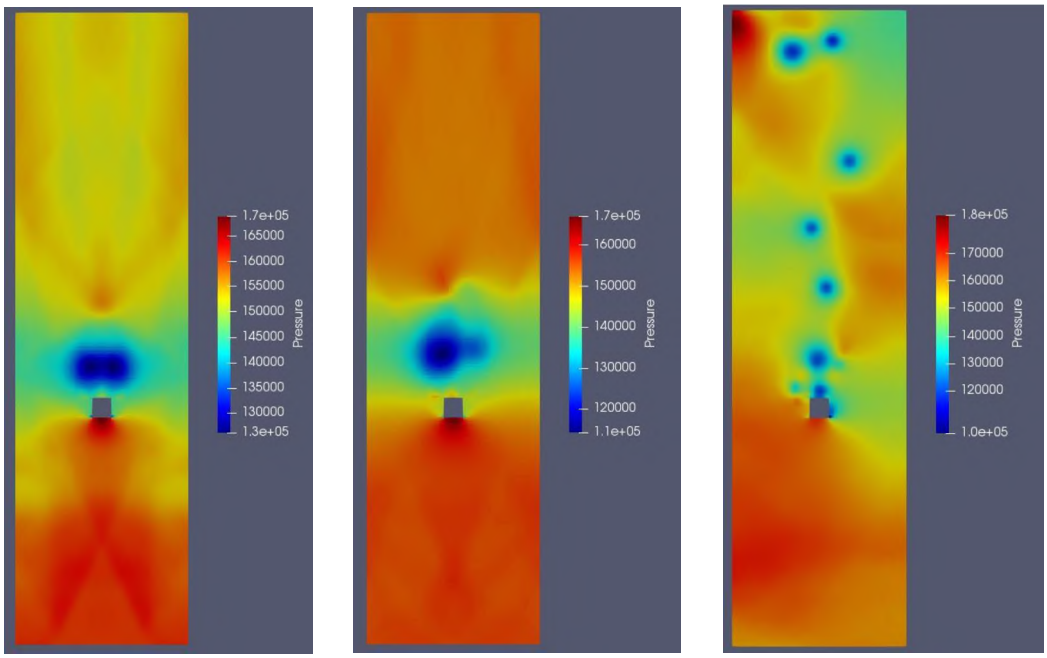
p = 158900 Па – давление;

– шаг по времени: $\tau = 2.0e^{-7}$ с;

– размеры объекта в примере: 10x10 ячеек сетки, что соответствует 0,2D*0,2D (сторона равна 0,01 м).

Так как сравнения с реальным проведённым экспериментом в работе не проводится, то невозможно внести корректную поправку в соответствии с формулой (2.2.3). Поэтому, считая, что размер экспериментального тела соответствует размеру моделируемого, поправку вносить не следует.

По окончании расчётов модель была визуализирована в Paraview. Отдельные этапы протекания процесса показаны на рисунке 2.



а) t = 0,0016 с б) t = 0,0029 с в) t = 0,0058 с

Рисунок 2

В момент t ≈ 0,0029 с, как видно на рисунке 2.б, происходит смещение области пониженного давления, которая до этого момента была симметричной (рисунок 2.а), вследствие превышения критического значения числа Рейнольдса, из-за чего в свою очередь образуется турбулентное течение. Это приводит к образованию так называемой вихревой дорожки (дорожки Кармана), которая видна на рисунке 2.в. Более явно её можно наблюдать на рисунке 3, где показано поле плотности исследуемой области.

Это явление для плохообтекаемых тел возникает при ограниченных значениях числа Рейнольдса и бывает довольно опасным для различных объектов (зданий, сооружений, техники), так как приводит к их разрушению. Поэтому важным моментом создания конструкций является учёт возможного появления вихревых дорожек и принятия мер по предотвращению этого.

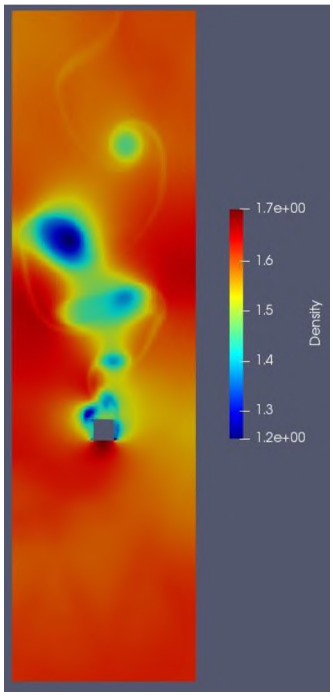


Рисунок 3 – Визуализация вихревой дорожки на графике плотности

Далее, как было описано в главе 2, был использован инструмент “Plot Selection Over Time”, чтобы получить данные о колебаниях давления в точках, окружающих объект.

Опытным путём было выяснено, что для данной модели оптимальным выбором являются 12 точек исследования, расположенных на равном расстоянии друг от друга. Также, для исследования влияния дальности расположения точки от объекта на уровень звуковых колебаний, было проведено несколько вариантов расчётов, в которых точки располагались на расстоянии 0,3D, 0,5D и 0,8D от центра объекта.

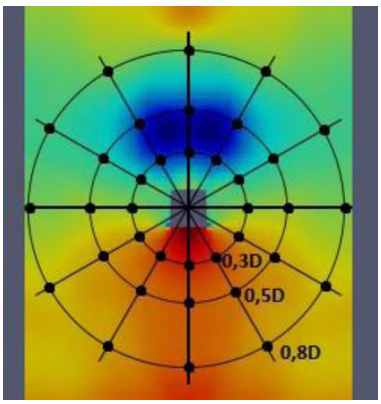


Рисунок 4 – Расположение точек исследования

Полученные данные в численном формате экспортировались в MS Excel, и были проведены расчёты для определения общего уровня звуковых пульсаций давления. В итоге получены конечные данные, представленные в таблице 2, а также в виде азимутальной диаграммы направленности на рисунке 5.

Т а б л и ц а 2 – Зависимость OASPL
от направления на точку и расстояния
до неё

Азимут	Расстояние до центра объекта		
	0.3D	0.5D	0.8D
180	119,8097	127,9251	130,1272
150	129,8959	130,6925	131,4455
120	130,5572	130,0003	131,741
90	130,9154	130,6243	130,4161
60	131,6899	130,8593	130,2191
30	132,451	131,7413	131,2284
0	132,5678	132,2297	131,8027
330	131,7382	132,494	132,449
300	130,8002	132,5274	132,391
270	130,147	131,2535	131,2209
240	130,191	130,6873	130,7772
210	129,4627	129,5303	130,8049

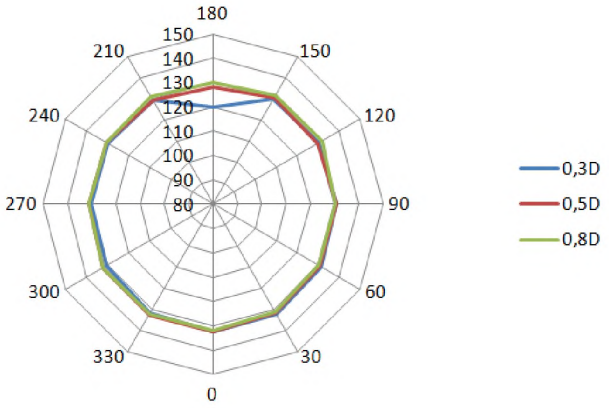


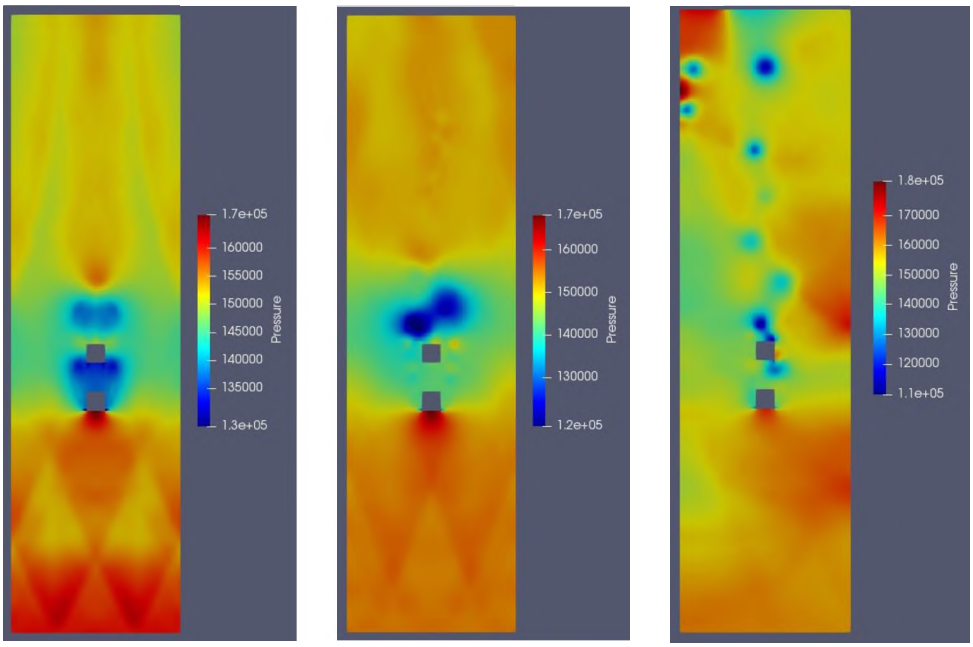
Рисунок 5 – Азимутальная диаграмма направленности общего уровня звукового давления

По приведённым таблице и диаграмме можно сделать вывод, что исследование на близком расстоянии более информативно, так как оно показывает уменьшение общего уровня шума за объектом, как следствие появления там области пониженного давления без последующих колебаний (до образования завихрений). Для близкого поля течения газа около объекта следует использовать более подробные математические модели, что и было проделано в данной работе, так как для этой области существенны нелинейные эффекты и могут образовываться скачки, где важна динамика вихреобразования, учет турбулентности и т.п. В свою очередь в дальнем поле, где подразумевается однородное течение вдали от источника шума, перенос акустических возмущений можно описывать волновым уравнением.

3.2 Каскад из двух кубических блоков

Настоящий пункт практической части работы посвящён анализу тех же параметров, что и предыдущий, для каскада из двух объектов размером $0{,}2D$ x $0{,}2D$, находящихся на расстоянии $0{,}3D$ друг от друга.

На рисунке 5 приведены отдельные моменты визуализации процесса.



а) t = 0.0016 с б) t = 0.0029 с в) t = 0.0058 с

Рисунок 6

На начальном этапе две области пониженного давления симметричны относительно продольной оси пары объектов, периодически объединяются в одну. Однако за вторым объектом область гораздо меньше в силу того, что он находится в области действия так называемого «кармана» за первым объектом, следовательно, испытывает меньшие нагрузки со стороны среды. Подобный эффект «кармана» часто используется автогонщиками на трассах и обычными автомобилистами, когда они пристраиваются за едущим впереди автомобилем. Это позволяет поддерживать скорость и экономить топливо за счёт того, что давление впереди автомобиля оказывается ниже, чем сзади.

