Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный университет имени М.В.
Ломоносова»

Механико-математический факультет
Кафедра вычислительной механики

Выпускная квалификационная работа
(Дипломная работа)
специалиста

Применение различных схем для численного решения уравнения переноса

Студент 621 группы
Сенченок Григорий Антонович

Научный руководитель:
д.ф.-м.н., профессор Меньшов Игорь Станиславович

Москва
2022

Аннотация

Предметом исследования данной работы является цифровое представление нестационарной геометрии на декартовых сетках. С этой целью рассматривается численное решение уравнения переноса для характеристической функции, определяющей положение рассматриваемой геометрии в пространстве. Ключевым моментом реализации данного подхода является минимизация численной диссипации разностной схемы в окрестности точек разрыва характеристической функции. Задача дипломной работы состоит в разработке и программной реализации численного метода на основе подсеточной реконструкции разрыва. Рассматриваются три различных схемы восполнения численного решения на подсеточном уровне. Это - классическиq метод Годунова с интероляцией типа константа, MUSCL (линейная функция), относительно новый метод THINC (восполнение сигмоидной функцией), а также предложенный метод Jump Reconstruction (JR) на основе восполнения разрывной функцией Хевисайда.

Содержание

1 Введение

В работе рассматривается задача численного моделирования движения твердого тела. Твердое тело задано как совокупность точек, лежащих на его границе. Такое представление называется геометрическим. При расчете движения отслеживается положение этих точек, таким образом, происходит моделирование движения в исследуемой области. Основным недостатком геометрического представления является сложность его совмещения с расчетом взаимодействия тела и сплошной среды. В настоящей работе представлен альтернативный способ описания пространственной геметрии, основанный на цифровом представлении движения твердого тела.
В промышленных системах расчета (например, CAD) для описания геометрии используется стандартный подход с описанием тела в виде набор примитивов - конечных элементов. На всей исследуемой области вводится сетка, состоящая из простых элементов: для двумерного случая – это многоугольники, для трехмерного – многогранники. Таким образом, происходит точный расчет для большого числа малых конечных элементов, что требует больших вычислительных мощностей для описания каждого примитива и сложного разбиения на эти конечные элементы.
В методе, описанном в данной работе, геометрия тела задается характеристической функцией. Данная функция в рассматриваемой области представляет собой индикатор и принимает значение 0, если в рассматриваемой точке находится твердое тело и 1, если жидкость. Эволюцию данной скалярной величины в пространстве и времени описывает уравнение переноса. Такой подход к представлению твердого тела называется цифровой геометрией: Digital Geometry (DG). Это сильно упрощает введение сетки и облегчает расчеты, сохраняя при этом точность на достаточно высоком уровне.

Для численного решения уравнения переноса широко используется дискретное представление характеристической функции на основе объема жидкости в ячейке сетки (volume of fluid, VOF). Ключевым моментом реализации данного метода является аппроксимация скачка, которая является разрывом характеристической функции и наблюдается на границе двух сред. Для данного процесса разрабатываются различные схемы геометрической реконструкции разрыва, позволяющие получать достаточную точность. В работе проводится исследование классических способов аппроксимации решения с помощью константного восполнения - метод Годунова [2], [3], с помощью линейного восполнения - метод MUSCL [11], [12]. А также проведено исследование альтернативной схемы вос-

полнения гиперболическим тангенсом - метод THINC. При использовании данной схемы была достигнута высокая точность расчетов. Схема THINC - (tangent of hyperbola for INterface capturing) схема отслеживание поверхности с помощью гиперболического тангенса [14], [16]. Также был представлен новый метод численной реконструкции с помощью функции скачка - метод Jump Reconstruction (JR). Данный метод основан на решении обобщенной задачи Римана со смещенным начальным разрывом. В простейшем случае векторное поле, которое воздействует на твердое тело, индуцировано самим твердым телом. Таким образом, расчеты, проведенные для цифровой геометрии возможно сравнить с точным решением. Проблему представления движения в цифровой геометрии можно разбить на две подзадачи: нахождение векторного поля скоростей твердого тела и нахождение характеристической функции твердого тела и жидкости в рассматриваемой области на каждом временном шаге. Также была решена задача точного расчета движения твердого тела при известном векторе скорости цента масс и угловой скорости: DirectMotion (DM).
Цель данной работы - численная реализация и сравнение различных схем геометрической реконструкции:

- метод Годунова
- схема MUSCL
- метод THINC
- метод JR

2 Постановка задачи

2.1 Цифровая геометрия

Общая задача цифрового представления нестационарной геометрии заключается в нахождении характеристической функции твердого тела в исследуемой области пространства в каждый момент времени по известному начальному положению точек твердого тела, а также заданной скорости центра твердого тела и угловой скорости. Данная задача разбивается на следующие подзадачи:

- Точное решение: Direct Motion
  - Нахождение положения центра твердого тела в каждый момент времени

  - Нахождение положения точек твердого тела в каждый момент времени
  - Нахождение поля скоростей, индуцированное твердым телом

- Численное решение уравнения переноса при заданном поле скоростей

2.2 Точное решение: Direct Motion

Необходимо реализовать программный алгоритм расчета положения твердого тела при движении.
Твердое тело представляет собой совокупность точек, расстояния, между положениями которых не изменяются. Известно начальное положение всех необходимых для расчета точек. Одна из точек твердого тела является центром масс. Скорость данной точки известна в любой момент времени. Программа должна рассчитать положение всех точек твердого тела и построить траекторию движения на заданном временном отрезке. Программный алгоритм должен обладать достаточно высокой точностью, для использования результатов его вычислений при расчете движения твердого тела в сплошной среде. Для подсчета точности вычисленного решения предполагается сравнение результатов численного метода с аналитическим решением.
Алгоритм должен работать для расчета положения точек тела, как в двумерном, так и в трехмерном пространстве.

2.3 Уравнение переноса

Движение твердого тела описывается с помощью уравнения переноса. Данное дифференциальное уравнение в частных производных описывает изменение скалярной величины в пространстве и времени. Необходимо по известному распределению функции скалярной величины f в начальный момент времени, а также с заданным полем скоростей на каждом моменте времени, рассчитать характеристическую функцию твердого тела и жидкости (скалярную величину f) в рассматриваемой области на каждом временном шаге. В рамках решения предыдущей задачи было рассчитано поле скоростей в области D, индуцированное твердым телом при движении.

При реализации программного алгоритма в задаче данной работы предполагается рассмотреть схемы Годунова, MUSCL, THINC и JR. Необходимо реализовать данный алгоритм в одномерном случае на языке программирования C++, а также использовать полученную схему для

каждого из двух или трех измерений, используя расщепление по направлениям.

2.4 Jump Reconstruction

Отдельно выделяется задача разработки нового алгоритма подсеточной реконструкции, позволяющей с высокой точностью определить границу разрыва сред, а также распределение характеристической функции. В качестве такого метода представляется метод Jump Reconstruction - восполнение кусочно - постоянной функцией, в основе которой лежит разрывная функция.
Необходима реализация данного алгоритма с помощью программных средств, а также сравнение результатов ее работы с признанными существующими методами подсеточной реконструкции и с аналитическим решением.

2.5 Ограничения и допущения

При реализации алгоритмов численных методов рассматриваемая область пространства и отрезок времени разбиваются на отрезки равной длины: вводится равномерная сетка с достаточно малой длиной отрезков. Значения искомых величин вычисляются в узлах данной сетки: в определенной клетке пространства и на определенном шаге по времени. Длина и количество отрезков разбиения выбираются таким образом, чтобы полученное численное решение аппроксимировало аналитическое с высокой точностью. То есть результатом численного решения задач, связанных с нахождением функции, являются не сами функции, а их приближения с некоторой точностью.

При реализации программного алгоритма расчета положения твердого тела при движении для данной задачи мы ограничиваемся двумерным случаем.

При решении задачи точного решения (Direct Motion) граница твердого тела представляет собой многоугольник, состоящий из характерных точек поверхности. Точки считаются характерными точками поверхности, если линию поверхности между двумя соседними точками можно с высокой точностью аппроксимировать прямой линией.

При реализации метода VoF характеристическая функция может принимать значения в диапазоне от 0 до 1. Значения характеристической функции в таком случае будут показывать объемную долю жидкого и твердого вещества в точке пространства x в момент времени t.

3 Точное решение

Была решена вспомогательная задача о расчете точного положения точек твердого тела при заданных условиях на скорости и начальное положение.
При известном векторе скорости центра масс, векторе угловой скорости твердого тела и начальном положении всех точек твердого тела были численно рассчитаны положения всех точек твердого тела в моменты времени t_i. Расчет был произведен для двумерного случая, но его результаты можно применить и при расчете движения в трехмерном пространстве. Без ограничения общности рассмотрим задачу точного движения в двумерном случае.

3.1 Постановка задачи для точного решения

В двумерное пространство помещено твердое тело, представляющее собой совокупность точек, расстояния, между положениями которых не изменяются. Задана абсолютная система координат.
Начальное положение точек твердого тела: $\overrightarrow{x}_k^0 = \overrightarrow{x}_k(0)$, в том числе центра масс: $\overrightarrow{x}_0^0 = \overrightarrow{x}_c^0 = \overrightarrow{x}_c(0)$.
Вектор скорости центра масс: $\overrightarrow{v} = \overrightarrow{v}(t)$. В случае двумерного пространства: $\overrightarrow{v} = (v_x(t), v_y(t), 0)^T$.
Вектор угловой скорости твердого тела: $\overrightarrow{\omega} = \overrightarrow{\omega}(t)$. В случае двумерного пространства: $\overrightarrow{\omega} = (0, 0, \omega_z(t))^T$.

Дискретизация. Введем сетку:
Возьмем отрезок времени $T = [0; t]$ и разобьем его на $stepN$ подотрезков $T_i = [t_{i-1}; t_i]$, $i = 1..stepN$ - шаги по времени. Длина каждого шага по времени: $\Delta t = t_i - t_{i-1}$. t_i - узлы данного разбиения.
Необходимо численно рассчитать положение всех точек $\overrightarrow{x}_k$ твердого тела в узлах моменты времени t_i, $i = 1..stepN$.

3.2 Теоретическая часть

За скорость любой точки твердого тела отвечает формула Эйлера:

$$\overrightarrow{v}_k(t) = \overrightarrow{v}_c(t) + [\overrightarrow{\omega}(t) \times \overrightarrow{r}_k(t)] \tag{1}$$

Где $\overrightarrow{r}_k(t)$ - радиус вектор от центра масс до k-ой точки твердого тела.

$$\overrightarrow{r}_k(t) = \overrightarrow{x}_k(t) - \overrightarrow{x}_c(t) \tag{2}$$

$$\overrightarrow{v}_k(t) = \frac{\mathrm{d}\overrightarrow{x}_k}{\mathrm{d}t} \tag{3}$$

Подставляя данные значения в уравнение (1), получим обычное дифференциальное уравнение для координаты каждой точки твердого тела:

$$\frac{\mathrm{d}\overrightarrow{x}_k}{\mathrm{d}t} = \overrightarrow{v}_c(t) - [(\overrightarrow{x}_k(t) - \overrightarrow{x}_c(t)) \times \overrightarrow{\omega}(t)] \tag{4}$$

Для центра твердого тела данное уравнение имеет вид:

$$\frac{\mathrm{d}\overrightarrow{x}_c}{\mathrm{d}t} = \overrightarrow{v}_c(t) \tag{5}$$

Интегрирование данного уравнения от 0 до t с использованием начальных условий $\overrightarrow{x}_c(0) = \overrightarrow{x}_c^0$ дает интегральное уравнение для координаты центра твердого тела:

$$\overrightarrow{x}_c(t) = \overrightarrow{x}_c^0 + \int_0^t \overrightarrow{v}_c(\tau)d\tau \tag{6}$$

Раскрывая разность под векторным произведением в уравнении (4), получаем:

$$\frac{\mathrm{d}\overrightarrow{x}_k}{\mathrm{d}t} = \overrightarrow{v}_c(t) - [\overrightarrow{x}_k(t) \times \overrightarrow{\omega}(t)] + [\overrightarrow{x}_c(t) \times \overrightarrow{\omega}(t)]$$

Интегрирование данного уравнения от 0 до t с использованием начальных условий $\overrightarrow{x}_k(0) = \overrightarrow{x}_k^0$ дает интегральное уравнение для координаты произвольной точки твердого тела:

$$\overrightarrow{x}_k(t) - \int_0^t [\overrightarrow{x}_k(\tau) \times -\overrightarrow{\omega}(\tau)]d\tau = \int_0^t (\overrightarrow{v}_c(\tau) + [\overrightarrow{x}_c(\tau) \times -\overrightarrow{\omega}(\tau)])d\tau + \overrightarrow{x}_k^0 \tag{7}$$

3.3 Численное решение

Полученные интегральные уравнения были решены численно методом квадратур.

Центр твердого тела. Для решения интегрального уравнения для центра твердого тела (6) была использована составная квадратурная формула трапеции. Пример программного кода на языке программирования C++ представлен в приложении (9.1) к данной работе.
В силу того, что в уравнении отсутствует векторные произведения, а также другие математические конструкции, способные вызвать смешивание координат, возможно разложение векторов по направлениям осей

координат и составление решения векторного интегрального уравнения из отдельных решений интегральных уравнений вдоль каждого из направлений Ox, Oy.
Таким образом, были получены значения $\overrightarrow{x}_k^i$ сетки в узлах t_i сетки.
Не смотря на использование квадратурной формулы трапеции, которая имеет всего лишь 2 порядок сходимости, численные результаты полученные данным методом имеют высокую точность. В силу того, что в качестве скорости центра твердого тела используется гладкая функция, а также за счет достаточно мелкой сетки численное решение получилось крайне близким к аналитическому решению.
Полученные значения координаты центра твердого тела в узлах t_i сетки будут использованы в дальнейшем для нахождения векторного поля скоростей твердого тела на декартовой сетке, введенной на исследуемой области. Для этого будет использована та же формула Эйлера для скоростей твердого тела (1).