Спустя время, как и в случае с одним объектом, наблюдались вихревые дорожки Кармана (подробнее показаны на рисунке 7). В данном случае оба объекта будут под угрозой, так как вихри, срываясь с первого, наталкиваются на второй, который в свою очередь также порождает вихревые потоки.

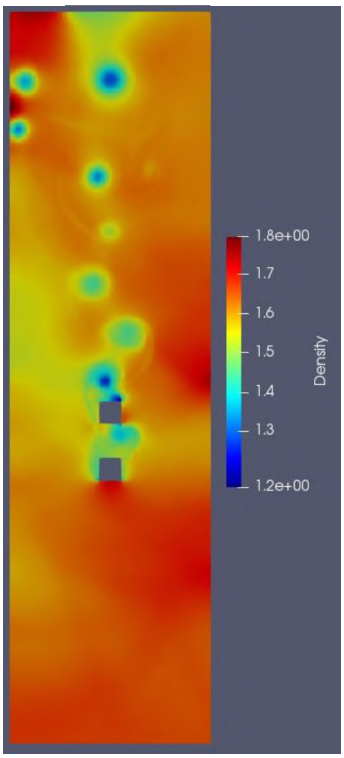


Рисунок 7– Визуализация вихревой дорожки на графике плотности

Для расчёта акустического поля целесообразно использовать не окружность, а вытянутую фигуру, чтобы захватить оба объекта. Например, можно использовать эллипс, фокусами которого являются центры объектов. Количество точек исследования было увеличено до 16.

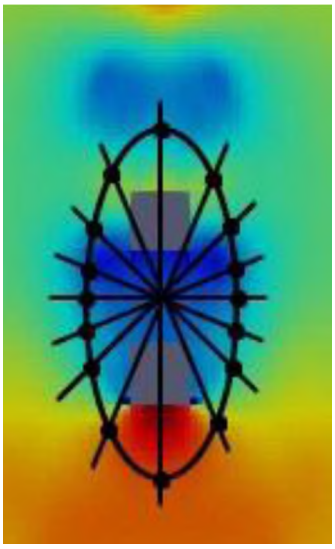


Рисунок 8 – Расположение точек исследования

На рисунке 9 представлена полученная для каскада кубов диаграмма направленности общего уровня шума (OASPL).

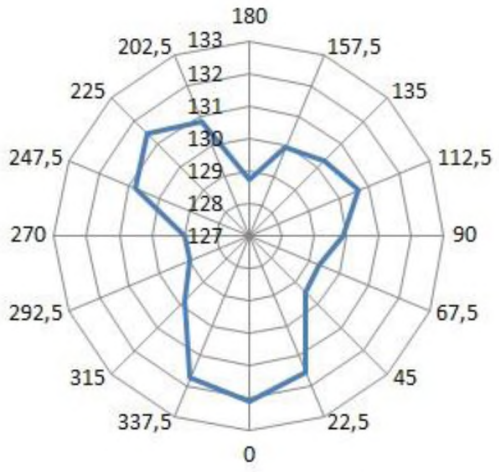


Рисунок 9

Как можно судить по диаграмме, минимальный уровень звуковых колебаний наблюдается сразу за вторым объектом, а также ближе к центру пространства между объектами, что обосновано наличием там областей низкого давления на протяжении всего времени. Заметим, что диаграмма не совсем симметрична: связано это с тем, что образование вихревых дорожек происходит преимущественно на правой стороне из-за чего в одних и тех же областях наблюдается как высокое, так и низкое давление, что в свою очередь делает картину менее информативной на больших промежутках времени, сглаживая средние значения. С левой же стороны ситуация более стабильна, а потому об уровне шума можно судить даже по расчётам приведённым выше.

Отдельный интерес представляет сравнение графиков изменения давления за каждым из объектов, которое представлено на рисунке 10. По этим графикам можно видеть, как в самом начале временного отрезка ударная волна гасится первым кубом и оказывает на второй меньшее давление. Далее видно, как до момента (t = 0,0003 с) давление в обеих точках

постепенно падает с незначительными колебаниями. После того, как происходит смещение области пониженного давления и образуются вихревые потоки, колебания давления за вторым блоком становятся сильнее, что обусловлено срывами с этого блока сильных вихрей. Говоря об этих процессах в терминах акустики, ударная волна повлечёт за собой повышение уровня шума до пика, далее он будет спадать в обеих областях. После появления завихрений колебания снова начнут вызывать повышение уровня шума, но на данном отрезке времени между блоками это будет заметно меньше, так как там сохраняется область низкого давления. Тем не менее, уровень шума незначительно повысится и в этой области, что и показывает увеличение колебаний в конце временного отрезка.

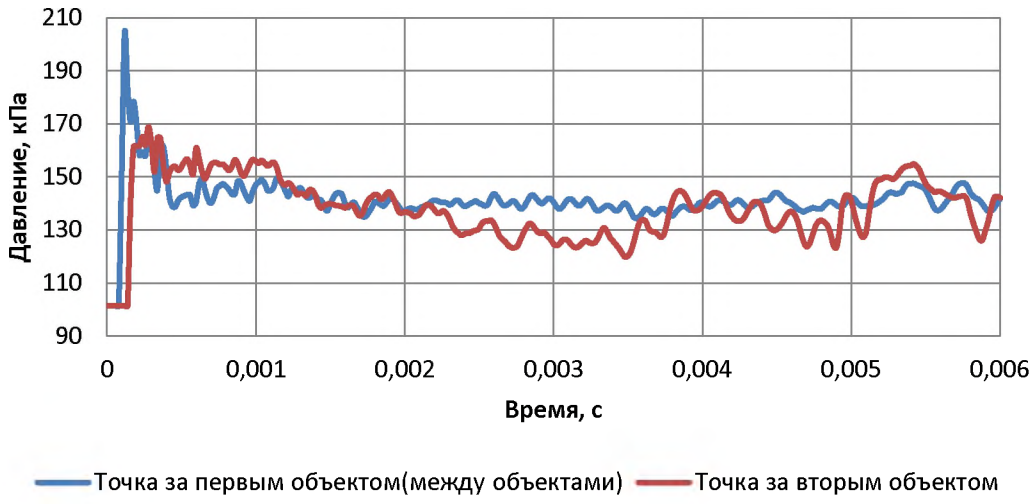


Рисунок 10 – Графики колебания давления

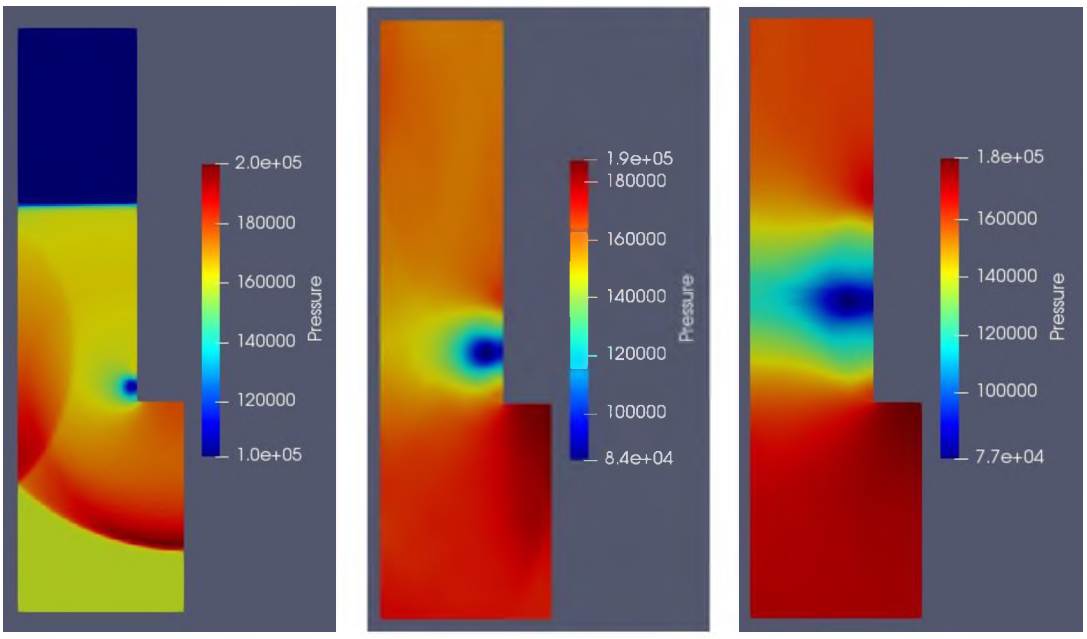
Расчёт общего уровня звуковых колебаний для точки между объектами дал значение 128,6 дБ – такая незначительная разница со значением для точки за вторым объектом (128,7 дБ), несмотря на меньшие колебания, обусловлен большим пиком в начале.

3.3 Ступенька

Моделирование обтекания прямой ступеньки, как и расчёт создаваемого при этом акустического поля, является одной из важных задач аэродинамики. В том или ином виде эта базовая задача встречается во многих реальных проектах.

Высота подъёма ступеньки, которая была выбран для модели, равен $0{,}5D$. Подъём находится на расстоянии $2{,}4D$ от границы, через которую втекает газ. Обтекание подобных тел вызывает колебания давления и звуковые волны, как в ближнем, так и в дальнем поле, но в данной работе акцент сделан именно на моделирование ближнего поля.

На рисунке 11 представлены этапы визуализации модели в программном комплексе Paraview. Можно наблюдать, как образуется отрыв потока и область пониженного давления, которая в дальнейшем увеличивается и сдвигается от края ступеньки.



а) t = 0.0003 с б) t = 0.002 с в) t = 0.006 с

Рисунок 11

Точки исследования были установлены, как показано на рисунке 12. Это расположение обусловлено желанием рассчитать общий уровень шума в непосредственной близости к ступеньке.

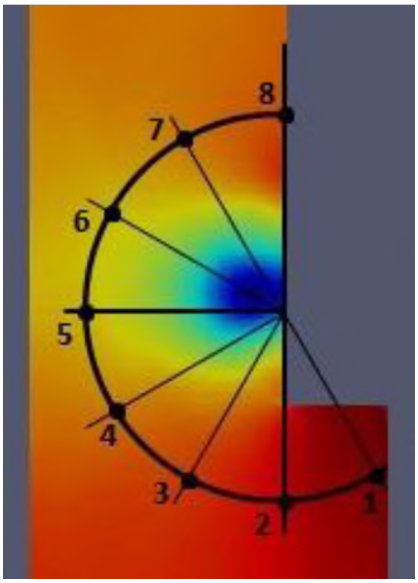


Рисунок 12 – Расположение точек исследования

На рисунке 13 представлен график зависимости общего уровня звукового давления от положения точки.