Точка твердого тела. Уравнение (7) также было численно решено методом квадратур. Подход к его решению аналогичен подходу к решению интегрального уравнения Вольтера второго рода. Рассматривается уравнение:

$$u(x) - \int_a^k K(x,s)u(s)ds = f(x) \tag{8}$$

В данном уравнении присутствуют следующие выражения:

x,s - переменная - параметр

$u(x)$ - искомая функция

$K(x,s)$ - ядро интегрального уравнения

$f(x)$ - функция правой части

В случае интегрального уравнения (7) для координаты точки твердого тела, параметр x и s - это время t и τ, а функции в уравнении Вольтера имеют вид:
Искомая функция:

$$u(t) = x_k(t)$$

Произведение функций $K(x,s)u(s)$ в нашем случае является векторным произведением $[\overrightarrow{x}_k(\tau) \times -\overrightarrow{\omega}(\tau)]$, и не зависит от t.
$a = 0$, при $t = t_0 = a = 0$

Функция правой части:

$$f(t) = \int_0^t (\overrightarrow{v}_c(\tau) + [\overrightarrow{x}_c(\tau) \times -\overrightarrow{\omega}(\tau)])d\tau + \overrightarrow{x}_k^0 \quad (9)$$

Значения данной функции были вычислены аналогично, методом квадратур с использованием составной формулы трапеции. Пример программного кода на C++ расчета значений функции правой части находится в Приложении (9.2).
Таким образом, были получены значения $\overrightarrow{f}^i$ вектор - функции $\overrightarrow{f}$ в узлах t_i.

$$\overrightarrow{x}_k^0 = \overrightarrow{f}(0) = \overrightarrow{f}_0$$

Применим принцип численного решения интегрального уравнения Вольтера второго рода для вычисления значений $\overrightarrow{x}_k^i$ вектор – функции $\overrightarrow{x}_k(t)$ в узлах равномерной сетки $T_i = [t_{i-1}; t_i]$, $i = 1..stepN$. Шаг сетки $\Delta t = t_i - t_{i-1}$. Количество узлов: $stepN + 1$.
При $t = t_i$ заменим интеграл в левой части уравнения на квадратурную формулу с коэффициентами $A_j^{stepN+1}$:

$$\overrightarrow{x}_k^i - \Delta x \sum_{j=0}^{i} A_j^{stepN+1} [\overrightarrow{x}_k^j \times -\overrightarrow{\omega}_j] = \overrightarrow{f}_i + \overrightarrow{R}_i^{stepN+1}(x)$$

Где $\overrightarrow{R}_i^{stepN+1}(x)$ - погрешность квадратурой формулы. Отбрасывая данную малую величину, решим систему уравнений относительно $\overrightarrow{x}_k^i$, $i = 0..stepN$.

$$\overrightarrow{x}_k^i - \Delta x \sum_{j=0}^{i-1} A_j^{stepN+1} [\overrightarrow{x}_k^j \times -\overrightarrow{\omega}_j] - \Delta x A_i^{stepN+1} [\overrightarrow{x}_k^i \times -\overrightarrow{\omega}_i] = \overrightarrow{f}_i$$

$$\overrightarrow{x}_k^i - [\overrightarrow{x}_k^i \times -\Delta x A_i^{stepN+1} \overrightarrow{\omega}_i] = \overrightarrow{f}_i + \Delta x \sum_{j=0}^{i-1} A_j^{stepN+1} [\overrightarrow{x}_k^j \times -\overrightarrow{\omega}_j] \quad (10)$$

Введем новые обозначения:

$$\overrightarrow{x}_k^i = \overrightarrow{x}$$

$$-\Delta x A_i^{stepN+1} \overrightarrow{\omega}_i = \overrightarrow{b}$$

$$\overrightarrow{f}_i + \Delta x \sum_{j=0}^{i-1} A_j^{stepN+1} [\overrightarrow{x}_k^j \times -\overrightarrow{\omega}_j] = \overrightarrow{c}$$

Тогда уравнение (10) примет вид:

$$\overrightarrow{x} - [\overrightarrow{x} \times \overrightarrow{b}] = \overrightarrow{c}$$

$$\overrightarrow{x} + [\overrightarrow{b} \times \overrightarrow{x}] = \overrightarrow{c}$$

Заменим операцию векторного произведения на произведение кососимметрической матрицы на вектор.

$$\overrightarrow{x} + [\overrightarrow{b}]_{\times} \overrightarrow{x} = \overrightarrow{c}$$

$$(E + [\overrightarrow{b}]_{\times}) \overrightarrow{x} = \overrightarrow{c}$$

$$\overrightarrow{x} = (E + [\overrightarrow{b}]_{\times})^{-1} \overrightarrow{c}$$

Матрица $E+[\overrightarrow{b}]_{\times}$ была рассчитана для двумерного случая, однако можно использовать тот же способ для расчета в трехмерном пространстве. Таким образом, мы выражаем вектор x_k^i для $i = 0..stepN$. Вектор x_k^i необходимо рассчитывать последовательно, так как в сумме $\sum_{j=0}^{i-1} A_j^{stepN+1} [\overrightarrow{x}_k^j \times -\overrightarrow{\omega}_j]$ используются значения x_k^j для $j = 0..i-1$.
В качестве квадратурной формулы была использована комбинация составной формулы Симпсона и правила трех восьмых.
В результате была реализована программа для вычисления положений k-ой точки твердого тела в моменты времени t_i,$i = 0..stepN$.
Пример программного кода, реализующий данный алгоритм находится в Приложении (9.3).

3.4 Поле скоростей твердого тела.

Поле скоростей твердого тела связано по формуле Эйлера:

$$\overrightarrow{u}(\overrightarrow{x}, t) = \overrightarrow{v_c}(t) + [\overrightarrow{\omega}(t) \times (\overrightarrow{x}_c(t) - \overrightarrow{x})]$$

Рассмотрим составляющие поля скоростей вдоль направлений X,Y,Z:

$$\overrightarrow{u}(\overrightarrow{r}, t) = (u_x, u_y, u_z)(\overrightarrow{r}, t)$$

$$\overrightarrow{r}(t) = (x - x_c(t), y - y_c(t), z - z_c(t))$$

Векторы $\overrightarrow{v_c}(t)$ - скорость центра т.т. и $\overrightarrow{\omega}(t)$ - угловая скорость т.т. считаются заданными.

$$\overrightarrow{v_c}(t) = (v_{cx}(t), v_{cy}(t), v_{cz}(t))$$

$$\overrightarrow{\omega}(t) = (\omega_x(t), \omega_y(t), \omega_z(t))$$

$$u_x(t) = v_{cx}(t) + \omega_y(t)z - \omega_y(t)z_c(t) - \omega_z(t)y + \omega_z(t)y_c(t)$$
$$u_y(t) = v_{cy}(t) - \omega_x(t)z + \omega_x(t)z_c(t) + \omega_z(t)x - \omega_z(t)x_c(t)$$
$$u_z(t) = v_{cz}(t) + \omega_x(t)y - \omega_x(t)y_c(t) - \omega_y(t)x + \omega_y(t)x_c(t)$$

Таким образом, можно видеть, что скорости твердого тела вдоль каждого из направлений не зависят от координаты рассматриваемой точки на этом направлении. То есть, например, скорости u_x твердого тела вдоль направления OX не зависят от координаты x, а лишь от положения y и z и от времени t.

В таком случае, при решении уравнения переноса мы можем рассматривать скорость постоянной вдоль каждого из направлений, но зависящей от времени. Поэтому вместо $u_{i-\frac{1}{2}}$, $u_{i+\frac{1}{2}}$ будем рассматривать u, зависящую от времени.

3.5 Пример расчета

Вектор – функции в данной программе задаются с помощью лямбда - выражений на языке программирования C++, что обеспечивает высокую гибкость в настройке программы под любую конфигурацию задачи. Например, вектор - функция скорости центра масс твердого тела может быть задана следующим образом:

```
function<VectorXd(double)> v = [=](double t)->VectorXd {
        VectorXd v(n);
        v(0) = 1;
        v(1) = 50 - 9.81 * t;
        return v;
    };
```

Для демонстрации работы программы было выбрано твердое тело, состоящее из 12 точек, которые в начальный момент времени имели следующие координаты:

Вектор угловой скорости:

```
function<Vector3d(double)> omega = [=](double t)->Vector3d {
    return Vector3d(0, 0, 1);
};
```

Количество шагов и длина шага сетки:

```
int stepsN = 100;
double h = 0.1;
```

Координаты выбранных точек твердого тела записаны в файле output.txt и имеют следующий вид:

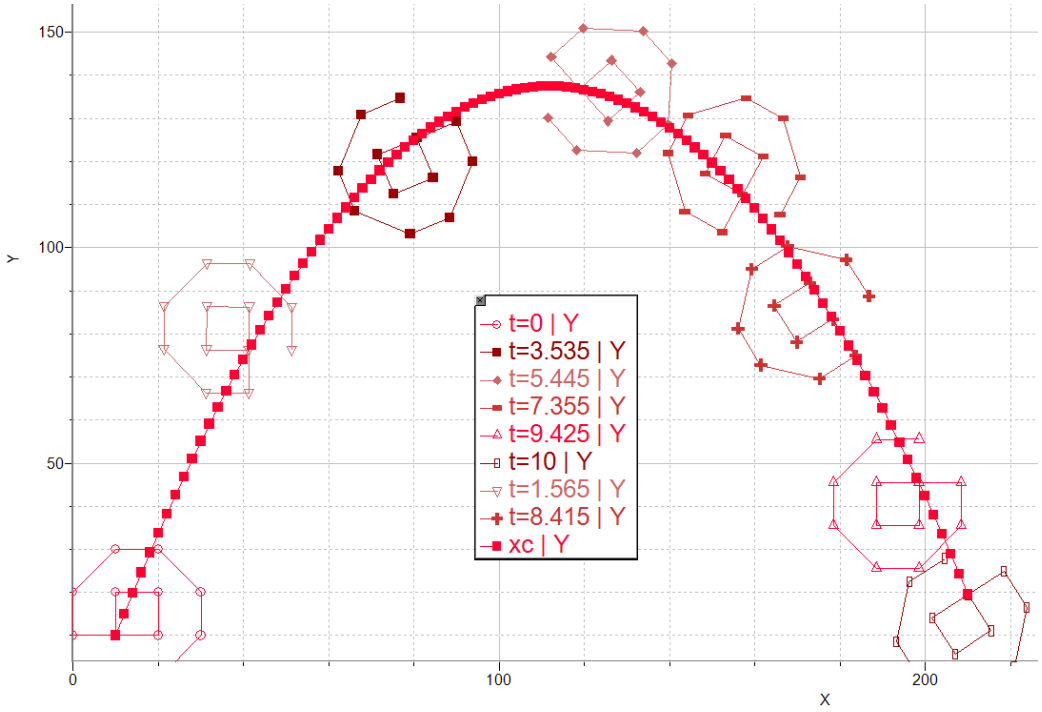

Рис. 1: Пример расчета движения точек твердого тела

Для более наглядной визуализации результатов расчет была реализована программа на языке программирования Python, которая создает анимацию движения твердого тела по предрассчитанным данным о положениях точек. Результат работы данной программы прилагается в файле TestMovement.gif.

3.6 Исследование сходимости

Для расчета ошибки численного моделирования в данном примере было вычислено аналитическое решение.
Ошибка была рассчитана по норме l_2: как среднеквадратичное отклонение по следующей формуле:

$$error_k = \sqrt{\sum_{i=0}^{stepN} ||\overrightarrow{x}^i_k - \overrightarrow{x_{real}}^i_k||^2}$$

Где $\overrightarrow{x_{real}}^i_k$ – положение k-ой точки твердого тела в момент времени $t = i\Delta t$, полученное из аналитического решения.
Ошибка расчета $\overrightarrow{x}_c$ - центра твердого тела:

Среднеквадратичное отклонение при вычислении 4000 шагов на отрезке времени 10с (длина временного шага 0.0025) составляет 2.47756e-10.
Ошибка расчета $\overrightarrow{x}_3$:
Среднеквадратичное отклонение при вычислении 4000 шагов на отрезке времени 10с (длина временного шага 0.0025) составляет 0.000469316.
Следует отметить сходимость рассчитанного решения к аналитическому: при измельчении шага Δx, значение ошибки $error_k$ стремится к 0:
.

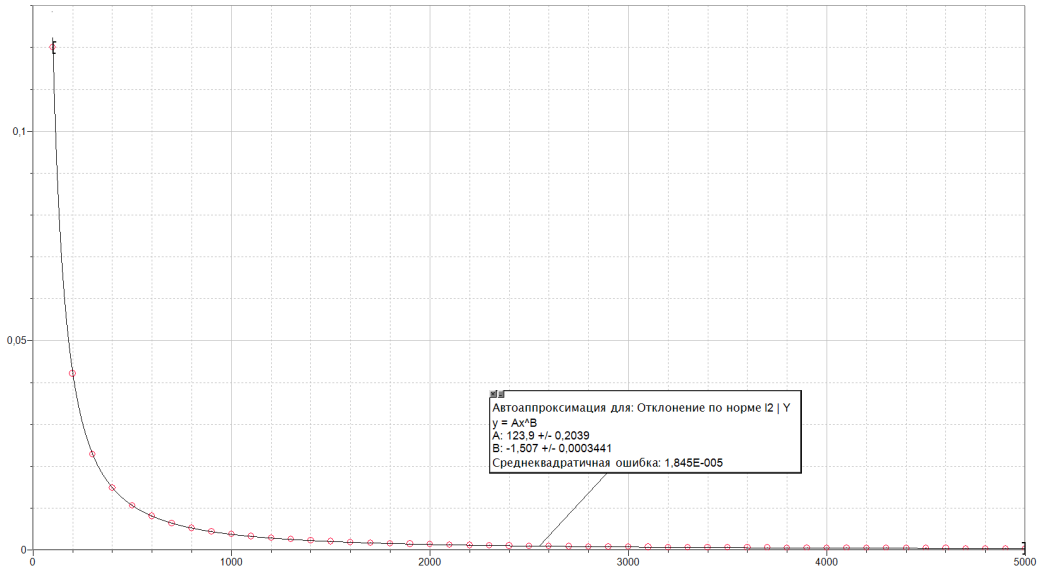

Рис. 2: Сходимость метода DM. Среднеквадратичное отклонение при измельчении шага по времени

4 Численное решение уравнения переноса в одномерном случае со стационарным полем скоростей

4.1 Постановка задачи

Теперь обратимся к решению уравнения переноса. Данное уравнение в частных производных представлено в следующем виде:

$$\frac{\partial f}{\partial t} + \nabla \cdot (\overrightarrow{u} f) - f \nabla \cdot \overrightarrow{u} = 0$$

Где $\overrightarrow{u}$ – векторное поле скоростей, f - переносимая скалярная величина. Определим как функцию Хевисайда, принимающую значения 0 и 1:

$$f(x,t) = \begin{cases} 1 & \mathbf{x} \notin D(t) \\ 0 & \mathbf{x} \in D(t) \end{cases} \tag{11}$$

∇ – оператор дивергенции.
В одномерном случае данная задача рассматривается в виде:

$$\frac{\partial f}{\partial t} + \frac{\partial}{\partial x}(uf) - f\frac{\partial u}{\partial x} = 0$$

Также мы предполагаем, что поле скоростей соленоидально, то есть $\nabla \cdot u = 0$. В этом случае задача рассматривается в виде:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0$$

Начальные условия для уравнения переноса:

- Постоянное и положительное поле скоростей $\overrightarrow{u}$
- Начальное распределение характеристической функции $f\ |_{t=0} = f(x,0)$

Необходимо найти:
Характеристическую функцию f в исследуемой области D на отрезке времени $[0;T]$.

4.2 Численное решение

Дискретизация. Отрезок $[0;X]$, на котором рассматривается данное уравнение, разбивается на *cellCount* последовательных подотрезков, длиной Δx_i каждый – ячейки сетки. i=1..cellCount. Положения $x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}$ являются узлами данной сетки (ребрами ячеек). $\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$. Для реализации программы была выбрана равномерная сетка с ячейками равной длины Δx. Зададим длину временного шага Δt и построим схему для вычисления средних значений функции $f(x,t)$ в каждой ячейке.

$$\overline{f}_i^n = \frac{1}{\Delta x_i} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} f(x,t_n)dx \tag{12}$$

- среднее по ячейке значение функции $f(x,t)$ на i-ом отрезке Δx_i на n-ом временном шаге. В дальнейшем будем обозначать его как просто f_i^n.
u - скорость в каждой ячейке исследуемой области (для твердого тела) на каждом шаге по времени (постоянное положительное поле скоростей).