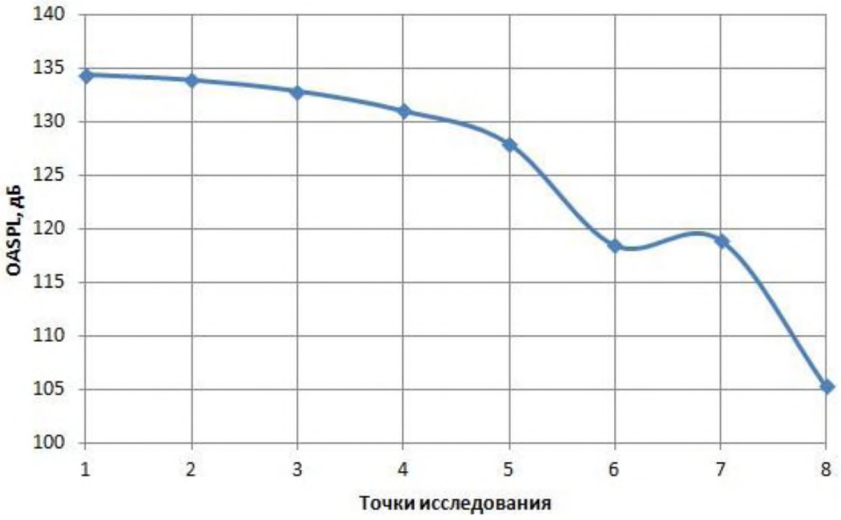


Рисунок 13 – График зависимости общего уровня звукового давления от положения точки

Ударная волна, пришедшаяся на ступеньку, вызвала колебания давления, которое привело к большому уровню звукового давления. Низкий уровень шума в точке 8 объясняется тем, что возникшая на подъёме ступеньки область низкого давления постепенно увеличивалась и медленно двигалась в направлении потока. Также при контакте потока со ступенькой происходит отрыв, из-за которого основная часть потока проходит над поверхностью ступеньки, не касаясь её и не вызывая колебаний давления.

Продемонстрировать разницу уровней звукового давления до ступеньки, на подъёме и после него позволит расчёт акустического поля на поверхности правой стенки, а также на подъёме ступеньки. Схема расположения точек показана на рисунке 14.

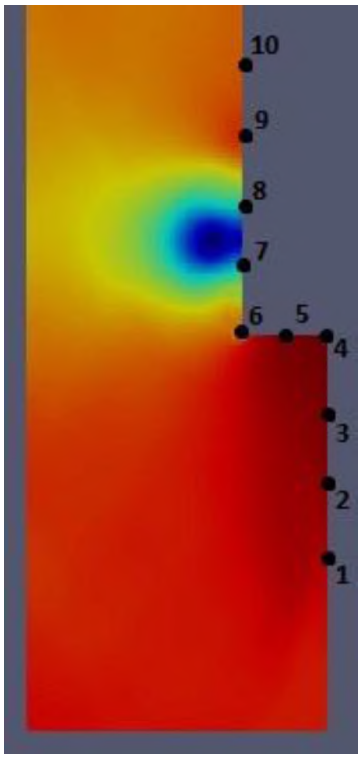


Рисунок 14 – Расположение точек исследования

В результате был получен график зависимости общего уровня звукового давления от расположения точки (рисунок 15), а также графики колебания давления в точках 1, 4, 6, 7 и 10 (Рисунок 16), чтобы подробнее разобрать происходящие в этих областях процессы.

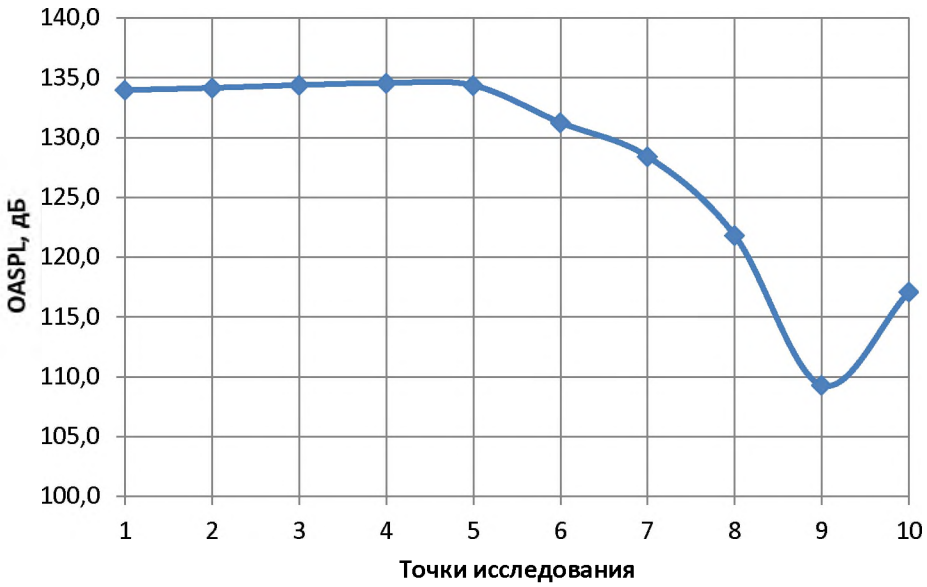


Рисунок 15 – График зависимости общего уровня звукового давления от положения точки

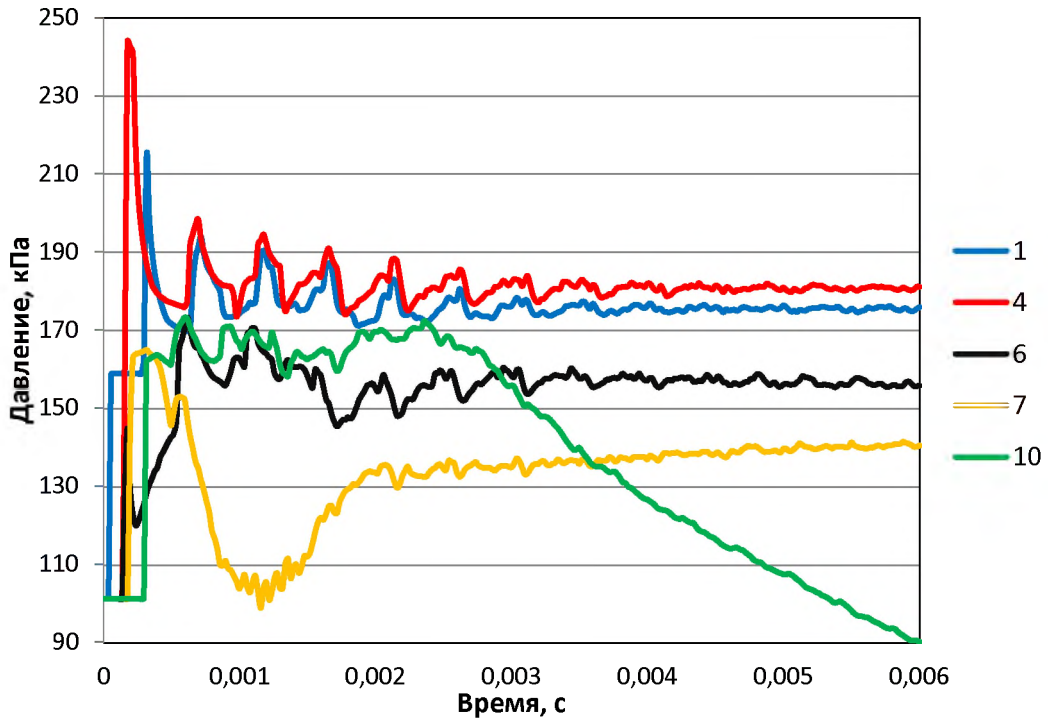


Рисунок 16 – Графики колебания давления

Видно, что общий уровень звукового давления согласуется с графиком, полученным для ближнего поля около ступеньки, то есть он снижается по направлению потока после подъёма. Довольно резкое возрастание уровня

шума в точке 10 показывает, что за моделируемый отрезок времени в данной точке область высокого давления сменилась зоной низкого давления, в то время как точки 7, 8, 9 находились в более стабильной части, где давление в основном понижалось.

На рисунке 16 можно увидеть некоторые закономерности поведения давления в разных областях ступеньки. Области до подъёма ступеньки и после него характеризуются своего рода семействами графиков: можно увидеть, насколько похожи графики разных краёв одной зоны (1 и 4 и 7 и 10). Разница графиков заключается в следующем:

– до подъёма: в общем уровне и пиковом значении давления, форма колебаний же практически одинакова. Чем ближе точка к подъёму, тем выше в ней уровень давления. Это и объясняет постепенный пологий подъём общего уровня звуковых колебаний на рисунке 15.

– после подъёма: в размере так называемого «провала». Влияние на график в данной области оказывает расширяющаяся и двигающаяся область низкого давления. Чем дальше точка от края ступеньки, тем шире будет «провал». Точка 6 находится на выступе подъёма, именно там зарождается область низкого давления – это объясняет небольшой подъём и резкий «провал» графика в этой точке. Далее давление здесь возрастает и довольно сильно колеблется, что становится причиной высокого уровня акустического излучения. До точки 7 область низкого давления доходит спустя время и задерживается дольше, вызывая похожие эффекты. Точка 10 характеризуется начальным колебанием давления, но спустя время давление там лишь снижается, и этот процесс продолжается до завершения моделирования.

3.4 Каверна (выемка)

Нестационарное пульсирующее течение, возникающее при обтекании плоской каверны параллельным ей набегающим потоком интересно как с точки зрения аэродинамических нагрузок, так и сильного звукового излучения. Несмотря на простую геометрию области, в ней имеет место довольно сложное течение, включающее турбулентные слои, возвратно-циркуляционную зону, вихреобразование. Практическая ценность исследования этих процессов, заключается в том, что такую структуру, как каверна, можно использовать как средство управления сложными течениями, например в качестве стабилизатора пламени [2].

Размеры моделируемой каверны составляют 0,5D x 1D, её левый край находится на расстоянии 2,4D от границы, через которую втекает газ. В ряде изученных при подготовке работ рассматривается влияние размеров и формы каверны на создаваемое излучение, и оно, безоговорочно, существенно, однако было решено остановиться на простом примере прямоугольной каверны. Также существует ряд статей, в которых подробно описано, как пульсирующие потоки газа создают акустическое излучение, распространяющееся в дальнее поле, отражаясь от стенок каверны. В данной же работе будет рассмотрено колебание давления в ближнем поле каверны, а также внутри неё.

На рисунке 17 представлен фрагмент визуализации модели. Рассмотрим его подробнее, чтобы понять, какие процессы здесь имеют место. На данном фрагменте поток движется слева направо. Внутри каверны можно наблюдать систему волн давления. Одна зона повышенного давления формируется у задней (правой) стенки, другая приближается к передней (левой). Между ними можно видеть две небольшие зоны пониженного давления и одну зону большего размера, которая опустилась в каверну вследствие разбиения начального вихря, образованного ударной волной, на

две: вторая часть вихря преодолела каверну. Сразу же за каверной наблюдается зона пониженного давления, вызванная расширением газа при преодолении им задней стенки. Чуть дальше по направлению потока можно заметить следы предыдущего распавшегося вихря [2].

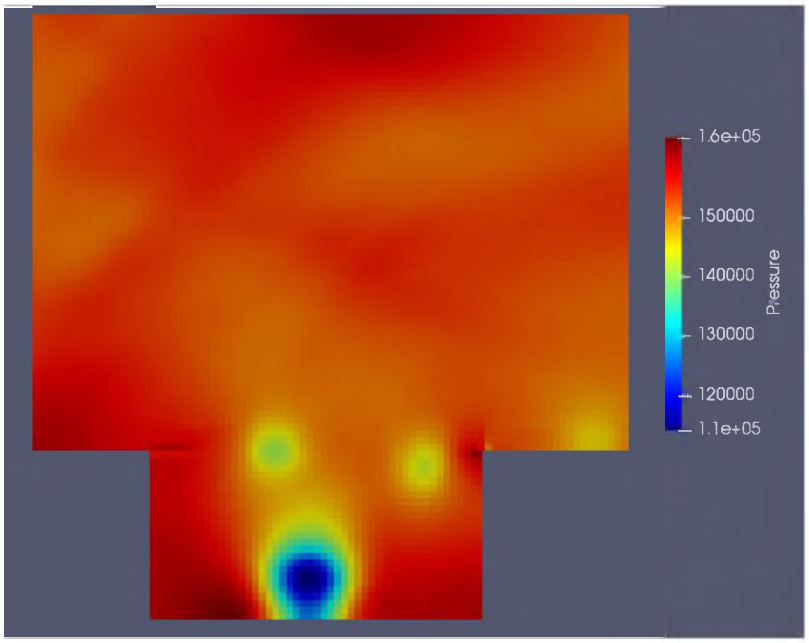
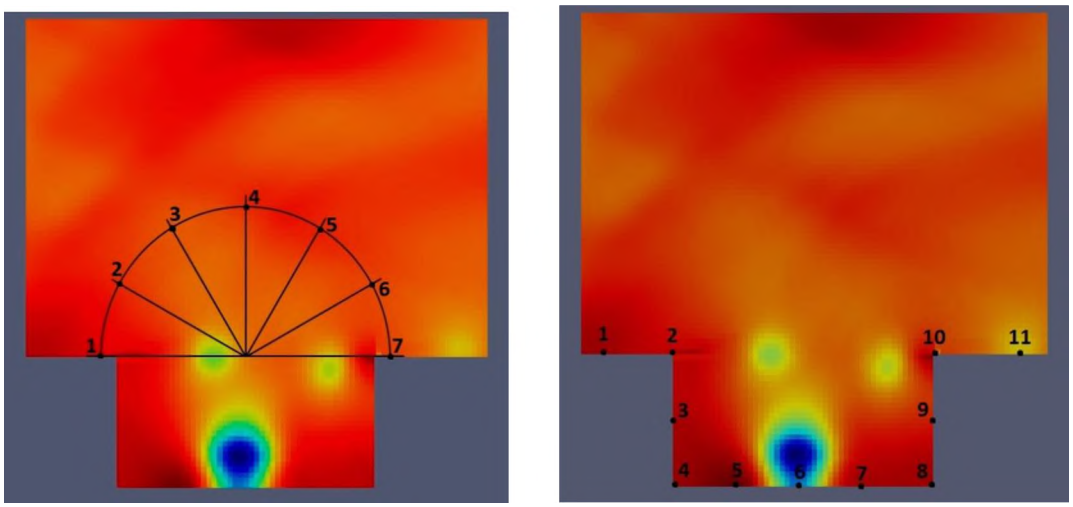


Рисунок 17

Как и в случае со ступенькой, для каверны будут проведены расчёты в ближнем поле над ней и на поверхности: до передней стенки, на ней, на дне, на задней стенке и после неё. Расположения точек исследования в обоих случаях показаны на рисунке 18.

Зона пониженного давления, находящаяся в глубине каверны, остаётся там на протяжении всего времени моделирования, вращаясь по часовой стрелке, что иллюстрирует создаваемые в каверне циркуляционные потоки.



а) б)

Рисунок 18 – Расположение точек исследования

На полученном графике (рисунок 19) для точек над каверной видно, что общий уровень звукового давления в различных точках ближнего поля отличается не более чем на 0,7 дБ, что является пренебрежимо малой величиной по сравнению с абсолютными значениями. Следовательно, можно считать, что в ближнем поле наблюдается однородное постоянное звуковое излучение, распространяющееся в дальнее поле.

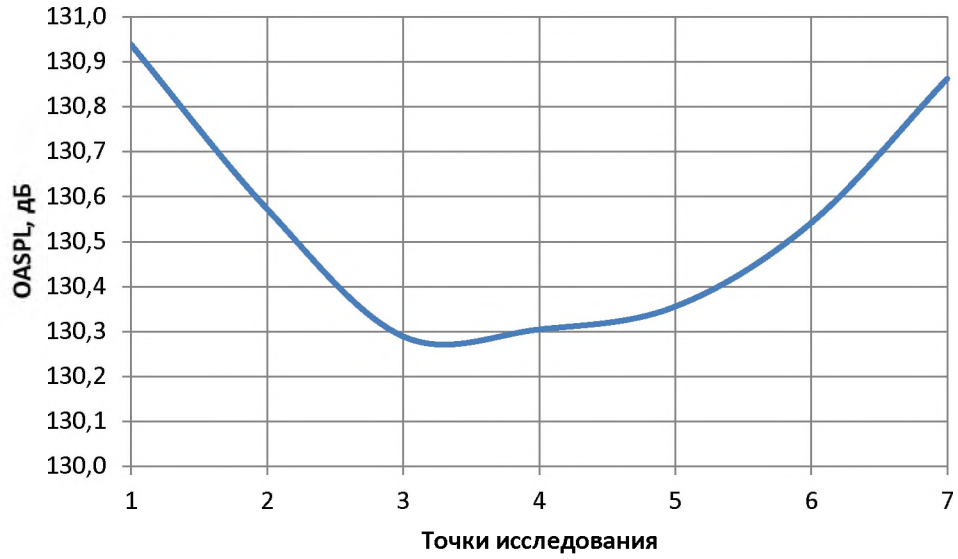


Рисунок 19 – График зависимости общего уровня звукового давления от положения точки

Фактически же акустические волны, исходящие от каверны, условно делятся на 2 группы: первая распространяется от левой кромки и генерируется потоком давления внутри каверны, вторая – от задней (правой) кромки и возникает из-за взаимодействия вихревых потоков с этой кромкой. Разница между этими группами практически не заметна (на рисунке 17 можно обратить внимание на две пересекающиеся дуги волн, отходящих от кромок) на выбранном масштабе и области исследования, так как они сливаются в одну волну, уравниваясь при расчёте общего уровня звукового давления.

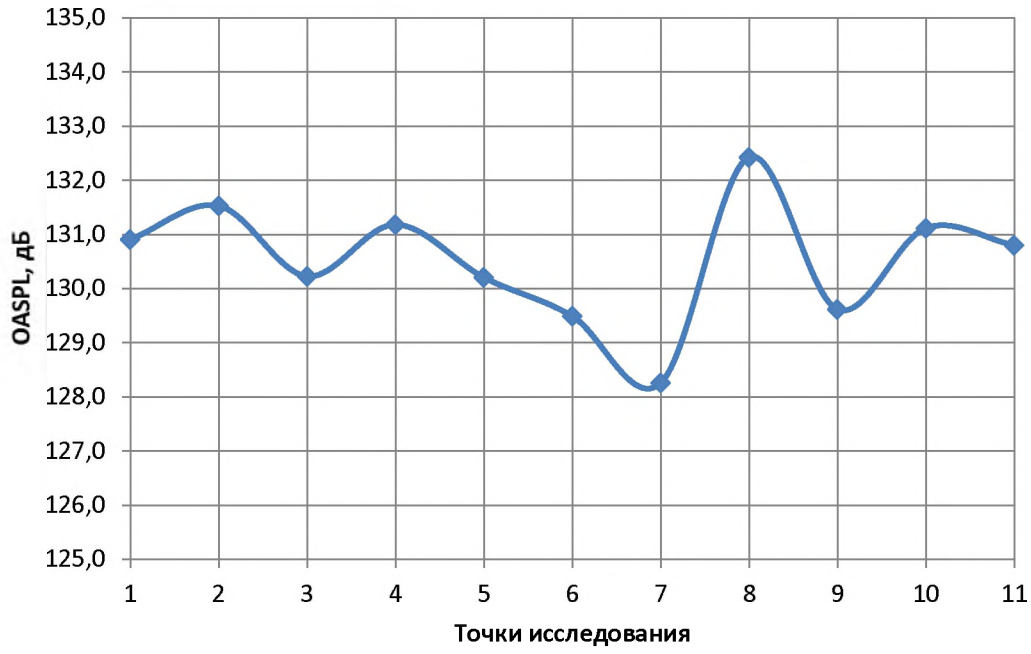


Рисунок 20 – График зависимости общего уровня звукового давления от положения точки

На рисунке 20, который показывает зависимость общего уровня звукового давления от положения точек на поверхности каверны, видно, как уровень звукового давления слегка поднимается, подходя к кромкам каверны. В промежутке между точками 2 и 10 можно наблюдать влияние на общий уровень звукового давления нелинейных эффектов, являющихся

следствием раздваивающегося потока и циркуляционного движения внутри каверны. Самый высокий показатель звукового давления приходится на точку 8, так как именно в углу между правой стенкой и дном образуется зона повышенного давления из-за части потока идущей внутрь каверны.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана программа на языке C++, позволяющая моделировать обтекание различных тел потоком газа на основе законов сохранения. Использовались схемы реконструкции газодинамических параметров и расчета потоков высокой точности. Преимуществом созданной программы является возможность простого изменения параметров течения газа, параметров среды, размера и положения обтекаемых объектов. Был исследован способ моделирования акустического поля, создаваемого при обтекании объекта, при помощи расчёта общего уровня звукового давления(OASPL).

Было проведено моделирование процесса обтекания и расчёт акустического поля для различных тел и объектов: куба, каскада из двух кубов, ступеньки, каверны. Для каждого объекта был дан анализ происходящих процессов на основании визуализации и графического представления акустического поля. Для куба было проведено исследование влияния удаления точки наблюдения акустических эффектов в пределах исследуемой области, которое показало, что изучение ближнего поля в приведённой области исследования наиболее информативно.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гарбарук А. В. Расчет аэродинамики и шума при обтекании тандема цилиндров / А. В. Гарбарук, Ф. Р. Спаларт, М. Х. Стрелец, М. Л. Шур // Математическое моделирование. – 2014. – Т. 26, № 6. – С. 119–136.

2. Савельев А. Д. Численное моделирование акустического излучения двумерной каверны в дозвуковом потоке / А. Д. Савельев // Ученые записки ЦАГИ. – 2014. –Т. 45, № 1. – С. 57–74.

3. Самарский А. А. Численные методы : учеб. пособие для вузов / А. А. Самарский, А. В. Гулин. М. : Наука, 1989. – 342 с.