Численное решение. Рассмотрим уравнение переноса в одномерном случае и при постоянной положительной скорости:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0$$

После интегрирования данного уравнения по времени на временном шаге $[t_n; t_{n+1}]$:

$$(f^{n+1} - f^n) + u\int_{t_n}^{t_{n+1}} \frac{\partial f}{\partial x} dt = 0$$

Представим производную $\frac{\partial f}{\partial x}$ в виде разностной схемы первого порядка:

$$\frac{\partial f}{\partial x} = \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{\Delta x}$$

Для приближенного вычисления интеграла $\int_{t_n}^{t_{n+1}} F dt$ используем квадратурную формулу прямоугольников:

$$\int_{t_n}^{t_{n+1}} F dt = F^{n+\frac{1}{2}}\Delta t$$

После подстановки уравнение примет следующий вид:

$$f_i^{n+1} - f_i^n + \frac{u\Delta t}{\Delta x}(f_{i+\frac{1}{2}}^{n+\frac{1}{2}} - f_{i-\frac{1}{2}}^{n+\frac{1}{2}}) = 0$$

Таким образом, значения характеристической функции f_i^{n+1} на следующем временном шаге определяются как:

$$f_i^{n+1} = f_i^n - \frac{u\Delta t}{\Delta x}(f_{i+\frac{1}{2}}^{n+\frac{1}{2}} - f_{i-\frac{1}{2}}^{n+\frac{1}{2}})$$

Значения f_i^n известны на каждом временном шаге, а для вычисления значений $f_{i+\frac{1}{2}}^{n+\frac{1}{2}}, f_{i-\frac{1}{2}}^{n+\frac{1}{2}}$ используем интерполяционную функцию Ψ:

$$f_{i+\frac{1}{2}}^{n+\frac{1}{2}} = \Psi_i(x_{i+\frac{1}{2}} - u\frac{\Delta t}{2})$$

$$f_{i-\frac{1}{2}}^{n+\frac{1}{2}} = \Psi_{i-1}(x_{i-\frac{1}{2}} - u\frac{\Delta t}{2})$$

Функция $\Psi_i(x)$ строится на каждой ячейке, а $\Psi_{i-1}(x)$ берется из предыдущей ячейки. Конкретное значение интерполяционной функции $\Psi_i(x)$ зависит от выбора схемы или комбинации схем численного решения.

5 Численное решение уравнения переноса в одномерном случае с нестационарным полем скоростей

Уравнение переноса в общем случае имеет следующий вид:

$$\frac{\partial f}{\partial t} + \nabla \cdot (f\overrightarrow{u}) = 0 \tag{13}$$

Где u – векторное поле скоростей, f - переносимая скалярная величина, ∇ – оператор дивергенции. Определим f как функцию Хевисайда, принимающую значения 0 и 1:

$$f(x,t) = \begin{cases} 1 & \mathbf{x} \in liquid \\ 0 & \mathbf{x} \notin liquid \end{cases} \tag{14}$$

В одномерном случае уравнение сводится к виду:

$$\frac{\partial f}{\partial t} + \frac{\partial (fu)}{\partial x} = 0 \tag{15}$$

5.1 Численное решение

Дискретизация. Для численного решения проводится дискретизация:

Отрезок $[0; X]$, на котором рассматривается данное уравнение, разбивается на *cellCount* последовательных подотрезков, длиной Δx_i каждый – ячейки сетки. i=1..cellCount. Положения $x_{i-\frac{1}{2}}$, $x_{i+\frac{1}{2}}$ являются узлами данной сетки (ребрами ячеек). $\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$. Для реализации программы была выбрана равномерная сетка с ячейками равной длины Δx. Зададим длину временного шага Δt и построим схему для вычисления средних значений функции $f(x,t)$ в каждой ячейке.

$$\overline{f}_i^n = \frac{1}{\Delta x_i} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} f(x, t_n) dx \tag{16}$$

- среднее по ячейке значение функции $f(x,t)$ на i-ом отрезке Δx_i на n-ом временном шаге. В дальнейшем будем обозначать его как просто f_i^n.

Численное решение в общем случае. Проинтегрируем уравнение переноса в одномерном случае (15) по времени на шаге $[t_n; t_{n+1}]$:

$$(f^{n+1} - f^n) + \int_{t_n}^{t_{n+1}} \frac{\partial (fu)}{\partial x} d\tau = 0$$

Для численного дифференцирования используем явную разностную схему 2 порядка:

$$\frac{\mathrm{d}(fu)}{\mathrm{d}x} = \frac{f_{i+\frac{1}{2}}u_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}u_{i-\frac{1}{2}}}{\Delta x_i}$$

В результате уравнение переноса преобразуется к виду:

$$(f_i^{n+1} - f_i^n) + \frac{1}{\Delta x_i}\int_{t_n}^{t_{n+1}} f_{i+\frac{1}{2}}u_{i+\frac{1}{2}}d\tau - \frac{1}{\Delta x_i}\int_{t_n}^{t_{n+1}} f_{i-\frac{1}{2}}u_{i-\frac{1}{2}}d\tau = 0 \quad (17)$$

Численное решение для нестационарного поля скоростей твердого тела. Перепишем уравнение (17), полученное для произвольного поля скоростей, для поля скоростей твердого тела:

$$(f_i^{n+1} - f_i^n) + \frac{1}{\Delta x_i}(\int_{t_n}^{t_{n+1}} uf_{i+\frac{1}{2}}d\tau - \int_{t_n}^{t_{n+1}} uf_{i-\frac{1}{2}}d\tau) = 0$$

Интегралы

$$\Phi_{i+\frac{1}{2}} = \int_{t_n}^{t_{n+1}} uf_{i+\frac{1}{2}}d\tau \quad (18)$$

$$\Phi_{i-\frac{1}{2}}\int_{t_n}^{t_{n+1}} uf_{i-\frac{1}{2}}d\tau \quad (19)$$

представляют собой потоки через правую $x_{i+\frac{1}{2}} = x_R$ и левую $x_{i-\frac{1}{2}} = x_L$ грани ячейки $\Delta x_i = \Delta x$ соответственно.

Для вычисления значений $f_{x^*}^{t^*}$ необходимо построить подсеточную реконструкцию решения на соответствующей ячейке. При таком подходе значения $f_{x^*}^{t^*}$ считаются как значения интерполяционной функции, соответствующей методу решения.

$$f_{x^*}^{t^*} = \Psi_0(x^*, t^*)$$

Интерполяционная функция $\Psi_0(x^*, t^*)$:

$$\Psi_0(x^*, t^*) = \begin{cases} \Psi_L(x(x^*, \tau^*)) & \int_{t_n}^{t_{n+1}} u^\tau d\tau \geq 0 \\ \Psi_R(x(x^*, \tau^*)) & \int_{t_n}^{t_{n+1}} u^\tau d\tau < 0 \end{cases} \quad (20)$$

Таким образом, реализуется метод *upwind*: расчета потоков через грани по направлению переноса. То есть, при вычислении потока Φ_{x^*} для положительной скорости u переноса на грани x^* ячейки в течение шага $[t_n; t_{n+1}]$, будет использована интерполяционная функция $\Psi_j(x(x^*, \tau^*))$, построенная на ячейке j слева от данной грани x^*, а при вычислении

потока для отрицательной скорости u будет использована интерполяционная функция $\Psi_j(x(x^*, \tau^*))$, построенная на ячейке j справа от данной грани x^*.

В дальнейшем для выбора интерполяционной функции Ψ_j в исследовании будут рассмотрены такие методы, как метод MUSCL (линейная интерполяция), метод THINC (интерполяция гиперболическим тангенсом) и метод Jump Reconstruction (интерполяция скачком).

Выражение для расчета объемной доли переносимой скалярной величины через потоки в ячейке Δx_i на следующем временном шаге t_{n+1} имеет вид:

$$f_i^{n+1} = f_i^n - \frac{1}{\Delta x_i}(\Phi_{i+\frac{1}{2}} - \Phi_{i-\frac{1}{2}}) = 0 \tag{21}$$

5.2 Метод характеристик

Интерполяционные функции $\Psi_i(x)$ - функции координаты. Для интегрирования по времени необходимо представить их как функции времени. Для этого был использован метод характеристик. Данный метод позволяет определить линии в плоскости (x, t), вдоль которых решение постоянно. Это позволяет доставить решение, выходящее из заданного положения $(x(t), t_n)$ в определенную точку $(x^*, t_n + \tau^*)$ в пространстве (x, t). таким образом, появляется возможность вместо координаты x, в качестве аргумента интерполяционной функции $\Psi_i(x)$ использовать функцию времени $x(t)$.

Рассмотрим решение уравнения переноса с помощью метода характеристик.

$$\frac{\mathrm{d}f}{\mathrm{d}t} + \frac{\mathrm{d}(fu)}{\mathrm{d}x} = 0 \tag{22}$$

Нам бы хотелось свести это дифференциальное уравнение в частных производных первого порядка к обыкновенному дифференциальному уравнению вдоль соответствующей кривой, то есть получить уравнение вида:

$$\frac{\mathrm{d}}{\mathrm{d}s} f(x(s), t(s)) = F(f, x(s), t(s))$$

где кривая $(x(s), t(s))$ — характеристика.
Установим, что

$$\frac{\mathrm{d}}{\mathrm{d}s} f(x(s), t(s)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}s} + \frac{\partial f}{\partial t}\frac{\mathrm{d}t}{\mathrm{d}s} \tag{23}$$

Положим, что

$$\frac{\mathrm{d}t}{\mathrm{d}s} = 1$$

Следовательно, при $t(0) = 0$, $s = t$. И теперь будем составлять ОДУ, используя метод характеристик в виде:

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), t) = F(f, x(t), t)$$

Будем искать решение вдоль характеристик, уравнение которых имеет вид:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = u(t)$$

В таком случае уравнение (23) можно переписать в виде:

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), t) = u(t)\frac{\partial f}{\partial x} + \frac{\partial f}{\partial t}$$

таким образом, вдоль характеристики $(x(t), t)$ исходное уравнение в частных производных превращается в ОДУ:

$$f'_t = F(f, x(t), t) = 0$$

Данное уравнение говорит о том, что вдоль характеристик решение постоянное. Таким образом, $f(x, t) = f(x_0, 0)$, где точки (x, t) и $(x_0, 0)$ лежат на одной характеристике. Видно, что для нахождения общего решения достаточно найти характеристики уравнения в виде:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = u(t) \tag{24}$$

Будем искать решение на временном слое t_n.
τ - время на слое $[t_n; t_{n+1}]$. То есть $\tau = 0 \Leftrightarrow t = t_n$, $\tau = \Delta t \Leftrightarrow t = t_{n+1}$.
Проинтегрируем уравнение (24) по t от t_n до $t_n + \tau$:

$$x(t_n + \tau) - x(t_n) = \int_{t_n}^{t_n+\tau} u(t)dt + C \tag{25}$$

Что является общим видом характеристической функции для данного уравнения в частных производных.
Найдем такую характеристику, которая в момент времени τ^* проходила через точку x^*. Подставим в характеристическую функцию (25) данные начальные условия:

$$x(t_n + \tau^*) - x(t_n) = \int_{t_n}^{t_n+\tau^*} u(t)dt + C^*$$

Тогда

$$C^* = x^* - x(t_n) - \int_{t_n}^{t_n+\tau^*} u(t)dt$$

Подставим C^* обратно в общий вид уравнения характеристической функции (25), чтобы получить характеристику:

$$x(t_n+\tau) - x(t_n) = \int_{t_n}^{t_n+\tau} u(t)dt + x^* - x(t_n) - \int_{t_n}^{t_n+\tau^*} u(t)dt$$

$$x(t_n+\tau) = \int_{t_n}^{t_n+\tau} u(t)dt + x^* - \int_{t_n}^{t_n+\tau^*} u(t)dt \tag{26}$$

Узнаем координату, из которой выходила данная характеристика в момент времени $\tau = 0$:

$$x\mid_{\tau=0} = \int_{t_n}^{t_n} u(t)dt + x^* - \int_{t_n}^{t_n+\tau^*} u(t)dt$$

$$x\mid_{\tau=0} = x^* - \int_{t_n}^{t_n+\tau^*} u(t)dt \tag{27}$$

Данное значение необходимо использовать как аргумент интерполяционной функции (20).

5.3 Аппроксимация интегралов

В зависимости от выбранного метода подсеточной реконструкции, интегралы в потоках и в характеристике могут быть вычислены приближенно, с использованием квадратурных формул в случае непрерывной интерполяции распределения (этот способ использовался для методов MUSCL и THINC), или точно в случае кусочно-постоянной интерполяции распределения (этот способ использовался для метода JR).

Непрерывное распределение. Рассмотрим непрерывную реконструкцию распределения. Для вычисления интегралов была использована квадратурная формула трапеции.
Интеграл в характеристике (27) был заменен на квадратурную формулу:

$$x\mid_{\tau=0} = x^* - \int_{t_n}^{t_n+\tau^*} u(t)dt = x^* - \Delta t\frac{u(t_n) + u(t_n+\tau^*)}{2} \tag{28}$$

Для аппроксимации интегралов потоков (18), (19) была использована квадратурная формула трапеции.
Интегралы в потоках были заменены на квадратурные формулы:

$$\Phi_{i+\frac{1}{2}} = \int_{t_n}^{t_{n+1}} u f_{i+\frac{1}{2}} d\tau = \frac{\Delta t}{2}(u^{n+1} f_{i+\frac{1}{2}}^{n+1} + u^n f_{i+\frac{1}{2}}^n) \tag{29}$$

$$\Phi_{i-\frac{1}{2}} = \int_{t_n}^{t_{n+1}} u f_{i-\frac{1}{2}} d\tau = \frac{\Delta t}{2}(u^{n+1} f_{i-\frac{1}{2}}^{n+1} + u^n f_{i-\frac{1}{2}}^n) \tag{30}$$

Таким образом, значения потоков (18), (19) будут иметь вид:

$$\Phi_{i+\frac{1}{2}} = \frac{\Delta t}{2}(u^n \Psi_0(x(x_{i+\frac{1}{2}}, 0)) + u^{n+1} \Psi_0(x(x_{i+\frac{1}{2}}, \Delta t))) \tag{31}$$

$$\Phi_{i-\frac{1}{2}} = \frac{\Delta t}{2}(u^n \Psi_0(x(x_{i-\frac{1}{2}}, 0)) + u^{n+1} \Psi_0(x(x_{i-\frac{1}{2}}, \Delta t))) \tag{32}$$

Подставляя в них характеристики с координатами соответствующих граней и с нужным шагом по времени, получим окончательный вид уравнений для расчета потоков:

$$\Phi_{i+\frac{1}{2}} = \frac{\Delta t}{2}(u^n \Psi_0(x_R) + u^{n+1} \Psi_0(x^R - \Delta t \frac{u^n + u^{n+1}}{2}))) \tag{33}$$

$$\Phi_{i-\frac{1}{2}} = \frac{\Delta t}{2}(u^n \Psi_0(x_L) + u^{n+1} \Psi_0(x^L - \Delta t \frac{u^n + u^{n+1}}{2}))) \tag{34}$$

Кусочно - постоянное распределение. Метод JR. В силу кусочно - постоянного характера распределения при использовании подхода аппроксимации скачка разрывной функцией (Jump Reconstruction), для данного метода необходим точный расчет интегралов. Найти аналитическое решение в данном случае возможно из-за того же кусочно - постоянного характера распределения.
Рассмотрим случай положительной скорости на временном шаге $[t_n; t_{n+1}]$: $u^n + u^{n+1} \geq 0$. Случай отрицательной скорости рассматривается аналогично.
Интеграл в характеристике (27) также был заменен на квадратурную формулу:

$$x \mid_{\tau=0} = x^* - \int_{t_n}^{t_n+\tau^*} u(t) dt = x^* - \Delta t u^n \tag{35}$$