4. Самарский А. А. Разностные методы решения задач газовой динамики : учеб. пособие для вузов / А. А. Самарский, Ю. П. Попов. М. : Наука, 1980. – 424 с.

5. Тишкин В. Ф. Разностные схемы трехмерной газовой динамики для задачи о развитии неустойчивости Рихтмаера-Мешкова / В. Ф. Тишкин, В. В. Никишин, И. В. Попов, А. П. Фаворский // Математическое моделирование. – 1995. – Т. 7, № 5. – С. 15–25.

6. Courant R. On the solution of nonlinear hyperbolic differential equations by finite differences / R. Courant, E. Isaacson, M. Rees // Communications on Pure Applied Mathematics. – 1952. – Vol. 5, № 3. – P. 243–255.

7. Henrik A. K. Mapped weighted essentially non-oscillatory schemes: Achieving optimal order near critical points / A. K. Henrik, T. D. Aslam, J. M. Powers // Journal of Computational Physics. – 2005. – Vol. 207, № 2. – P. 542–567.

8. Jiang G. -S. Efficient implementation of weighted ENO schemes / G. -S. Jiang, C. -W. Shu // Journal of computational physics. – 1996. – Vol. 126, № 1. – P. 202–228.

9. Shu C. -W. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws / C.-W. Shu // ICASE Report 97-65. – 1997. – 84 p.

10. Toro E. F. Riemann Solvers and Numerical Methods for Fluid Dynamics : A Practical Introduction / E. F.Toro. – 3rd edn. – Heidelberg : Springer-Verlag Berlin Heidelberg, 2009. – 724 p.

11. Toro E. F. The HLLC Riemann solver / E. F. Toro // Shock Waves. – 2019. – №29. – P. 1065–1082.

12. Wijker J. J. Spacecraft Structures / J. J. Wijker. – 1st edn. – Heidelberg : Springer-Verlag Berlin Heidelberg, 2008. – 504 p.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <math.h>

double const M_PI = 3.14159265358979323846;

const double    D = 50.e-3;
const int          STEP_OUT = 100;
const int          NX = 89;
const int          NY = 325;
const double XMIN = 0.0;
const double XMAX = 1.78 * D;
const double YMIN = 0.0;
const double YMAX = 6.5 * D;
const double HX = (XMAX - XMIN) / NX;
const double HY = (YMAX - YMIN) / NY;
const double TMAX = 4.5e0;
const double TAU = 2.0e-7;
const int    K_WENO = 3;

const double    HC_X = 1.78 * D / 2.;
const double    HC_Y = 3. * D;
const double    H_R2 = D * D / 4.;
const double    SH_Y = 1. * D;

const double    GAS_R = 8.31446261815324;

const int       COMP_COUNT = 2;
const double    COMP_CP[COMP_COUNT] = { 1006., 1006. };
const double    COMP_M[COMP_COUNT] = { 28.98e-3, 28.98e-3 };


const double R0 = 1.65, R1 = 1.20, R2 = 0.166;
const double U0 = 0.0, U1 = 0.0, U2 = 0.0;
const double V0 = 114.4, V1 = 0.0, V2 = 0.0;
const double P0 = 1.589e5, P1 = 1.01325e5, P2 = 1.01325e5;
const double C0[COMP_COUNT] = { 0., 1. };
const double C1[COMP_COUNT] = { 0., 1. };
const double C2[COMP_COUNT] = { 1., 0. };


double** c[COMP_COUNT];
double** r;
double** u;
double** v;
double** p;
double** E;
double** e;

double** ru;
double** rv;
```

```
double** re;
double** rc[COMP_COUNT];

double** ru_old;
double** rv_old;
double** re_old;
double** rc_old[COMP_COUNT];

void  draw(char* fName, int nx, int ny);
double KR(double x);
double _max_(double x, double y);

const int lo[2] = { K_WENO, K_WENO };
const int hi[2] = { K_WENO + NX - 1, K_WENO + NY - 1 };
const int NX_ALL = NX + K_WENO + K_WENO;
const int NY_ALL = NY + K_WENO + K_WENO;

char str[1024];

double calc_gam(int i, int j)
{
	double cp_mix = 0.;
	double M_ = 0.;
	for (int ic = 0; ic < COMP_COUNT; ic++) {
		M_ += c[ic][i][j] / COMP_M[ic];
		cp_mix += c[ic][i][j] * COMP_CP[ic];
	}
	double M_mix = 1. / M_;
	double cv_mix = cp_mix - GAS_R / M_mix;
	return cp_mix / cv_mix;
}


void init()
{
	for (int ic = 0; ic < COMP_COUNT; ic++) c[ic] = new double* [NX_ALL];
	r = new double* [NX_ALL];
	u = new double* [NX_ALL];
	v = new double* [NX_ALL];
	p = new double* [NX_ALL];
	E = new double* [NX_ALL];
	e = new double* [NX_ALL];

	ru = new double* [NX_ALL];
	rv = new double* [NX_ALL];
	re = new double* [NX_ALL];
	for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic] = new double* [NX_ALL];

	ru_old = new double* [NX_ALL];
	rv_old = new double* [NX_ALL];
	re_old = new double* [NX_ALL];
	for (int ic = 0; ic < COMP_COUNT; ic++) rc_old[ic] = new double* [NX_ALL];

	for (int i = 0; i < NX_ALL; i++)
	{
		for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i] = new double[NY_ALL];
		r[i] = new double[NY_ALL];
		u[i] = new double[NY_ALL];
		v[i] = new double[NY_ALL];
```

```
            p[i] = new double[NY_ALL];
            E[i] = new double[NY_ALL];
            e[i] = new double[NY_ALL];

            ru[i] = new double[NY_ALL];
            rv[i] = new double[NY_ALL];
            re[i] = new double[NY_ALL];
            for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i] = new double[NY_ALL];

            ru_old[i] = new double[NY_ALL];
            rv_old[i] = new double[NY_ALL];
            re_old[i] = new double[NY_ALL];
            for (int ic = 0; ic < COMP_COUNT; ic++) rc_old[ic][i] = new
double[NY_ALL];
      }



      for (int i = lo[0]; i <= hi[0]; i++)
      {
            double x = XMIN + (i - lo[0]) * HX;
            for (int j = lo[1]; j <= hi[1]; j++)
            {
                  double y = YMIN + (j - lo[1]) * HY;
                  if (y < SH_Y)
                  {
                        r[i][j] = R0; //post shock
                        u[i][j] = U0;
                        v[i][j] = V0;
                        p[i][j] = P0;
                        for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j] =
C0[ic];
                  }
                  else {
                        r[i][j] = R1; // pre shock
                        u[i][j] = U1;
                        v[i][j] = V1;
                        p[i][j] = P1;
                        for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j] =
C1[ic];
                  }
                  double gam = calc_gam(i, j);
                  e[i][j] = p[i][j] / r[i][j] / (gam - 1.0);
                  E[i][j] = e[i][j] + (u[i][j] * u[i][j] + v[i][j] * v[i][j]) /
2.0;

                  ru[i][j] = r[i][j] * u[i][j];
                  rv[i][j] = r[i][j] * v[i][j];
                  re[i][j] = r[i][j] * E[i][j];
                  for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] = r[i][j] *
c[ic][i][j];
            }
      }
}


void bnd_cond()
{
```

```
	for (int j = lo[1]; j <= hi[1]; j++)//по Y(left and right borders)
	{
		for (int i = 0; i < K_WENO; i++)
		{
			int i_ex = lo[0] - i - 1;
			int i_in = lo[0] + i;

			r[i_ex][j] = r[i_in][j];
			for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i_ex][j] =
c[ic][i_in][j];
			u[i_ex][j] = -u[i_in][j];
			v[i_ex][j] = v[i_in][j];
			p[i_ex][j] = p[i_in][j];
			e[i_ex][j] = p[i_ex][j] / r[i_ex][j] / (calc_gam(i_in, j) - 1.0);
			E[i_ex][j] = e[i_ex][j] + (u[i_ex][j] * u[i_ex][j] + v[i_ex][j] *
v[i_ex][j]) * 0.5;

			for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i_ex][j] =
r[i_ex][j] * c[ic][i_ex][j];
			ru[i_ex][j] = r[i_ex][j] * u[i_ex][j];
			rv[i_ex][j] = r[i_ex][j] * v[i_ex][j];
			re[i_ex][j] = r[i_ex][j] * E[i_ex][j];

			i_ex = hi[0] + i + 1;
			i_in = hi[0] - i;

			r[i_ex][j] = r[i_in][j];
			for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i_ex][j] =
c[ic][i_in][j];
			u[i_ex][j] = -u[i_in][j];
			v[i_ex][j] = v[i_in][j];
			p[i_ex][j] = p[i_in][j];
			e[i_ex][j] = p[i_ex][j] / r[i_ex][j] / (calc_gam(i_in, j) - 1.0);
			E[i_ex][j] = e[i_ex][j] + (u[i_ex][j] * u[i_ex][j] + v[i_ex][j] *
v[i_ex][j]) * 0.5;

			for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i_ex][j] =
r[i_ex][j] * c[ic][i_ex][j];
			ru[i_ex][j] = r[i_ex][j] * u[i_ex][j];
			rv[i_ex][j] = r[i_ex][j] * v[i_ex][j];
			re[i_ex][j] = r[i_ex][j] * E[i_ex][j];
		}
	}
	for (int i = lo[0]; i <= hi[0]; i++)//По X(top and bottom borders)
	{
		for (int j = 0; j < K_WENO; j++)
		{
			int j_ex = lo[1] - j - 1;
			int j_in = lo[1] + j;
			//Реализовано постоянное втекание(на нижней границе всегда
постоянное значение POSTshock)
			//и вытекание (на верхней гранце скорость газа не отражается)
			r[i][j_ex] = R0;
			for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j_ex] = C0[ic];
			u[i][j_ex] = U0;
			v[i][j_ex] = V0;
			p[i][j_ex] = P0;
			e[i][j_ex] = p[i][j_ex] / r[i][j_ex] / (calc_gam(i, j_in) - 1.0);
```

```
                    E[i][j_ex] = e[i][j_ex] + (u[i][j_ex] * u[i][j_ex] + v[i][j_ex] *
v[i][j_ex]) * 0.5;

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j_ex] =
r[i][j_ex] * c[ic][i][j_ex];
                    ru[i][j_ex] = r[i][j_ex] * u[i][j_ex];
                    rv[i][j_ex] = r[i][j_ex] * v[i][j_ex];
                    re[i][j_ex] = r[i][j_ex] * E[i][j_ex];

                    j_ex = hi[1] + j + 1;
                    j_in = hi[1] - j;

                    r[i][j_ex] = r[i][j_in];
                    for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j_ex] =
c[ic][i][j_in];
                    u[i][j_ex] = u[i][j_in];
                    v[i][j_ex] = v[i][j_in];
                    p[i][j_ex] = p[i][j_in];
                    e[i][j_ex] = p[i][j_ex] / r[i][j_ex] / (calc_gam(i, j_in) - 1.0);
                    E[i][j_ex] = e[i][j_ex] + (u[i][j_ex] * u[i][j_ex] + v[i][j_ex] *
v[i][j_ex]) * 0.5;

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j_ex] =
r[i][j_ex] * c[ic][i][j_ex];
                    ru[i][j_ex] = r[i][j_ex] * u[i][j_ex];
                    rv[i][j_ex] = r[i][j_ex] * v[i][j_ex];
                    re[i][j_ex] = r[i][j_ex] * E[i][j_ex];
            }
      }
}


bool trash(int x_point, int y_point, int x_size, int y_size, int i, int j) {
      bool res;
      ((i >= x_point) && (i <= x_point + x_size) && (j >= y_point) && (j <= y_point
+ y_size)) ?
            res = true : res = false;
      return res;
}

//Отступ от левой и правой границы
int BackStep = 40;

/*
//каверна============================================================================
const int x_point1 = lo[0] + 65;
const int x_point2 = hi[0];
const int y_point1 = 160;
//размер объекта
const int x_size1 = x_point2 - x_point1;//длина
const int y_size1 = hi[1]-y_point1;//ширина

//вторая сторона каверны
const int x_point1_2 = lo[0] + 65;
const int x_point2_2 = hi[0];
const int y_point1_2 = lo[1];
//размер объекта
const int x_size1_2 = x_point2_2 - x_point1_2;//длина
const int y_size1_2 = 120-y_point1_2;//ширина
```

```
//=====================================================================
*/

//ступенька============================================================
const int x_point1 = lo[0] + 65;
const int x_point2 = hi[0];
const int y_point1 = 120;
//размер объекта
const int x_size1 = x_point2 - x_point1;//длина
const int y_size1 = hi[1]-y_point1;//ширина
//=====================================================================

//два куба=============================================================
//точка расположения объекта(нижнего левого угла)
//const int x_point1 = lo[0] + BackStep;
//const int x_point2 = hi[0] - BackStep;
//const int y_point1 = 120;
//размер объекта
//const int x_size1 = x_point2 - x_point1;//длина
//const int y_size1 = x_point2 - x_point1;//ширина

//точка расположения объекта(нижнего левого угла)
//const int x_point1_2 = lo[0] + BackStep;
//const int x_point2_2 = hi[0] - BackStep;
//const int y_point1_2 = 145;
//размер объекта
//const int x_size1_2 = x_point2_2 - x_point1_2;//длина
//const int y_size1_2 = x_point2_2 - x_point1_2;//ширина
//=====================================================================

void lr_bnd_cond_cube(int x_point, int y_point, int x_size, int y_size) {
	for (int j = y_point; j <= y_point + y_size; j++)//по Y(left and right
borders)
	{
		for (int i = 0; i < K_WENO; i++)
		{

			int i_in = x_point + x_size + i + 1;//for right border
			int i_ex = x_point + x_size - i;

			r[i_ex][j] = r[i_in][j];
			for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i_ex][j] =
c[ic][i_in][j];
			u[i_ex][j] = -u[i_in][j];
			v[i_ex][j] = v[i_in][j];
			p[i_ex][j] = p[i_in][j];
			e[i_ex][j] = p[i_ex][j] / r[i_ex][j] / (calc_gam(i_in, j) - 1.0);
			E[i_ex][j] = e[i_ex][j] + (u[i_ex][j] * u[i_ex][j] + v[i_ex][j] *
v[i_ex][j]) * 0.5;

			for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i_ex][j] =
r[i_ex][j] * c[ic][i_ex][j];
			ru[i_ex][j] = r[i_ex][j] * u[i_ex][j];
			rv[i_ex][j] = r[i_ex][j] * v[i_ex][j];
			re[i_ex][j] = r[i_ex][j] * E[i_ex][j];

			i_in = x_point - i - 1;//for left border
			i_ex = x_point + i;
```

```
                    r[i_ex][j] = r[i_in][j];
                    for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i_ex][j] =
c[ic][i_in][j];
                    u[i_ex][j] = -u[i_in][j];
                    v[i_ex][j] = v[i_in][j];
                    p[i_ex][j] = p[i_in][j];
                    e[i_ex][j] = p[i_ex][j] / r[i_ex][j] / (calc_gam(i_in, j) - 1.0);
                    E[i_ex][j] = e[i_ex][j] + (u[i_ex][j] * u[i_ex][j] + v[i_ex][j] *
v[i_ex][j]) * 0.5;

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i_ex][j] =
r[i_ex][j] * c[ic][i_ex][j];
                    ru[i_ex][j] = r[i_ex][j] * u[i_ex][j];
                    rv[i_ex][j] = r[i_ex][j] * v[i_ex][j];
                    re[i_ex][j] = r[i_ex][j] * E[i_ex][j];


            }
      }
}


void du_bnd_cond_cube(int x_point, int y_point, int x_size, int y_size) {
      for (int i = x_point; i <= x_point + x_size; i++)//По X(top and bottom
borders)
      {
            for (int j = 0; j < K_WENO; j++)
            {
                  int j_in = y_point + y_size + j + 1;//for top border
                  int j_ex = y_point + y_size - j;

                  r[i][j_ex] = r[i][j_in];
                  for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j_ex] =
c[ic][i][j_in];
                  u[i][j_ex] = u[i][j_in];
                  v[i][j_ex] = -v[i][j_in];
                  p[i][j_ex] = p[i][j_in];
                  e[i][j_ex] = p[i][j_ex] / r[i][j_ex] / (calc_gam(i, j_in) - 1.0);
                  E[i][j_ex] = e[i][j_ex] + (u[i][j_ex] * u[i][j_ex] + v[i][j_ex] *
v[i][j_ex]) * 0.5;

                  for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j_ex] =
r[i][j_ex] * c[ic][i][j_ex];
                  ru[i][j_ex] = r[i][j_ex] * u[i][j_ex];
                  rv[i][j_ex] = r[i][j_ex] * v[i][j_ex];
                  re[i][j_ex] = r[i][j_ex] * E[i][j_ex];

                  j_in = y_point - j - 1;//for bottom border
                  j_ex = y_point + j;

                  r[i][j_ex] = r[i][j_in];
                  for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j_ex] =
c[ic][i][j_in];
                  u[i][j_ex] = u[i][j_in];
                  v[i][j_ex] = -v[i][j_in];
                  p[i][j_ex] = p[i][j_in];
                  e[i][j_ex] = p[i][j_ex] / r[i][j_ex] / (calc_gam(i, j_in) - 1.0);
                  E[i][j_ex] = e[i][j_ex] + (u[i][j_ex] * u[i][j_ex] + v[i][j_ex] *
v[i][j_ex]) * 0.5;
```

```
                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j_ex] =
r[i][j_ex] * c[ic][i][j_ex];
                    ru[i][j_ex] = r[i][j_ex] * u[i][j_ex];
                    rv[i][j_ex] = r[i][j_ex] * v[i][j_ex];
                    re[i][j_ex] = r[i][j_ex] * E[i][j_ex];


             }
      }
}

double rl, pl, ul, vl, cl[COMP_COUNT];
double rr, pr, ur, vr, cr[COMP_COUNT];
double r_[2 * K_WENO];
double c_[COMP_COUNT][2 * K_WENO];
double p_[2 * K_WENO];
double u_[2 * K_WENO];
double v_[2 * K_WENO];

void WENO(double* UU, double& Um, double& Up);
void calc_flux_hllc(
      double rl, double ul, double vl, double pl, double cl[], double gaml,
      double rr, double ur, double vr, double pr, double cr[], double gamr,
      double& qu, double& qv, double& qe, double qc[]);

void step()
{
      double thx = TAU / HX;
      double thy = TAU / HY;
      double fu, fv, fe, fc[COMP_COUNT];

      // X direction
      lr_bnd_cond_cube(x_point1, y_point1, x_size1, y_size1);
      //lr_bnd_cond_cube(x_point1_2, y_point1_2, x_size1_2, y_size1_2);
      for (int i = lo[0] - 1; i <= hi[0]; i++)
      {
            for (int j = lo[1]; j <= hi[1]; j++)
            {
                  for (int k = -K_WENO + 1; k <= K_WENO; k++)
                  {
                        for (int ic = 0; ic < COMP_COUNT; ic++) c_[ic][k + K_WENO -
1] = c[ic][i + k][j];
                        r_[k + K_WENO - 1] = r[i + k][j];
                        p_[k + K_WENO - 1] = p[i + k][j];
                        u_[k + K_WENO - 1] = u[i + k][j];
                        v_[k + K_WENO - 1] = v[i + k][j];
                  }
                  for (int ic = 0; ic < COMP_COUNT; ic++) WENO(c_[ic], cl[ic],
cr[ic]);
                  WENO(r_, rl, rr);
                  WENO(p_, pl, pr);
                  WENO(u_, ul, ur);
                  WENO(v_, vl, vr);

                  double gaml = calc_gam(i, j);
                  double gamr = calc_gam(i + 1, j);

                  calc_flux_hllc(rl, ul, vl, pl, cl, gaml,
                        rr, ur, vr, pr, cr, gamr,
```

```
                        fu, fv, fe, fc);

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] -= fc[ic] *
thx;
                    ru[i][j] -= fu * thx;
                    rv[i][j] -= fv * thx;
                    re[i][j] -= fe * thx;

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i + 1][j] +=
fc[ic] * thx;
                    ru[i + 1][j] += fu * thx;
                    rv[i + 1][j] += fv * thx;
                    re[i + 1][j] += fe * thx;
             }
      }

      // Y direction
      du_bnd_cond_cube(x_point1, y_point1, x_size1, y_size1);
      //du_bnd_cond_cube(x_point1_2, y_point1_2, x_size1_2, y_size1_2);
      for (int i = lo[0]; i <= hi[0]; i++)
      {
             for (int j = lo[1] - 1; j <= hi[1]; j++)
             {
                    for (int k = -K_WENO + 1; k <= K_WENO; k++)
                    {
                           r_[k + K_WENO - 1] = r[i][j + k];
                           for (int ic = 0; ic < COMP_COUNT; ic++) c_[ic][k + K_WENO -
1] = c[ic][i][j + k];
                           p_[k + K_WENO - 1] = p[i][j + k];
                           u_[k + K_WENO - 1] = u[i][j + k];
                           v_[k + K_WENO - 1] = v[i][j + k];
                    }
                    for (int ic = 0; ic < COMP_COUNT; ic++) WENO(c_[ic], cl[ic],
cr[ic]);
                    WENO(r_, rl, rr);
                    WENO(p_, pl, pr);
                    WENO(u_, ul, ur);
                    WENO(v_, vl, vr);

                    double gaml = calc_gam(i, j);
                    double gamr = calc_gam(i, j + 1);

                    calc_flux_hllc(rl, vl, ul, pl, cl, gaml,
                           rr, vr, ur, pr, cr, gamr,
                           fv, fu, fe, fc);

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] -= fc[ic] *
thy;
                    ru[i][j] -= fu * thy;
                    rv[i][j] -= fv * thy;
                    re[i][j] -= fe * thy;