Подставляя данную формулу в уравнения потоков (18), (19), получим

$$\Phi_{i+\frac{1}{2}} = \int_{t_n}^{t_{n+1}} u f_{i+\frac{1}{2}} d\tau = \int_{t_n}^{t_{n+1}} u \Psi_0(x(x_{i+\frac{1}{2}}, \tau)) d\tau = u^{n+\frac{1}{2}} \int_{t_n}^{t_{n+1}} \Psi_0(x_{i+\frac{1}{2}} - \Delta t u^n) d\tau \tag{36}$$

$$\Phi_{i-\frac{1}{2}} = \int_{t_n}^{t_{n+1}} u f_{i-\frac{1}{2}} d\tau = \int_{t_n}^{t_{n+1}} u\Psi_0(x(x_{i-\frac{1}{2}}, \tau))d\tau) = u^{n+\frac{1}{2}} \int_{t_n}^{t_{n+1}} \Psi_0(x_{i-\frac{1}{2}} - \Delta t u^n)d\tau \tag{37}$$

Метод Jump Reconstruction работает только на разрывных ячейках, поэтому построенная на разрывной ячейке функция $\Psi_i(x)$ будет иметь вид:

$$\Psi_i(x) = \begin{cases} \Psi_+ & x \le x^* \\ \Psi_- & x > x^* \end{cases}$$

Где x^* - положение разрыва в абсолютных координатах, введенных на сетке.
Таким образом, значения потоков (18), (19) будут иметь вид:

$$\Phi_{i+\frac{1}{2}} = u^{n+\frac{1}{2}}(\Psi_i^+ \tau^* + \Psi_i^-(\Delta t - \tau^*)) \tag{38}$$

$$\Phi_{i-\frac{1}{2}} = u^{n+\frac{1}{2}}(\Psi_{i-1}^+ \tau^* + \Psi_{i-1}^-(\Delta t - \tau^*)) \tag{39}$$

Где

$$\tau^* = \frac{x_{i+\frac{1}{2}} - x_i^*}{u^n}$$

Здесь предполагается, что $\tau^* \le \Delta t$. В случае, если $\tau^* > \Delta t$, то

$$\Phi_{i+\frac{1}{2}} = u^{n+\frac{1}{2}} \Psi_i^+ \Delta t \tag{40}$$

$$\Phi_{i-\frac{1}{2}} = u^{n+\frac{1}{2}} \Psi_{i-1}^+ \Delta t \tag{41}$$

Оба случая можно объединить:

$$\Phi_{i+\frac{1}{2}} = u^{n+\frac{1}{2}}(\Psi_i^+ min(\Delta t, \tau^*) + \Psi_i^- max(0, \Delta t - \tau^*)) \tag{42}$$

$$\Phi_{i-\frac{1}{2}} = u^{n+\frac{1}{2}}(\Psi_{i-1}^+ min(\Delta t, \tau^*) + \Psi_{i-1}^- min(0, \Delta t - \tau^*)) \tag{43}$$

6 Исследование различных схем

Функция $\Psi_i(x)$ позволяет интерполировать значения на границах ячеек. Ее выбор определяет вид схемы численного решения: интерполяционной функции в точках разрыва функции – индикатора f, и, соответственно, вид получаемого решения. При выборе функции $\Psi(x)$ были рассмотрены следующие схемы:

- Схема Годунова – схема кусочно-постоянной аппроксимации, имеет 1 порядок точности

- Схема MUSCL - монотонная восходящая схема для законов сохранения, имеет 2 порядок точности
- Схема THINC - гиперболический тангенс для отслеживания поверхности, позволяет существенно снизить эффект «численно вязкости», повышая, таким образом, точность решения.
- Схема JR - восполнение разрывной функцией, Jump Reconstruction

Схемы THINC и JR применяется только при выполнении определенных условий на значения в ячейках сетки. Там, где эти условия не выполняются схему данные схемы комбинируют с другими схемами интерполяции, например со схемой Годунова или MUSCL.

6.1 Метод Годунова(const)

Схема Годунова - численная схема для решения дифференциальных уравнений в частных производных. Данная схема представляет собой консервативный метод конечных объемов, который решает точную или приближенную задачу Римана на границах между ячейками. Метод Годунова имеет точность первого порядка как в пространстве, так и во времени, но может использоваться в качестве базовой схемы для разработки схем более высокого порядка.

При реализации схемы Годунова в качестве функции $\Psi_i(x)$ берется константа – значение f_i^n на том же временном шаге t_n и в той же ячейке Δx_i.

$$\Psi_i(x) = f_i$$

Таким образом, общая формула расчета значений f_i^{n+1} на следующем временном шаге представляет собой:

$$f_i^{n+1} = f_i^n - \frac{u^n + u^{n+1}}{2}(f_i^n - f_{i-1}^n)\frac{\Delta t}{\Delta x}$$

Результаты расчетов с помощью данного метода, после прохождения 1-6 периодов на разных сетках представлены ниже:

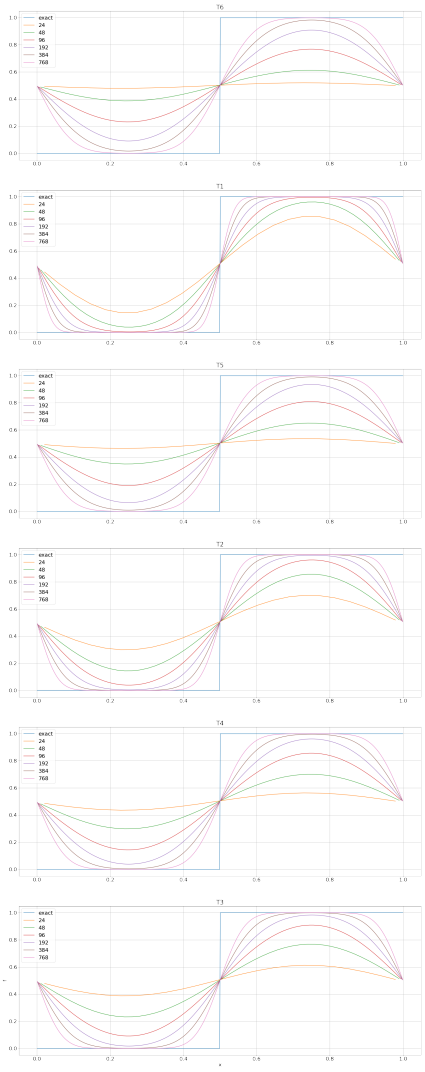

Рис. 3: Расчеты 1D. Схема Годунова

Можно наблюдать значительное "размазывание"значений характеристической функции, которое увеличивается с каждым шагом по времени. Таким образом, проявляется эффект численной вязкости, который силь-

но искажает исходную форму твердого тела, представленного характеристической функцией.

6.2 Метод MUSCL (linear)

Схема MUSCL представляет собой метод конечных объемов и может применяться в случаях, когда система терпит разрывы. MUSCL - Monotonic Upstream-centered Scheme for Conservation Laws (Монотонная восходящая схема для законов сохранения). Данная схема имеет пространственную точность второго порядка. В отличие от схемы Годунова, для расчета потоков на границах ячеек используются не средние значения на ячейках, а значения линейных функций с ограничением наклона по левой и правой ячейке.

При реализации схемы MUSCL в качестве функции $\Psi_i(x)$ берется линейная функция, проходящая через центр ячейки – положение x_i, и имеющая угол наклона, равный значению функции *minmod* от f^n_{iR} и f^n_{iL}.

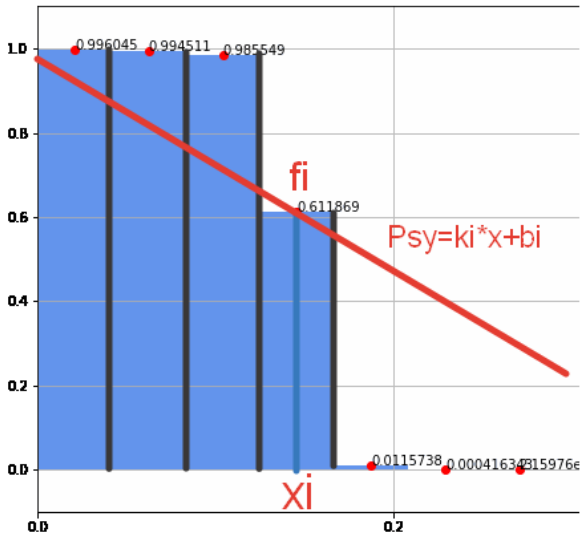

Рис. 4: Схема MUSCL

$$\Psi_i(x) = k^n_i x + b^n_i$$

Где

$$k_i = minmod(f^n_{iR}, f^n_{iL})$$

$$f_{iR} = \frac{f^n_{i+1} - f^n_i}{\Delta x}$$

$$f_{iL} = \frac{f^n_i - f^n_{i-1}}{\Delta x}$$

$$minmod(a,b) = \begin{cases} 0 & ab < 0 \\ sgn(a)min(|a|,|b|) & ab \geq 0 \end{cases}$$

$$b_i^n = f_i^n - k_i^n x_i$$

Результаты расчетов с помощью данного метода, после прохождения 1-6 периодов на разных сетках представлены ниже:

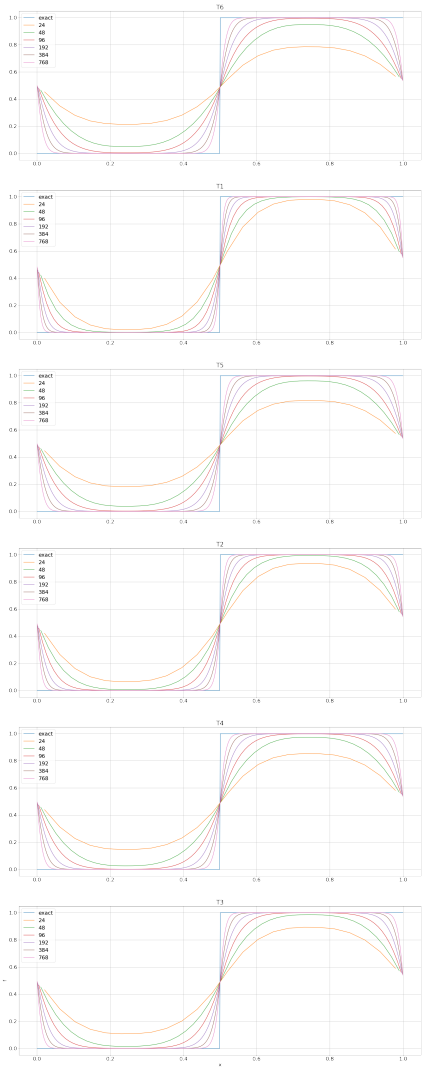

Рис. 5: Расчеты 1D. Схема MUSCL

Можно видеть, что линейный метод MUSCL меньше "размазывает"значения характеристической функции по сравнению с постоянным восполнением метода Годунова.

6.3 Метод THINC (tanh)

Схема THINC - (tangent of hyperbola for interface capturing: гиперболический тангенс для отслеживания поверхности) представляет собой метод конечных объемов.

Данная схема применяется только в ячейках, где выполняется условие:

$$\begin{cases} \epsilon < f_i^n < 1 - \epsilon & \text{то есть, } f_i^n \in (0;1) \\ (f_{i+1}^n - f_i^n)(f_i^n - f_{i-1}^n) & \text{условие монотонности} \end{cases}$$

Где ϵ - малая величина. В расчетах использовалось значение $\epsilon = 1e-4$. В тех ячейках, где данное условие не выполняется, используется схема Годунова или MUSCL.

Для аппроксимации функции на каждом отрезке строится функция:

$$F_i(x) = \frac{1}{2}(1 + \gamma_i tanh(\beta(\frac{x - x_{i-\frac{1}{2}}}{\Delta x} - x_i^*)))$$

Где параметры β, γ_i определяются следующим образом:

$$\gamma_i = sgn(f_{i+1}^n - f_{i-1}^n)$$

β определяет сжатие по оси X – скорость прыжка.

Параметр x_i^* – относительное расстояние до середины прыжка f от левой границы отрезка $x_{i-\frac{1}{2}}$.

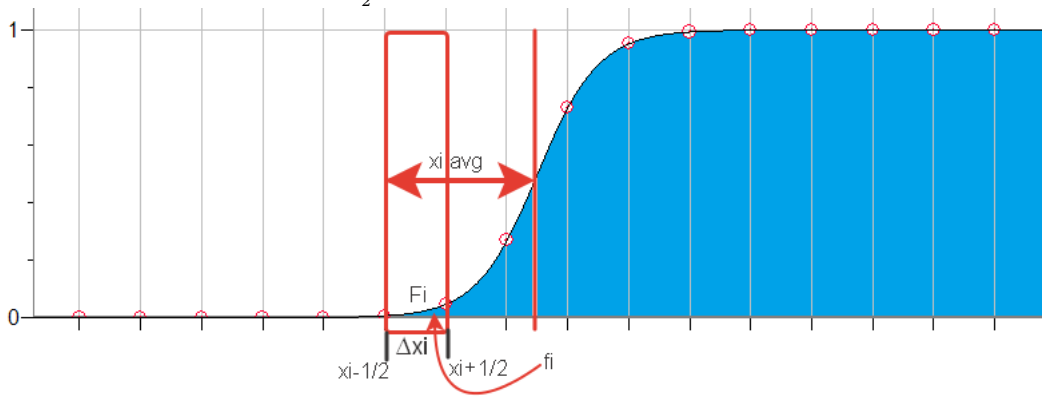

Рис. 6: Схема THINC. x_i^*.

Тогда интерполяционная функция для метода THINC строится нормировкой функции $F_i(x)$ на значения $f_{i\pm1}^n$ следующим образом:

$$\Psi_i(x) = f_{min} + F_i(x)\Delta f_i$$

Где

$$f_{min} = min(f_{i-1}^n, f_{i+1}^n)$$
$$f_{max} = max(f_{i-1}^n, f_{i+1}^n)$$
$$\Delta f_i = f_{max} - f_{min}$$

Здесь $F_i(x)$ - это функция гиперболического тангенса, аппроксимирующая f(x) на ячейке Δx_i.

x_i^* определяется из интегрального уравнения:

$$f_i^n = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Psi_i(x) dx$$

Аналитическое решение данного уравнения:

$$f_i^n = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} (f_{min} + \frac{1}{2}(1 + \gamma_i tanh(\beta(\frac{x - x_{i-\frac{1}{2}}}{\Delta x} - x_i^*)))\Delta f_i) dx$$

$$x_i^* = \frac{1}{2\beta} log \left(\frac{exp(\frac{\beta}{\gamma_i}(1 + \gamma_i - 2\frac{f_i^n - f_{min}}{\Delta f_i})) - 1}{1 - exp(\frac{\beta}{\gamma_i}(1 - \gamma_i - 2\frac{f_i^n - f_{min}}{\Delta f_i}))} \right)$$

Таким образом, общий алгоритм построения на ячейке выглядит следующим образом:

1. По значениям f_{i-1}^n и f_{i+1}^n рассчитываются значения γ_i, f_{min}, f_{max}, Δf_i.

2. По значениям β, $gamma_i$, f_i^n, f_{min}, Δf_i вычисляется значение x_i^*, необходимое для задания функции $F_i(x)$.

3. Теперь готово все необходимое для задания функции $\Psi_i(x) = f_{min} + F_i(x)\Delta f_i$ на отрезке Δx_i.

Результаты расчетов с помощью комбинации схемы THINC и схемы Годунова, после прохождения 1-6 периодов на разных сетках представлены ниже:

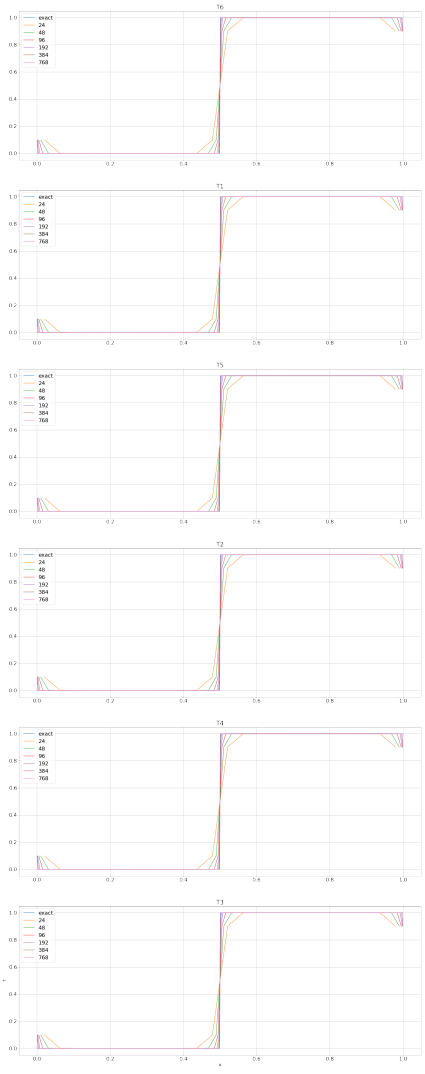

Рис. 7: Расчеты 1D. Схема THINC + Годунов

Результаты расчетов с помощью комбинации схемы THINC и схемы MUSCL, после прохождения 1-6 периодов на разных сетках представлены ниже:

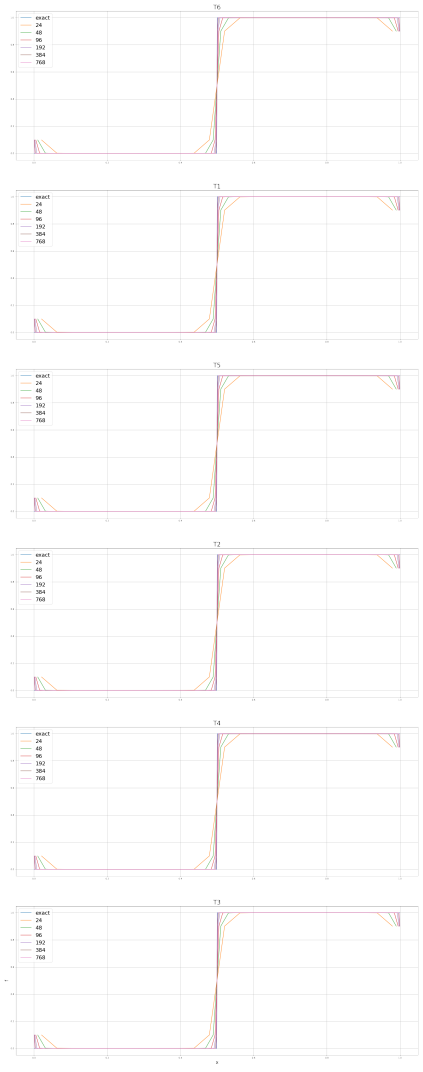

Рис. 8: Расчеты 1D. Схема THINC + MUSCL

Можно наблюдать значительное, по сравнению с классическими методами, уменьшение "размазывание"значений характеристической функции.

6.4 Метод JR (Jump Reconstruction)

Метод Jump Reconstruction позволяет в точности восстановить значения характеристической функции благодаря аппроксимации скачка разрывной функцией.

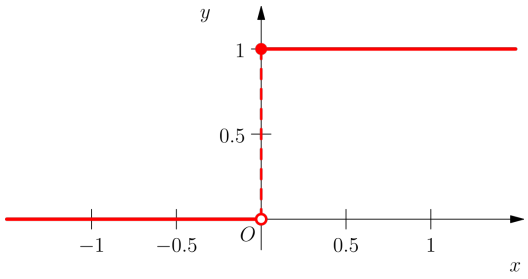

Рис. 9: Разрывная функция

Интерполяционная функция $\Psi_i(x)$ строится на тех же ячейках, где выполняются условия схемы THINC:

$$\begin{cases} \epsilon < f_i^n < 1 - \epsilon & \text{то есть, } f_i^n \in (0;1) \\ (f_{i+1}^n - f_i^n)(f_i^n - f_{i-1}^n) & \text{условие монотонности} \end{cases}$$

Где ϵ - малая величина. В расчетах использовалось значение $\epsilon = 1e-4$. В тех ячейках, где данное условие не выполняется, используется схема Годунова или MUSCL.
Для аппроксимации функции на каждом отрезке строится функция:

$$F_i(x) = \frac{(1-\gamma_i)}{2} + \gamma_i H(x - x_i^*)$$

Где $H(x)$ - функция Хевисайда.
Параметр x_i^* – положение прыжка f. x_i^* определяется из интегрального уравнения:

$$f_i^n = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Psi_i(x) dx$$

Аналитическое решение данного уравнения:

$$f_i^n = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} (f_{min} + \frac{(1-\gamma_i)}{2} + \gamma_i H(x - x_i^*))\Delta f_i) dx$$

$$x_i^* = x_{i+\frac{1}{2}} - \Delta x_i (\frac{f_i - f_{min}}{\Delta f_i \gamma_i} - \frac{1-\gamma_i}{2\gamma_i})$$

Таким образом, общий алгоритм построения на ячейке выглядит следующим образом:

1. По значениям f_{i-1}^n и f_{i+1}^n рассчитываются значения γ_i, f_{min}, f_{max}, Δf_i.

2. По значениям $gamma_i$, f_i^n, f_{min}, Δf_i вычисляется значение x_i^*, необходимое для задания функции $F_i(x)$.

3. Теперь готово все необходимое для задания функции $\Psi_i(x) = f_{min} + F_i(x)\Delta f_i$ на отрезке Δx_i.

Однако для вычисления интегралов потоков при данном методе с кусочно - постоянным восполнением необходим точный расчет интегралов, описанный в разделе (5.3).

Результаты расчетов с помощью комбинации схемы JR и схемы Годунова, после прохождения 1-6 периодов на разных сетках представлены ниже:

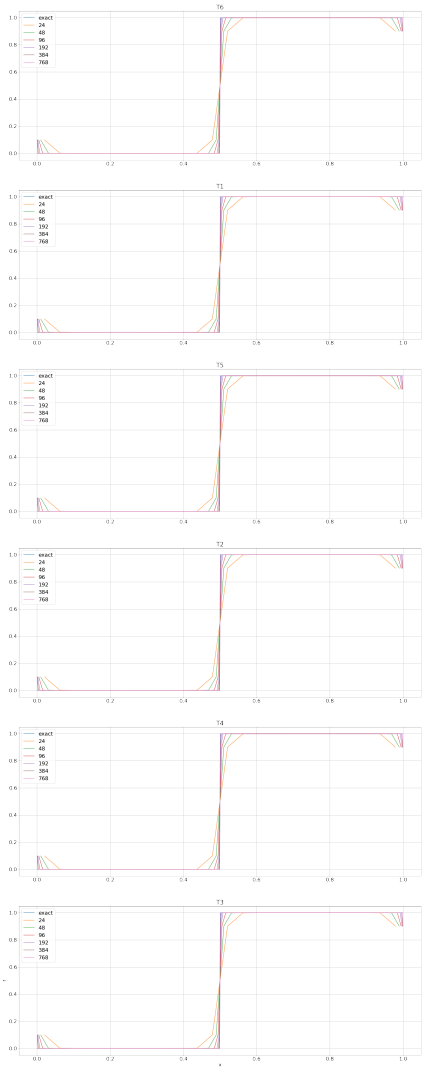

Рис. 10: Расчеты 1D. Схема JR + Годунов

Таким образом, при использовании метода Jump Reconstruction в одномерном случае на каждом временном шаге не происходит "размазывания"значений, и присутствует единственная ячейка с переходными зна-

чениями объемных долей. Таким образом, решение в одномерном случае получается точным.

6.5 Тестовые расчеты

Таким образом, программа для расчета движения строится следующим образом:
Начальные условия – значения f_i^0, которые аппроксимирую скалярную величину f заданы в массиве.

Рис. 11: Начальные условия для тестовых расчетов

Исследуемая область устроена таким образом, что последняя ячейка $i = cellCount - 1$ - предыдущая для первой ячейки i, и наоборот. То есть вся область представляет собой кольцо, заданы периодические граничные условия. Количество ячеек подобрано таким образом, чтобы «планка» проходила полный круг за целое число шагов по времени. Тогда после прохождения каждого периода точное решение будет соответствовать начальным условиям. Размер ячеек постоянен и равен соответственно:

$$\Delta x = \frac{L}{N}$$

Шаг по времени выбирается в соответствие с условием Куранта по формуле:

$$\Delta t = CFL\frac{\Delta x}{u}$$

Тогда для прохождения одного периода потребуется T шагов по времени:

$$T = \frac{N}{CFL}$$

Все расчеты производились с числом CFL=0.3. Это позволило убрать нежелательные осцилляции в методе THINC при расчете рядом с границей скачка. Количество ячеек сетки N тогда было выбрано: [24, 48,

96, 192, 384, 768]. При расчетах использовалась скорость u=0.1. На каждом шаге по времени происходит вычисление новых значений f_i^{n+1} через потоки, которые рассчитываются в соответствии с выбранным методом восполнения.
Функция $\Psi_i(x)$ выбирается на каждой ячейке в соответствии со схемой расчетов. Для каждого следующего отрезка Δx_{i+1} мы сохраняем функцию $\Psi_i(x)$, и используем ее в качестве $\Psi_{i-1}(x)$.
Таким образом, мы делаем шаг по времени для всех отрезков Δx_i.

Пример программной реализации на языке программирования C++ численного решения уравнения переноса в одномерном случае находится в Приложении [9.4].
Результаты расчетов были визуализированы с помощью программы на языке Python.
Примеры расчетов для используемых методов на сетках с разным разрешением:

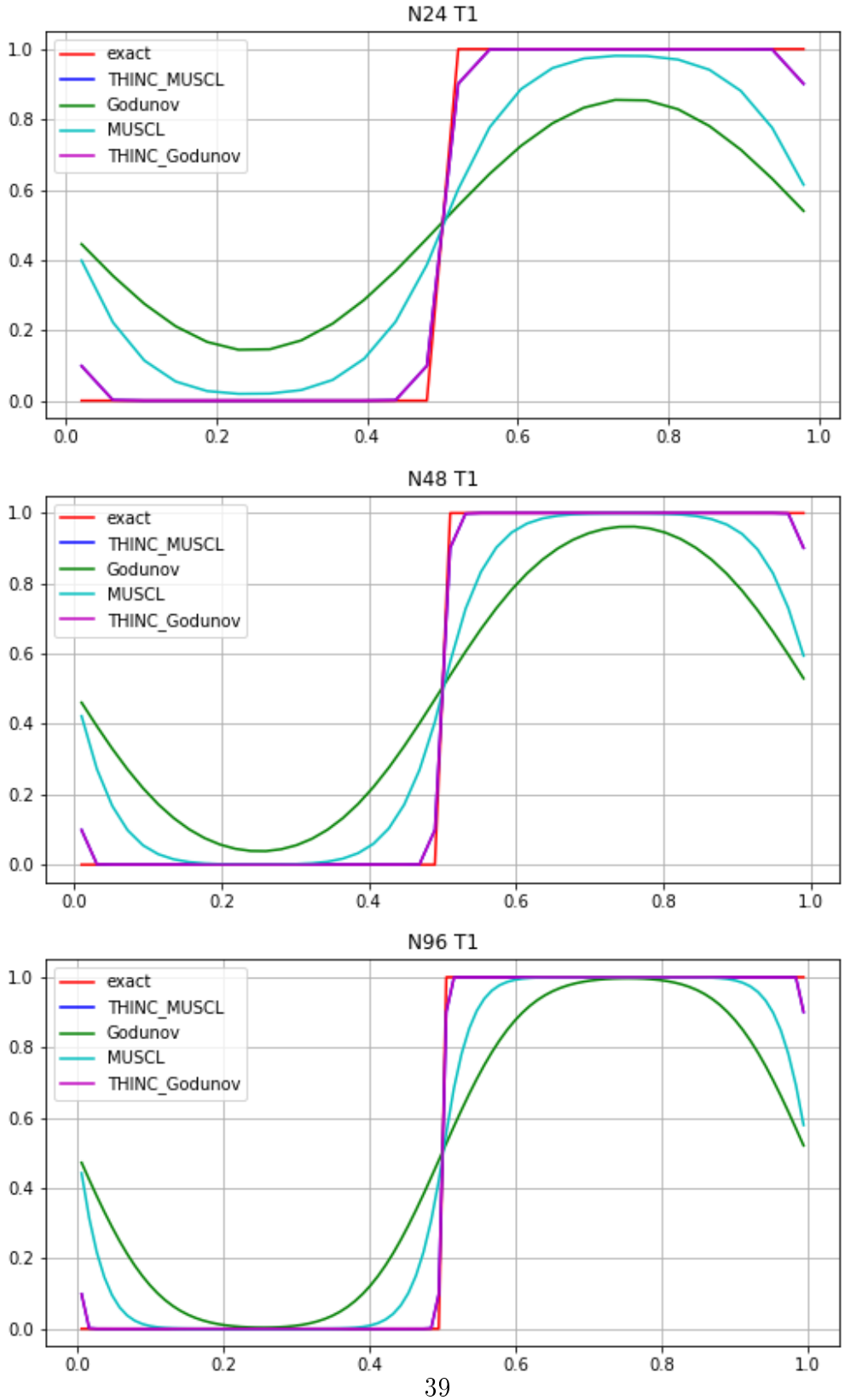

Рис. 12: Расчеты 1D движения планки различными методами. Число ячеек 24, 48, 96. Прохождение 1 периода.