                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j + 1] +=
fc[ic] * thy;
                    ru[i][j + 1] += fu * thy;
                    rv[i][j + 1] += fv * thy;
                    re[i][j + 1] += fe * thy;
             }
      }
```

```
}

void fix_c()
{
	for (int i = lo[0]; i <= hi[0]; i++)
	{
		for (int j = lo[1]; j <= hi[1]; j++)
		{
			for (int ic = 0; ic < COMP_COUNT; ic++) {
				if (c[ic][i][j] < 0.) c[ic][i][j] = 0.;
				if (c[ic][i][j] > 1.) c[ic][i][j] = 1.;
				rc[ic][i][j] = r[i][j] * c[ic][i][j];
			}
		}
	}
}

int main(int argc, char* argv[])
{
	init();


	double t = 0.0;
	int i_step = 0;

	sprintf(str, "ro.%010d", i_step);
	draw(str, NX, NY);

	while (t < TMAX)
	{
		t += TAU; i_step++;

		bnd_cond();

		for (int i = 0; i < NX_ALL; i++)
		{
			for (int ic = 0; ic < COMP_COUNT; ic++) memcpy(rc_old[ic][i],
rc[ic][i], sizeof(double) * (NY_ALL));
			memcpy(ru_old[i], ru[i], sizeof(double) * (NY_ALL));
			memcpy(rv_old[i], rv[i], sizeof(double) * (NY_ALL));
			memcpy(re_old[i], re[i], sizeof(double) * (NY_ALL));
		}

		step();
		for (int i = lo[0]; i <= hi[0]; i++)
		{
			for (int j = lo[1]; j <= hi[1]; j++)
			{
				r[i][j] = 0.;
				for (int ic = 0; ic < COMP_COUNT; ic++) r[i][j] +=
rc[ic][i][j];
				for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j] =
rc[ic][i][j] / r[i][j];
				u[i][j] = ru[i][j] / r[i][j];
				v[i][j] = rv[i][j] / r[i][j];
				E[i][j] = re[i][j] / r[i][j];
				e[i][j] = E[i][j] - (u[i][j] * u[i][j] + v[i][j] * v[i][j])
* 0.5;
				p[i][j] = r[i][j] * e[i][j] * (calc_gam(i, j) - 1.0);
```

```
                }
            }

            fix_c();

            bnd_cond();
            step();
            for (int i = lo[0]; i <= hi[0]; i++)
            {
                for (int j = lo[1]; j <= hi[1]; j++)
                {
                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] += 3.
* rc_old[ic][i][j];
                    ru[i][j] += 3. * ru_old[i][j];
                    rv[i][j] += 3. * rv_old[i][j];
                    re[i][j] += 3. * re_old[i][j];
                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] /= 4.;
                    ru[i][j] /= 4.;
                    rv[i][j] /= 4.;
                    re[i][j] /= 4.;

                    r[i][j] = 0.;
                    for (int ic = 0; ic < COMP_COUNT; ic++) r[i][j] +=
rc[ic][i][j];
                    for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j] =
rc[ic][i][j] / r[i][j];
                    u[i][j] = ru[i][j] / r[i][j];
                    v[i][j] = rv[i][j] / r[i][j];
                    E[i][j] = re[i][j] / r[i][j];
                    e[i][j] = E[i][j] - (u[i][j] * u[i][j] + v[i][j] * v[i][j])
* 0.5;
                    p[i][j] = r[i][j] * e[i][j] * (calc_gam(i, j) - 1.0);
                }
            }

            fix_c();

            bnd_cond();
            step();
            for (int i = lo[0]; i <= hi[0]; i++)
            {
                for (int j = lo[1]; j <= hi[1]; j++)
                {
                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] *= 2.;
                    ru[i][j] *= 2.;
                    rv[i][j] *= 2.;
                    re[i][j] *= 2.;
                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] +=
rc_old[ic][i][j];
                    ru[i][j] += ru_old[i][j];
                    rv[i][j] += rv_old[i][j];
                    re[i][j] += re_old[i][j];
                    for (int ic = 0; ic < COMP_COUNT; ic++) rc[ic][i][j] /= 3.;
                    ru[i][j] /= 3.;
                    rv[i][j] /= 3.;
                    re[i][j] /= 3.;

                    r[i][j] = 0.;
```

```
                        for (int ic = 0; ic < COMP_COUNT; ic++) r[i][j] +=
rc[ic][i][j];
                        for (int ic = 0; ic < COMP_COUNT; ic++) c[ic][i][j] =
rc[ic][i][j] / r[i][j];
                        u[i][j] = ru[i][j] / r[i][j];
                        v[i][j] = rv[i][j] / r[i][j];
                        E[i][j] = re[i][j] / r[i][j];
                        e[i][j] = E[i][j] - (u[i][j] * u[i][j] + v[i][j] * v[i][j])
* 0.5;
                        p[i][j] = r[i][j] * e[i][j] * (calc_gam(i, j) - 1.0);
                  }
            }

            fix_c();

            if (i_step % STEP_OUT == 0)
            {
                  sprintf(str, "ro.%010d", i_step);
                  draw(str, hi[0] - lo[0] + 1, hi[1] - lo[1] + 1);

            }
            if (i_step % 100 == 0) printf("step = %d\t\tt = %e\n", i_step, t);
      }


      for (int i = 0; i < NX_ALL; i++)
      {
            for (int ic = 0; ic < COMP_COUNT; ic++) delete[] c[ic][i];
            delete[] r[i];
            delete[] u[i];
            delete[] v[i];
            delete[] p[i];
            delete[] E[i];
            delete[] e[i];

            for (int ic = 0; ic < COMP_COUNT; ic++) delete[] rc[ic][i];
            delete[] ru[i];
            delete[] rv[i];
            delete[] re[i];

            for (int ic = 0; ic < COMP_COUNT; ic++) delete[] rc_old[ic][i];
            delete[] ru_old[i];
            delete[] rv_old[i];
            delete[] re_old[i];
      }
      for (int ic = 0; ic < COMP_COUNT; ic++) delete[] c[ic];
      delete[] r;
      delete[] u;
      delete[] v;
      delete[] p;
      delete[] E;
      delete[] e;

      for (int ic = 0; ic < COMP_COUNT; ic++) delete[] rc[ic];
      delete[] ru;
      delete[] rv;
      delete[] re;

      for (int ic = 0; ic < COMP_COUNT; ic++) delete[] rc_old[ic];
```

```
	delete[] ru_old;
	delete[] rv_old;
	delete[] re_old;
	return 0;
}

double KR(double x)
{
	if (x >= 0.01 && x <= 0.03) return -0.01 * sin(M_PI * (x - 0.01) / 0.02);
	return 0.0;
}


inline double	_MAX_(double x, double y)
{
	if (x <= y) return y;
	return x;
}

void WENO(double* UU, double& Um, double& Up)
{
	double BETA[3];
	double ALPHA[3];
	double eps = 1.0e-6;
	if ((UU[2] - UU[1]) * (UU[3] - UU[2]) < 0.0) Um = UU[2];
	else
	{
		BETA[0] = (13. / 12.) * (UU[2] - 2 * UU[3] + UU[4]) * (UU[2] - 2 * UU[3]
+ UU[4]) + 0.25 * (3 * UU[2] - 4 * UU[3] + UU[4]) * (3 * UU[2] - 4 * UU[3] + UU[4]);
		BETA[1] = (13. / 12.) * (UU[1] - 2 * UU[2] + UU[3]) * (UU[1] - 2 * UU[2]
+ UU[3]) + 0.25 * (UU[1] - UU[3]) * (UU[1] - UU[3]);
		BETA[2] = (13. / 12.) * (UU[0] - 2 * UU[1] + UU[2]) * (UU[0] - 2 * UU[1]
+ UU[2]) + 0.25 * (UU[0] - 4 * UU[1] + 3 * UU[2]) * (UU[0] - 4 * UU[1] + 3 * UU[2]);
		ALPHA[0] = 0.3 / ((eps + BETA[0]) * (eps + BETA[0]));
		ALPHA[1] = 0.6 / ((eps + BETA[1]) * (eps + BETA[1]));
		ALPHA[2] = 0.1 / ((eps + BETA[2]) * (eps + BETA[2]));
		Um = (ALPHA[0] * (2 * UU[2] + 5 * UU[3] - UU[4]) + ALPHA[1] * (-UU[1] +
5 * UU[2] + 2 * UU[3]) +
			ALPHA[2] * (2 * UU[0] - 7 * UU[1] + 11 * UU[2])) / ((ALPHA[0] +
ALPHA[1] + ALPHA[2]) * 6);
	}
	if ((UU[3] - UU[2]) * (UU[4] - UU[3]) < 0.0) Up = UU[3];
	else
	{
		BETA[0] = (13. / 12.) * (UU[3] - 2 * UU[4] + UU[5]) * (UU[3] - 2 * UU[4]
+ UU[5]) + 0.25 * (3 * UU[3] - 4 * UU[4] + UU[5]) * (3 * UU[3] - 4 * UU[4] + UU[5]);
		BETA[1] = (13. / 12.) * (UU[2] - 2 * UU[3] + UU[4]) * (UU[2] - 2 * UU[3]
+ UU[4]) + 0.25 * (UU[2] - UU[4]) * (UU[2] - UU[4]);
		BETA[2] = (13. / 12.) * (UU[1] - 2 * UU[2] + UU[3]) * (UU[1] - 2 * UU[2]
+ UU[3]) + 0.25 * (UU[1] - 4 * UU[2] + 3 * UU[3]) * (UU[1] - 4 * UU[2] + 3 * UU[3]);
		ALPHA[0] = 0.1 / ((eps + BETA[0]) * (eps + BETA[0]));
		ALPHA[1] = 0.6 / ((eps + BETA[1]) * (eps + BETA[1]));
		ALPHA[2] = 0.3 / ((eps + BETA[2]) * (eps + BETA[2]));
		Up = (ALPHA[0] * (11 * UU[3] - 7 * UU[4] + 2 * UU[5]) + ALPHA[1] * (2 *
UU[2] + 5 * UU[3] - UU[4]) +
			ALPHA[2] * (-UU[1] + 5 * UU[2] + 2 * UU[3])) / ((ALPHA[0] +
ALPHA[1] + ALPHA[2]) * 6);
	}
}
```