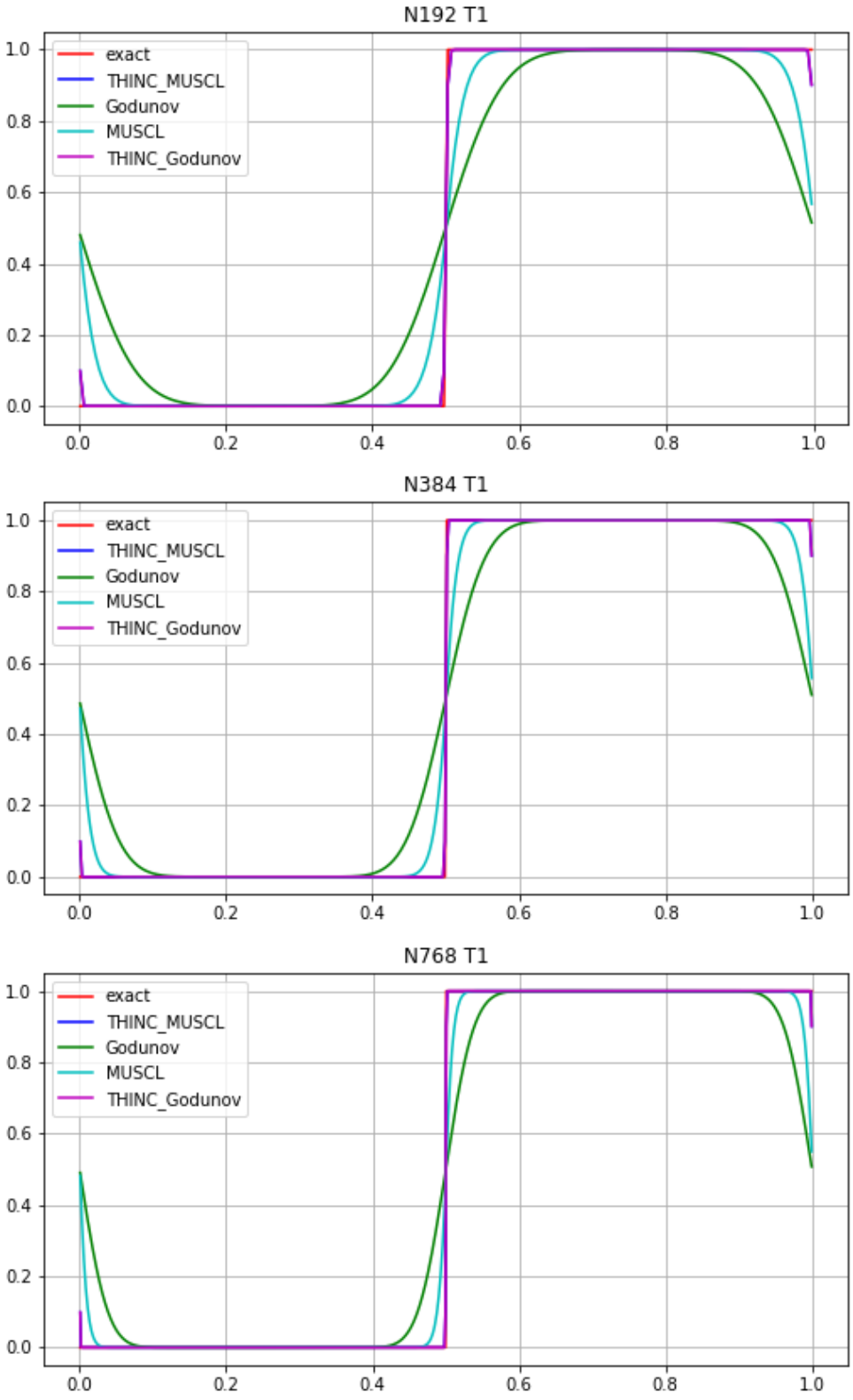

Рис. 13: Расчеты 1D движения пленки различными методами. Число ячеек 192, 384, 768. Прохождение 1 периода.

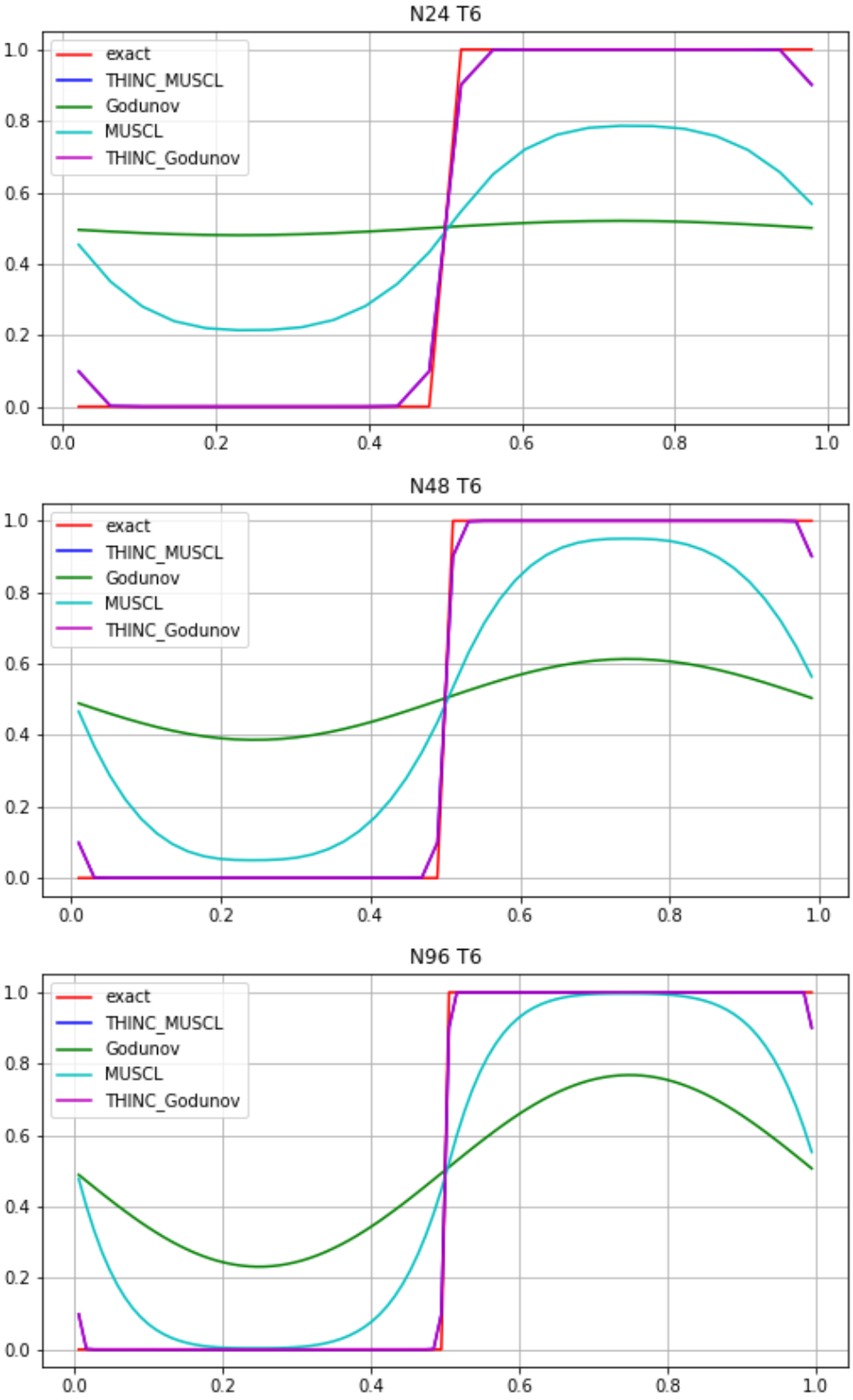

Рис. 14: Расчеты 1D движения планки различными методами. Число ячеек 24, 48, 96. Прохождение 6 периодов.

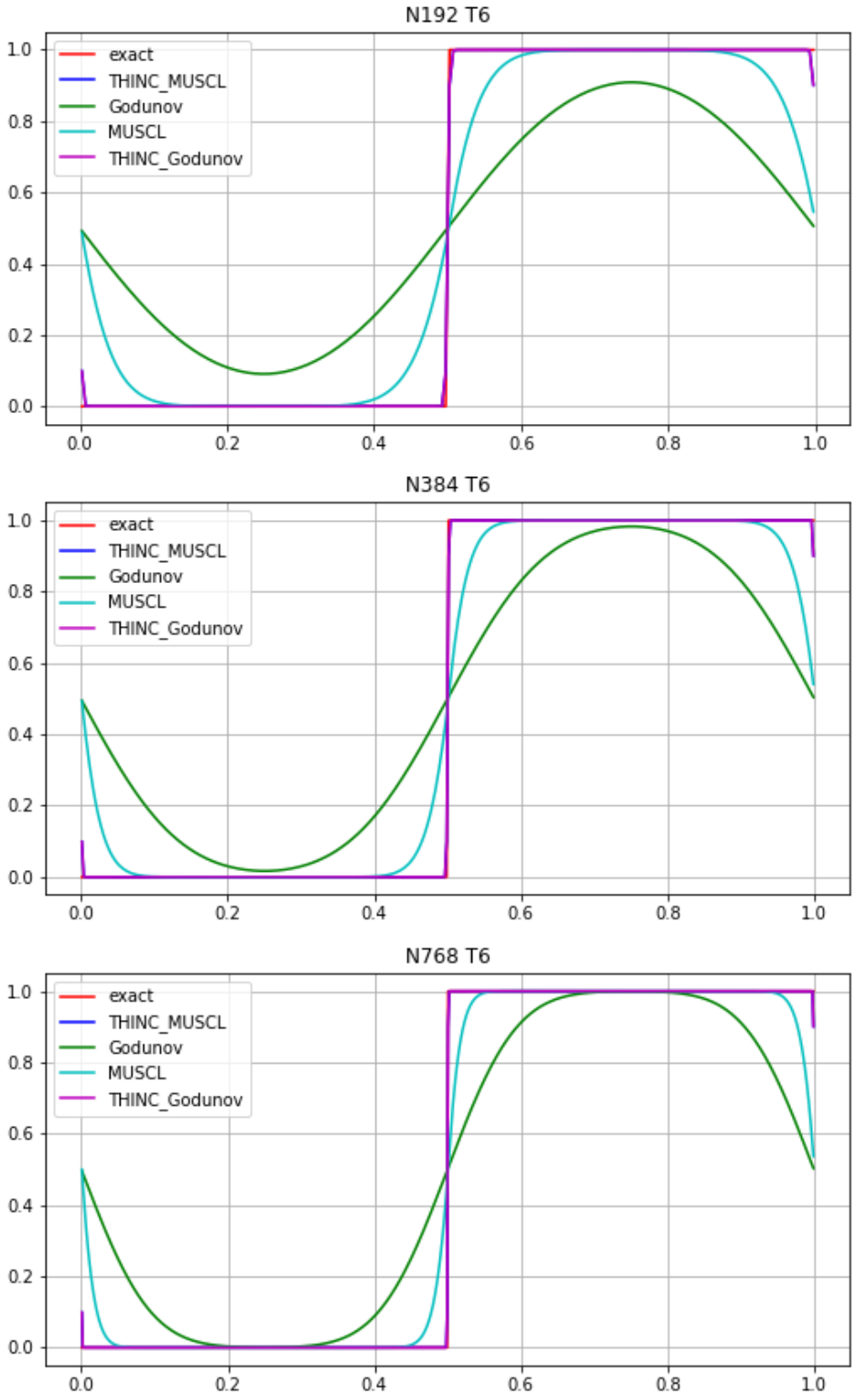

Рис. 15: Расчеты 1D движения планки различными методами. Число ячеек 192, 384, 768. Прохождение 642 периодов.

6.6 Исследование сходимости

Расчет ошибки Для всех вычислений была рассчитана ошибка по норме l_2 по формуле:

$$error_{l_2}(N,T) = \sqrt{\sum_{i=0}^{N-1}((f_i^T - f_i^{exact})\Delta x)^2}$$

Где $f_i^{exact} = f_i^0$ (так как планка проходит целое число периодов).

Данная ошибка была вычислена после прохождения T=[1..6] периодов на сетках с N=[24, 48, 96, 192, 384, 768] ячейками. Соответствующие таблицы ошибок для всех методов представлены ниже:

	T1	T2	T3	T4	T5	T6
N24	0.0595699	0.0768529	0.0876215	0.0938853	0.0974552	0.0994732
N48	0.0343195	0.0422635	0.0489433	0.0543952	0.0586747	0.0619692
N96	0.0203693	0.024306	0.0272539	0.0299097	0.0323704	0.0346239
N192	0.0121176	0.0144143	0.015962	0.0171938	0.0182741	0.0192768
N384	0.00720701	0.00857172	0.00948658	0.0101944	0.0107809	0.0112883
N768	0.00428586	0.0050971	0.00564099	0.0060617	0.00640951	0.00670845

Таблица 1: Ошибка в схеме Годунова

	T1	T2	T3	T4	T5	T6
N24	0.0392346	0.0445669	0.0490333	0.0534235	0.0576686	0.0616839
N48	0.0223648	0.0253282	0.0271637	0.028482	0.0295385	0.0304656
N96	0.0126646	0.0143063	0.0153512	0.0161337	0.0167661	0.0173002
N192	0.00715315	0.00806674	0.00864907	0.00908569	0.00943849	0.00973635
N384	0.00403337	0.00454284	0.00486817	0.00511226	0.0053096	0.00547623
N768	0.00227142	0.00255613	0.00273812	0.00287473	0.00298521	0.00307853

Таблица 2: Ошибка в схеме MUSCL

	T1	T2	T3	T4	T5	T6
N24	0.00823062	0.00823302	0.00823357	0.00823392	0.00823399	0.00823399
N48	0.0041157	0.00411636	0.00411636	0.00411657	0.00411685	0.00411681
N96	0.00205787	0.00205821	0.00205827	0.00205829	0.00205829	0.00205843
N192	0.0010291	0.00102914	0.00102914	0.00102922	0.00102921	0.00102922
N384	0.000514571	0.000514608	0.000514609	0.000514604	0.000514602	0.000514604
N768	0.000257304	0.000257302	0.000257302	0.000257301	0.000257302	0.000257302

Таблица 3: Ошибка в схеме THINC + Годунов

	T1	T2	T3	T4	T5	T6
N24	0.00823047	0.00823123	0.00823164	0.00823161	0.00823284	0.00823292
N48	0.00411565	0.00411571	0.00411591	0.00411609	0.00411618	0.00411591
N96	0.00205785	0.00205808	0.00205803	0.0020579	0.00205794	0.00205799
N192	0.00102904	0.00102895	0.00102899	0.00102916	0.00102914	0.00102913
N384	0.000514474	0.000514581	0.000514565	0.000514566	0.000514574	0.000514576
N768	0.000257291	0.000257283	0.000257288	0.000257299	0.000257295	0.000257293

Таблица 4: Ошибка в схеме THINC + MUSCL

Схема JR в одномерном случае дает около нулевую ошибку, так как восстанавливает решение в точности.

Из полученных данных можно сделать вывод, что использование схемы THINC дает значительное увеличение точности по сравнению с методом Годунова или MUSCL. Схема THINC позволяет на порядок уменьшить величину ошибки. Сравнивая комбинацию методов THINC + Годунов и THINC+MUSCL, можно заметить очень маленькую разницу в результатах расчетов и величинах ошибки (менее 1%). Такой результат получается вследствие того, что вычисляемая функция $\Psi_i(x)$ в тех ячейках, где значения f_i^n близки к 0 или 1 (используется MUSCL), получалась практически горизонтальной прямой, то есть самим значением f_i^n, как в схеме Годунова. Таким образом, нет значимой разницы в комбинировании схемы THINC со схемой Годунова или MUSCL для текущей реализации решения задачи.

Исследование сходимости. Полагая, что метод имеет порядок сходимости β, если $\exists \alpha \in (0; 1]$ такое что выполняется неравенство $error_{l_2}(N, T) \leq \alpha \Delta x^{\beta}$, было проведено исследование сходимости. Приведя неравенство к виду $ln(error_{l_2}(N, T)) \leq \beta ln(\Delta x) + const$, построены графики зависимости логарифма ошибки к логарифму размера ячейки. Данные графики

аппроксимируются прямыми, угол наклона которых – это порядок аппроксимации β.
Построенные графики с рассчитанными значениями порядка сходимости представлены ниже:

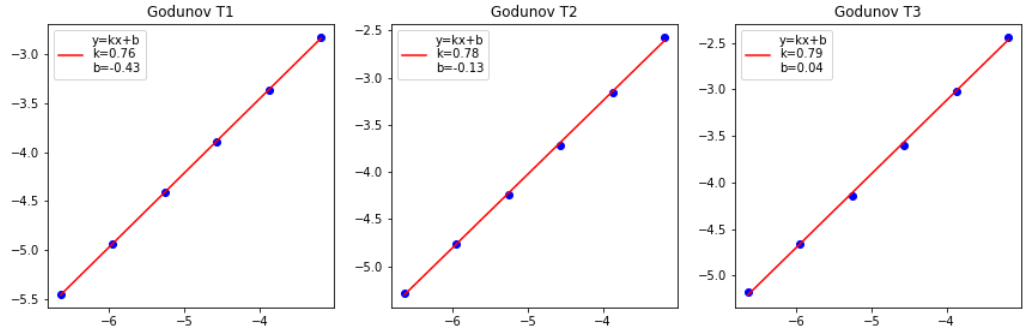

Рис. 16: Графики сходимости для схемы Годунова. Период 1-3

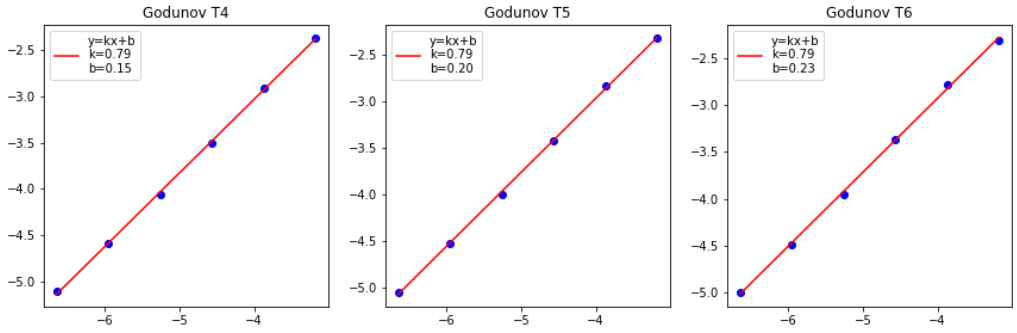

Рис. 17: Графики сходимости для схемы Годунова. Период 4-6

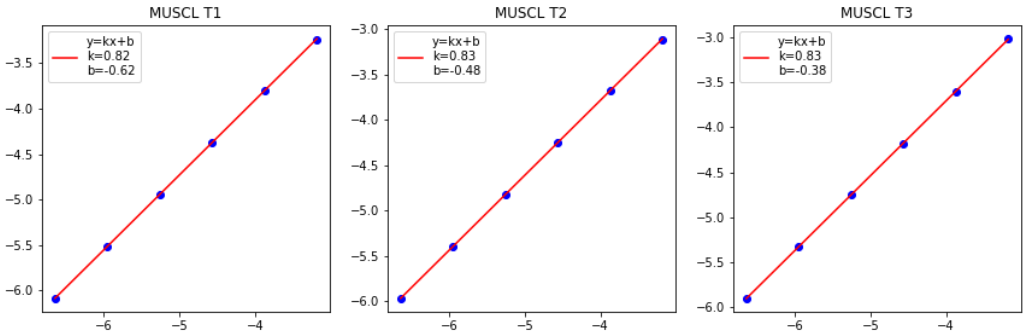

Рис. 18: Графики сходимости для схемы MUSCL. Период 1-3

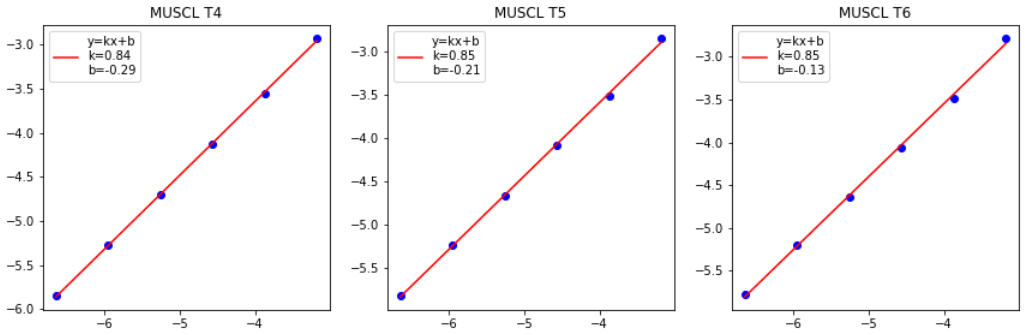

Рис. 19: Графики сходимости для схемы MUSCL. Период 4-6

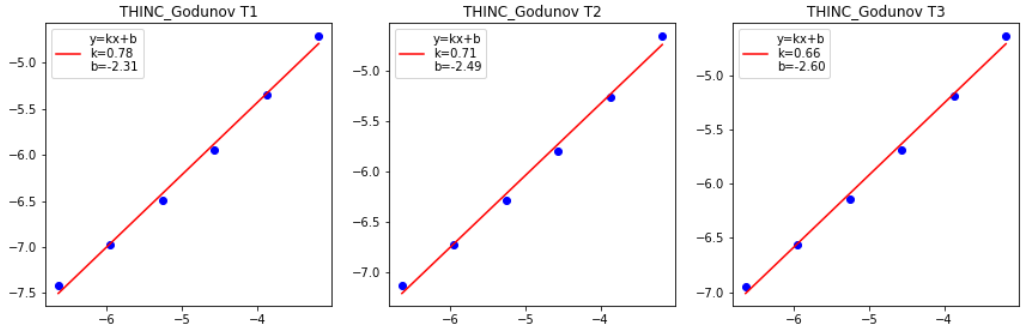

Рис. 20: Графики сходимости для схемы THINC + Годунов. Период 1-3

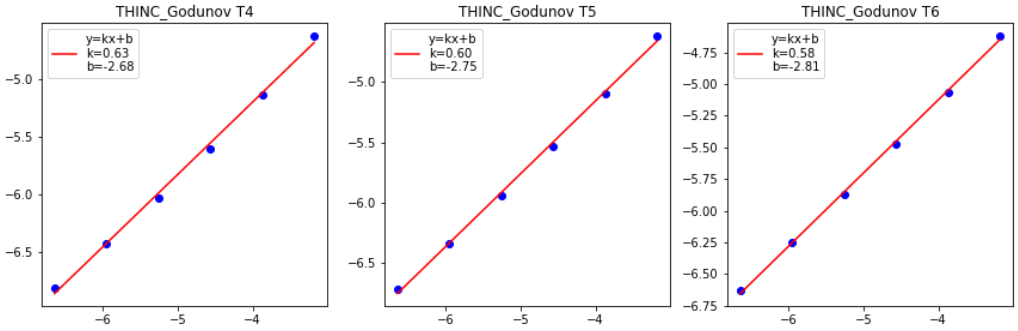

Рис. 21: Графики сходимости для схемы THINC + Годунов. Период 4-6

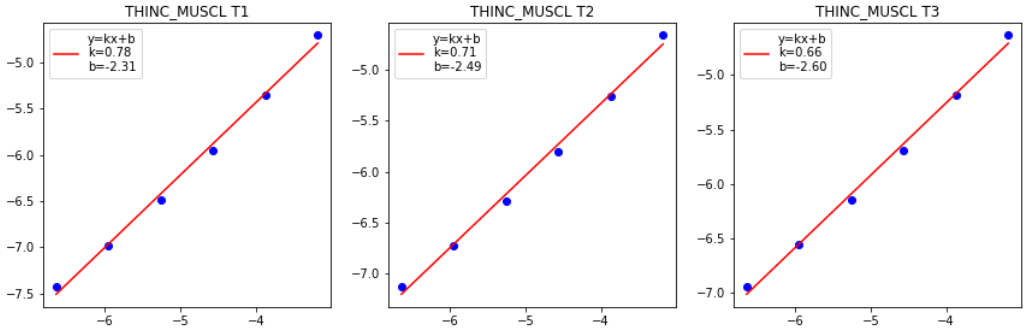

Рис. 22: Графики сходимости для схемы THINC + MUSCL. Период 1-3

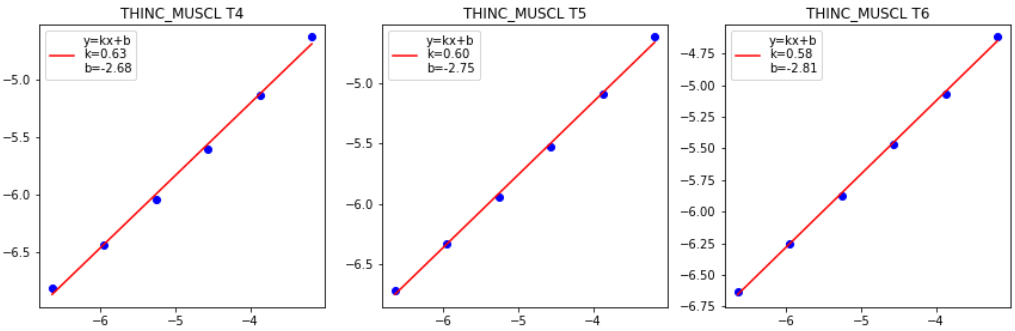

Рис. 23: Графики сходимости для схемы THINC + MUSCL. Период 4-6

7 Многомерное обобщение решения уравнения переноса с нестационарным полем скоростей

Алгоритм численного решения уравнения переноса для случая двух или трех измерений может быть построен на основе численного решения уравнения переноса в одномерном случае. Исходя из вида данного уравнения в частных производных, есть возможность рассчитать решение в проекции на одном из направлений, при заданном положении данной проекции по другим направлениям.

Уравнение переноса в общем случае имеет вид (13):

$$\frac{\partial f}{\partial t} + \nabla \cdot (f \overrightarrow{u}) = 0 \tag{44}$$

В разделе (5.1) проводится численное решение данного уравнения в одномерном случае (21) в проекции на одну из осей. Получена алгебраическая

формула для нахождения значения скалярной величины на следующем временном шаге через потоки на гранях ячеек.

$$f_i^{n+1} = f_i^n - \frac{1}{\Delta x}(\Phi_{i+\frac{1}{2}} - \Phi_{i-\frac{1}{2}}) = 0 \tag{45}$$

Данное уравнение можно записать для каждого из направлений X,Y,Z на введенной сетке при фиксированном положении на других осях:

$$f_i^{n+1} = f_i^n - \frac{1}{\Delta x}(\Phi_{i+\frac{1}{2}} - \Phi_{i-\frac{1}{2}}) = 0 \tag{46}$$

, $i = 1..N$, $j, k = const$

$$f_j^{n+1} = f_j^n - \frac{1}{\Delta x}(\Phi_{j+\frac{1}{2}} - \Phi_{j-\frac{1}{2}}) = 0 \tag{47}$$

, $j = 1..M$, $i, k = const$

$$f_k^{n+1} = f_k^n - \frac{1}{\Delta x}(\Phi_{k+\frac{1}{2}} - \Phi_{k-\frac{1}{2}}) = 0 \tag{48}$$

, $k = 1..P$, $i, j = const$ Скорость u, содержащаяся в вычислении потоков Φ постоянна на всех ячейках каждого из направлений при фиксированном положении на других направлениях, поэтому может быть использована аналогично одномерному случаю для каждого из направлений. Эта особенность характерна для поля скоростей твердого тела и описана в разделе (3.4).

7.1 Расщепление по направлениям

Метод расщепления по направлениям заключается в том, что исходная задача решается в проекции на одно из направлений на сетке при фиксированных положениях данного направления по другим осям сетки. Затем полученное решение используется в качестве начальных условий для расчета по следующему направлению. В результате, для каждой ячейки сетки будут учитываться потоки на всех ее гранях.

То есть, в двумерном случае, скалярная величина, содержащаяся в каждая ячейке, представляющей собой квадрат, будет пересчитываться 2 раза. Сначала рассчитываются потоки на первой паре противоположных граней, затем, поверх изменений после прохождения по одному из направлений, на второй паре противоположных граней. При этом необходимо рассчитать M x N ячеек. То есть одномерный модуль численного алгоритма решения будет запущен M раз для каждого направления X (при каждом фиксированном значении по направлению Y) и N раз для каждого направления Y (при каждом фиксированном значении по направлению X). Данный алгоритм может быть записан в виде:

1. Решение одномерного уравнения переноса на шаге по времени Δt_{n+1} по X направлению M раз (при Y = 1..M) при начальных условиях f^n.

2. Полученное решение f'^{n+1} является начальными условиями для решения по направлению Y.

3. Решение одномерного уравнения переноса на шаге по времени Δt_{n+1} по Y направлению N раз (при X = 1..N) при начальных условиях f'^{n+1}.

4. Полученное решение является решением для шага Δt_{n+1}: $f''^{n+1} = f^{n+1}$ и используется в качестве начальных условий для следующего шага по времени.

Таким образом, была реализована программа решения уравнения переноса в многомерном случае на основе модуля расчета уравнения переноса в одномерном случае. Основой данного алгоритма является расщепление по направлениям, при этом используется одномерный модуль программы, описанный в разделе (5.1).

7.2 Тестовые расчеты

В качестве тестовых вычислений был проведен расчет движения твердого тела в поле стационарной и нестационарной скорости с элементами поступательного и вращательного движения.