```
void draw(char* fName, int nx, int ny)
{
	char htmlName[50];
	strcpy(htmlName, fName);
	strcat(htmlName, ".vtk");
	FILE* fp = fopen(htmlName, "w");
	fprintf(fp, "# vtk DataFile Version 2.0\n");
	fprintf(fp, "results\n");
	fprintf(fp, "ASCII\n");
	fprintf(fp, "DATASET UNSTRUCTURED_GRID\n");

	int pCount = (nx + 1) * (ny + 1);
	fprintf(fp, "POINTS %d float\n", pCount);

	for (int j = 0; j <= ny; j++)
	{
		for (int i = 0; i <= nx; i++)
		{
			fprintf(fp, "%f %f %f  \n", XMIN + HX * i, YMIN + HY * j, 0.0);
		}
	}
	fprintf(fp, "\n");

	int cellsCount = nx * ny;

	fprintf(fp, "CELLS %d %d\n", cellsCount, 5 * cellsCount);
	for (int i = 0; i < nx; i++)
	{
		for (int j = 0; j < ny; j++)
		{
			fprintf(fp, "4 %d %d %d %d  \n", j * (nx + 1) + i, j * (nx + 1) +
i + 1, (j + 1) * (nx + 1) + i + 1, (j + 1) * (nx + 1) + i);
		}
	}
	fprintf(fp, "\n");

	fprintf(fp, "CELL_TYPES %d\n", cellsCount);
	for (int i = 0; i < cellsCount; i++) fprintf(fp, "9\n");
	fprintf(fp, "\n");

	fprintf(fp, "CELL_DATA %d\n", cellsCount);

	fprintf(fp, "SCALARS Density float 1\nLOOKUP_TABLE default\n");
	for (int i = lo[0]; i <= hi[0]; i++)
	{
		for (int j = lo[1]; j <= hi[1]; j++)
		{
			fprintf(fp, "%f ", r[i][j]);
		}
		fprintf(fp, "\n");
	}

	fprintf(fp, "SCALARS Pressure float 1\nLOOKUP_TABLE default\n");
	for (int i = lo[0]; i <= hi[0]; i++)
	{
		for (int j = lo[1]; j <= hi[1]; j++)
		{
			//will cut "trash" pressure in Paraview
```

```
                    (trash(x_point1, y_point1, x_size1, y_size1, i, j))? //||
trash(x_point1_2, y_point1_2, x_size1_2, y_size1_2, i, j) ) ?
                          fprintf(fp, "%f ", 0) : fprintf(fp, "%f ", p[i][j]);
                    if (p[i][j] != p[i][j]) {
                          int zhrv = 0;
                    }
             }
             fprintf(fp, "\n");
      }

      fprintf(fp, "SCALARS Energy float 1\nLOOKUP_TABLE default\n");
      for (int i = lo[0]; i <= hi[0]; i++)
      {
             for (int j = lo[1]; j <= hi[1]; j++)
             {
                    fprintf(fp, "%f ", E[i][j]);
             }
             fprintf(fp, "\n");
      }

      fprintf(fp, "SCALARS Mach_number float 1\nLOOKUP_TABLE default\n");
      for (int i = lo[0]; i <= hi[0]; i++)
      {
             for (int j = lo[1]; j <= hi[1]; j++)
             {
                    fprintf(fp, "%f ", sqrt((u[i][j] * u[i][j] + v[i][j] * v[i][j]) /
(calc_gam(i, j) * p[i][j] / r[i][j])));
             }
             fprintf(fp, "\n");
      }

      fprintf(fp, "VECTORS Velosity float\n");
      for (int i = lo[0]; i <= hi[0]; i++)
      {
             for (int j = lo[1]; j <= hi[1]; j++)
             {
                    fprintf(fp, "%f %f %f  ", u[i][j], v[i][j], 0.0);
             }
             fprintf(fp, "\n");
      }

      for (int ic = 0; ic < COMP_COUNT; ic++) {
             fprintf(fp, "SCALARS C_%03d float 1\nLOOKUP_TABLE default\n", ic);
             for (int i = lo[0]; i <= hi[0]; i++) {
                    for (int j = lo[1]; j <= hi[1]; j++) {
                          fprintf(fp, "%f ", c[ic][i][j]);
                    }
                    fprintf(fp, "\n");
             }
      }

      fclose(fp);
      printf("File '%s' saved...\n", htmlName);


}
```

```
#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 calc_flux_hllc(
	double rl, double ul, double vl, double pl, double cl[], double gaml,
	double rr, double ur, double vr, double pr, double cr[], double gamr,
	double& qu, double& qv, double& qe, double qc[])
{
	int		i;
	double		sl, sr, p_star, s_star, p_pvrs, ql, qr, tmp;

	double czl = sqrt(pl * gaml / rl);
	double czr = sqrt(pr * gamr / rr);

	double el = pl / (rl * (gaml - 1.));
	double er = pr / (rr * (gamr - 1.));

	double e_tot_l = el + 0.5 * (ul * ul + vl * vl);
	double e_tot_r = er + 0.5 * (ur * ur + vr * vr);

	p_pvrs = 0.5 * (pl + pr) - 0.5 * (ur - ul) * 0.25 * (rl + rr) * (czl + czr);
	p_star = (p_pvrs > 0.) ? p_pvrs : 0.;

	ql = (p_star <= pl) ? 1 : sqrt(1. + (gaml + 1.) * (p_star / pl - 1.) / (2. *
gaml));
	qr = (p_star <= pr) ? 1 : sqrt(1. + (gamr + 1.) * (p_star / pr - 1.) / (2. *
gamr));

	sl = ul - czl * ql;
	sr = ur + czr * qr;

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

	s_star = pr - pl;
	s_star += rl * ul * (sl - ul);
	s_star -= rr * ur * (sr - ur);
	s_star /= (rl * (sl - ul) - rr * (sr - ur));

	if (s_star < sl) s_star = sl;
	if (s_star > sr) s_star = sr;


	if (!((sl <= s_star) && (s_star <= sr))) {
		printf("HLLC: inequaluty SL <= S* <= SR is FALSE.\n");
		exit(1);
	}

	if (sl >= 0.) {
		qu = rl * ul * ul + pl;
		qv = rl * vl * ul;
		qe = (rl * e_tot_l + pl) * ul;
```

Окончание ПРИЛОЖЕНИЯ А

```
	for (i = 0; i < COMP_COUNT; i++) {
		qc[i] = rl * ul * cl[i];
	}}
else if (sr <= 0.) {
	qu = rr * ur * ur + pr;
	qv = rr * vr * ur;
	qe = (rr * e_tot_r + pr) * ur;
	for (i = 0; i < COMP_COUNT; i++) {
		qc[i] = rr * ur * cr[i];
	}}
else {
	if (s_star >= 0) {
		qu = F_HLLC_U( /*  UK, FK, SK, SS, PK, RK, VK */
			rl * ul,
			rl * ul * ul + pl,
			sl, s_star, pl, rl, ul
		);
		qv = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
			rl * vl,
			rl * ul * vl,
			sl, s_star, pl, rl, ul
		);
		qe = F_HLLC_E( /*  UK, FK, SK, SS, PK, RK, VK */
			rl * e_tot_l,
			(rl * e_tot_l + pl) * ul,
			sl, s_star, pl, rl, ul
		);
		for (i = 0; i < COMP_COUNT; i++) {
			qc[i] = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
				rl * cl[i],
				rl * cl[i] * ul,
				sl, s_star, pl, rl, u);}}
	else {
		qu = F_HLLC_U( /*  UK, FK, SK, SS, PK, RK, VK */
			rr * ur,
			rr * ur * ur + pr,
			sr, s_star, pr, rr, ur
		);
		qv = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
			rr * vr,
			rr * ur * vr,
			sr, s_star, pr, rr, ur
		);
		qe = F_HLLC_E( /*  UK, FK, SK, SS, PK, RK, VK */
			rr * e_tot_r,
			(rr * e_tot_r + pr) * ur,
			sr, s_star, pr, rr, ur
		);
		for (i = 0; i < COMP_COUNT; i++) {
			qc[i] = F_HLLC_V( /*  UK, FK, SK, SS, PK, RK, VK */
				rr * cr[i],
				rr * cr[i] * ur,
				sr, s_star, pr, rr, ur
			);
		}}}}
```

Заявление о самостоятельном характере выполнения выпускной квалификационной работы

Я, Багапов Ариф Ренатович, студент 4 курса, направления подготовки Прикладная математика и информатика, заявляю, что в моей работе на тему «Моделирование акустических полей при обтекании тел потоком газа», 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. Огарева» на наличие заимствований, в соответствии с которым обнаружение неправомерных заимствований является основанием для отрицательного отзыва руководителя работы.

Багапов А.Р. _____ _____

ФИО *подпись*

11 июня 2020г.

дата

Заведующему кафедрой
прикладной математики,
дифференциальных уравнений и
теоретической механики, канд.
физ.-мат. наук, доц.,
Р. В. Жалнину
студента 4 курса
очной формы обучения
(на бесплатной основе)
направления подготовки
«Прикладная математика и
информатика» факультета
математики и информационных
технологий
Багапова Арифа Ренатовича

заявление

Прошу разместить мою выпускную квалификационную работу на тему «Моделирование акустических полей при обтекании тел потоком газа» в электронной библиотечной системе университета в полном объеме.

Багапов А.Р.
ФИО

подпись

11 июня 2020г.
дата

ОТЗЫВ

о выпускной квалификационной работе
студента Багапова Арифа Ренатовича
обучающегося по направлению подготовки
01.03.02 «Прикладная математика и информатика»
на тему «Моделирование акустических полей при обтекании тел потоком газа»

Задача моделирования акустических полей при обтекании различных тел потоком газа является одной из актуальных задач современной прикладной математики. Наличие эффективных инструментов математического моделирования и оценки акустических характеристик может позволить конструировать изделия с минимальным акустическим воздействием на окружающие объекты. Это делает актуальным развитие методов моделирования акустических полей.

Работа Багапова А.Р. посвящена построению вычислительного алгоритма для моделирования акустических полей при обтекании тел потоком газа. В ходе работы автором самостоятельно были решены следующие задачи:

- изучена литература и выполнен обзор современного состояния вопроса;
- реализован алгоритм расчета акустических характеристик;
- выполнено моделирование акустических полей при обтекании различных конфигураций тел.

Во время выполнения работы Багапов А.Р. продемонстрировал самостоятельность в решении поставленных перед ним задач, проявил усердие и устремленность к достижению поставленных целей.

Бакалаврская работа Багапова Арифа Ренатовича соответствует заданию и всем требованиям, предъявляемым к работам подобного рода. Считаю, что работа заслуживает оценки «отлично», а ее автор заслуживает присвоения степени бакалавра по направлению подготовки «Прикладная математика и информатика»

Научный руководитель
к.ф.-м.н., доцент Р.В. Жалнин

ОТЧЕТ

о результатах проверки работы обучающегося

на наличие заимствований

Ф.И.О. автора работы Багапов Ариф Ренатович

Тема работы Моделирование акустических полей при обтекании тел потоком газа

Руководитель Жалнин Руслан Викторович

Представленная работа прошла проверку на наличие заимствований в системе «Антиплагиат. ВУЗ»

Результаты автоматической проверки: оригинальность 82.08 %

цитирования 3.86 %

заимствования 14.06 %

Результаты анализа полного отчета на наличие заимствований:

правомерные заимствования: 14.06 %

корректные цитирования: 3.86 %

неправомерные заимствования: нет

признаки обхода системы: нет

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

Студент допускается к предварительной защите и защите ВКР в ГЭК.

Руководитель работы
к.ф.-м.н., доцент Р. В. Жалнин