Начальные условия для движения тела в поле стационарной скорости:

$$\overrightarrow{u}(t) = (0.1, 0.1)$$

$CFL = 0.3$
$areaWidth = 1.0$
$areaHeight = 1.0$
$cellCountX = 128$
$cellCountY = 128$
$NTimeSteps = 400$

Результаты расчетов для схемы Годунова:

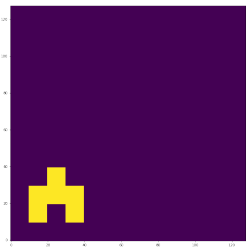

Рис. 24: t=0

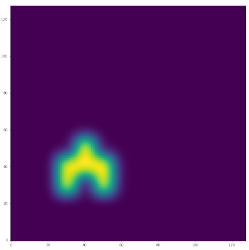

Рис. 25: t=80

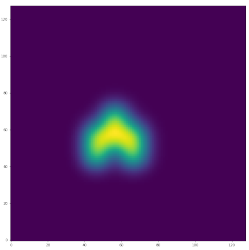

Рис. 26: t=160

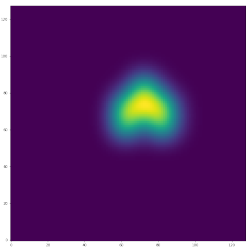

Рис. 27: t=240

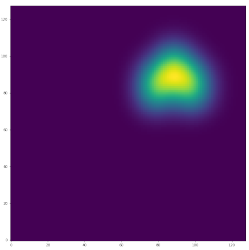

Рис. 28: t=320

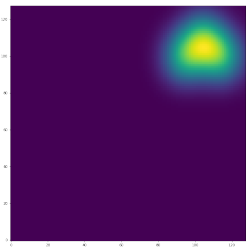

Рис. 29: t=400

Результаты расчетов для схемы MUSCL:

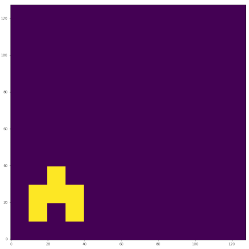

Рис. 30: t=0

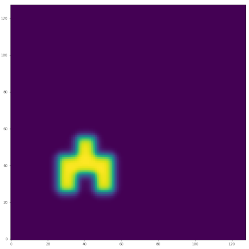

Рис. 31: t=80

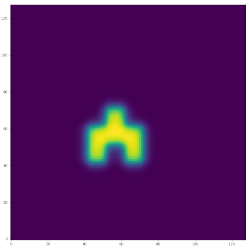

Рис. 32: t=160

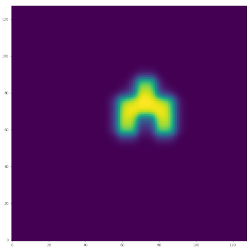

Рис. 33: t=240

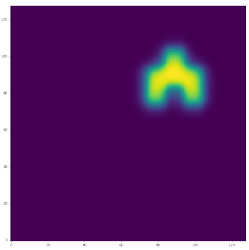

Рис. 34: t=320

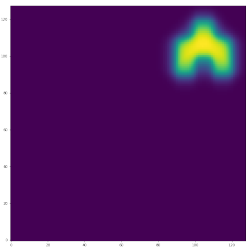

Рис. 35: t=400

Результаты расчетов для схемы THINC:

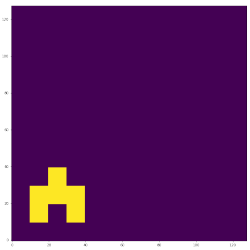

Рис. 36: t=0

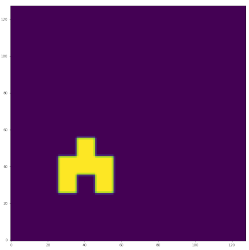

Рис. 37: t=80

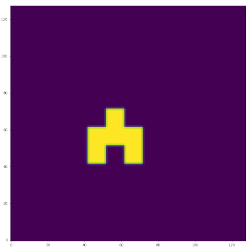

Рис. 38: t=160

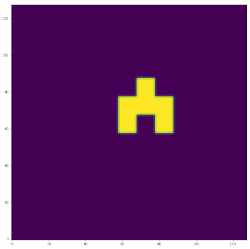

Рис. 39: t=240

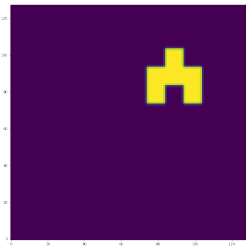

Рис. 40: t=320

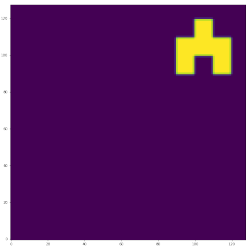

Рис. 41: t=400

Результаты расчетов для схемы JR:

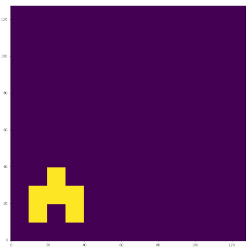

Рис. 42: t=0

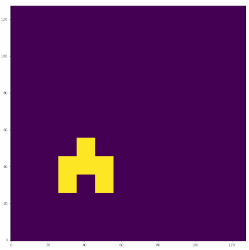

Рис. 43: t=80

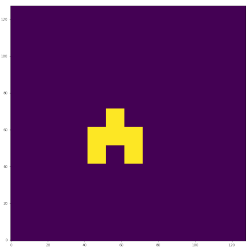

Рис. 44: t=160

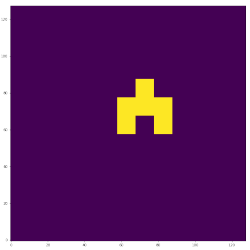

Рис. 45: t=240

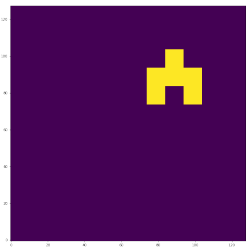

Рис. 46: t=320

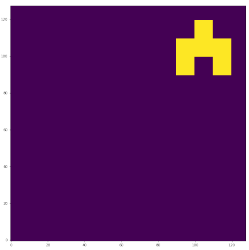

Рис. 47: t=400

Начальные условия для движения тела в нестационарном поле скорости твердого тела:

$$\overrightarrow{v}_c(t) = (1, 4 - 8t)$$

$$\overrightarrow{\omega}_c(t) = (0, 0, \pi)$$

$areaWidth = 2.0$
$areaHeight = 2.0$
$cellCountX = 128$
$cellCountY = 128$
$NTimeSteps = 512$

Результаты расчетов для схемы MUSCL:

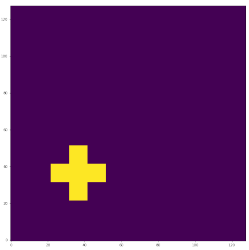

Рис. 48: t=0

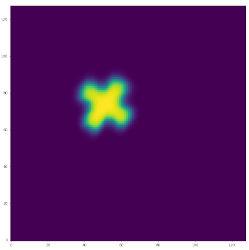

Рис. 49: t=102

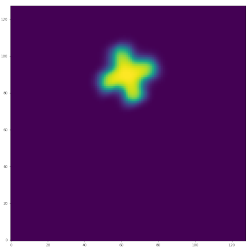

Рис. 50: t=205

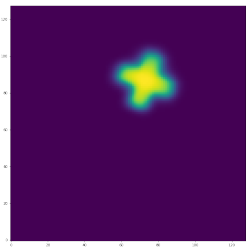

Рис. 51: t=307

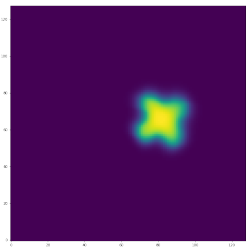

Рис. 52: t=410

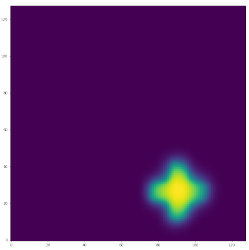

Рис. 53: t=512

Результаты расчетов для схемы THINC:

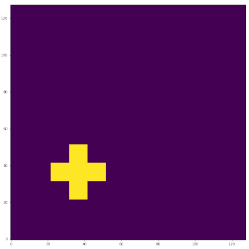

Рис. 54: t=0

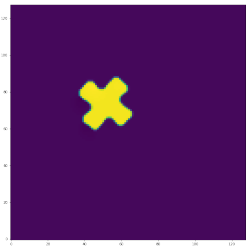

Рис. 55: t=102

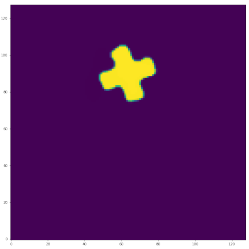

Рис. 56: t=205

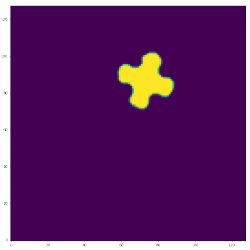

Рис. 57: t=307

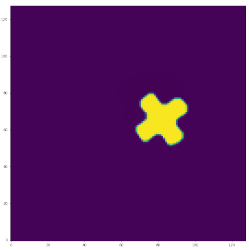

Рис. 58: t=410

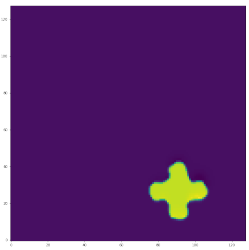

Рис. 59: t=512

Результаты расчетов для схемы JR:

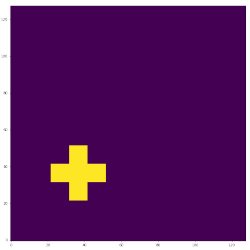

Рис. 60: t=0

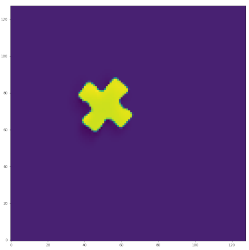

Рис. 61: t=102

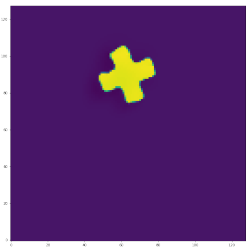

Рис. 62: t=205

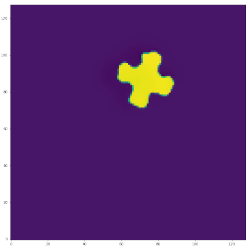

Рис. 63: t=307

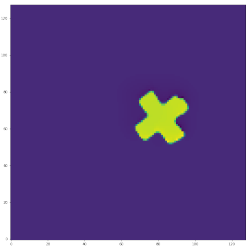

Рис. 64: t=410

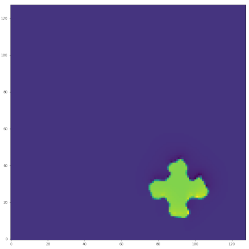

Рис. 65: t=512

Сравнение размазывания исходной формы твердого тела после поступательного и вращательного движения:

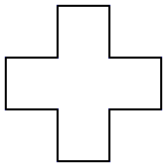

Рис. 66: Initial

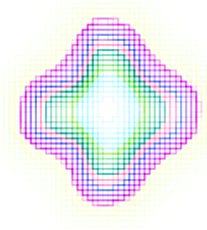

Рис. 67: MUSCL

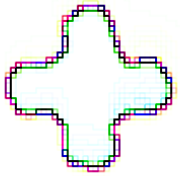

Рис. 68: THINC

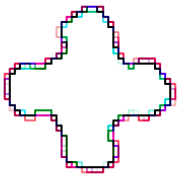

Рис. 69: JR

8 Заключение

Были исследованы численные методы решения уравнения переноса и реализован его программный алгоритм для одномерного случая.

Исследование схем для решения уравнение переноса в одномерном случае позволяет использовать данный подход в двумерном и трехмерном пространстве методом расщепления по координатным направлениям. Программа, написанная для одномерного случая, используется в дальнейшем как расчетный блок для многомерного случая.

Для представления цифровой геометрии были исследованы схемы с разным порядком точности, в том числе схема Годунова, MUSCL и THINC. Применение схемы THINC в отличие от других схем позволяет с высокой точностью описать поведение геометрии в цифровом виде, значительно уменьшить величину ошибки при расчетах.

Был разработан метод точного восполнения разрыва характеристической функции на подсеточном уровне (Jump Reconstruction, JR). Метод Jump Reconstruction позволяет в точности восстановить решение в одномерном случае и показывает хорошие результаты при многомерном обобщении.

Список литературы

[1] Годунов С.К. О единственности решения уравнений гидродинамики / С.К.Годунов // Математический сборник. - 1956. - Т.40, N 4. - С.467-478.

[2] Багриновский К.А. Разностные схемы для многомерных задач / К.А.Багриновский, С.К.Годунов // Доклады Академии наук СССР. - 1957. - Т.115, N 3. - С.431-433.

[3] Годунов С.К. Разностный метод численного расчета разрывных решений уравнений гидродинамики / С.К.Годунов // Математический сборник. - 1959. - Т.47, N 3. - С.271-306.

[4] Годунов С.К. О неединственном «размазывании» разрывов в решениях квазилинейных систем / С.К.Годунов // Доклады Академии наук СССР. - 1961. - Т.136, N 2. - С.272-273.

[5] Годунов С.К. Оценки невязок для приближенных решений простейших уравнений газовой динамики / С.К.Годунов // Журнал вычислительной математики и математической физики. - 1961. - Т.1, N 4. - С.622-637.

[6] Годунов С.К. О разностных схемах второго порядка точности для многомерных задач / С.К.Годунов, А.В.Забродин // Журнал вычислительной математики и математической физики. - 1962. - Т.2, N 4. - С.706-708.

[7] Годунов С.К. Разностные методы численного решения задач газовой динамики / С.К.Годунов, К.А.Семендяев // Журнал вычислительной математики и математической физики. - 1962. - Т.2, N 1. - С.3-14.

[8] Годунов С.К. Проблема обобщенного решения в теории квазилинейных уравнений и в газовой динамике / С.К.Годунов // Успехи математических наук. - 1962. - Т.17, вып.3. - С.147-158.

[9] Годунов С.К. Об использовании подвижных сеток в газодинамических расчетах / С.К.Годунов, Г.П.Прокопов // Журнал вычислительной математики и математической физики. - 1972. - Т.12, N 2. - С.429-440.

[10] Бахвалов П.А. Нестационарный метод геометрического корректора и его использование для оценки точности конечно-объемной схемы на

неструктурированных сетках // Препринты ИПМ им. М.В.Келдыша. - 2016. - № 122. - 28 с.

[11] Бочкарева Е.В., Храпов С.С. Численное моделирование динамики звуковых волн в активных средах с использованием схемы MUSCL // Вестник Волгоградского государственного университета. Серия 1: Математика. Физика. – 2015. – № 1 (18). - С. 13-22.

[12] К. В. Костюшин, В. А. Шувариков, Сравнение схем типа "MUSCL"для расчета течений идеального газа в соплах Лаваля, Вестн. Томск. гос. ун-та. Матем. и мех., 2018.

[13] Чжан Ч., Меньшов И.С. Сквозной метод расчета уравнений переноса многокомпонентной гетерогенной системы на фиксированных эйлеровых сетках // Математическое моделирование. – 2019. - №4 (31). – С. 111–130.

[14] Xiao F., Honma Y., Kono T. A simple algebraic interface capturing scheme using hyperbolic tangent function // International Journal for Numerical Methods in Fluids. 2005. Т. 48, № 9, с. 1023—1040.

[15] Einfeldt B., Munz C. D., Roe P. L. On Godunov-type methods near low densities // Journal of Computational Physics. 1991. Т. 92, № 2, с. 273—295.

[16] Ronit Kumar, Lidong Cheng, Yunong Xiong, Bin Xie, Rémi Abgrall, Feng Xiao. THINC scaling method that bridges VOF and level set schemes. // Journal of Computational Physics. 2021. 436: 110323.

9 Приложения

9.1 Пример программного кода нахождения координаты центра твердого тела в моменты времени t_i

```
function<Vector2D(vector<Vector2D>, double)> trapezoidQuadrature(){
    return [&](vector<Vector2D> f, double dx) -> Vector2D {
        return (f[0]+f[1])/2.*dx;
    };
}

void EESolver2DCenterStep(Vector2D& xc, const vector<Vector2D>& vc, int it,
const EESolver2DParams& params){
    if(it<=0) return;
    xc+=trapezoidQuadrature()({vc[it-1], vc[it]}, params.getDt());
}

void SolveEE2D(vector<Vector2D>& vertices, //vertices[0] is center
               const vector<Vector2D>& vc, const vector<double>& w,
               const EESolver2DParams& params, EESolver2DOutput& out){
    out.print(vertices, 0);
    for(int it=1; it<=params.getNTimeSteps(); it++){
        EESolver2DCenterStep(vertices[0], vc, it, params);
        //TODO solve for vertices here
        out.print(vertices, it);
    }
}
```

9.2 Пример программного кода нахождения значений правой функции в моменты времени t_i

```
Array<VectorXd, Dynamic, 1> integrateVector(Array<VectorXd, Dynamic, 1> f, double a
    Array<VectorXd, Dynamic, 1> integral(f.size());
    double h = (b - a) / (f.size() - 1.);
    VectorXd i0 = VectorXd::Zero(f(0).size());
    integral(0) = i0;
    for (int i = 1; i < f.size(); i++) {
        integral(i) = integral(i - 1);
        integral(i) += (f(i - 1) + f(i)) * h / 2;
    }
    return integral;
}

function<VectorXd(double)> getRightFunc(function<VectorXd(double)> v, VectorXd xi0
    return [=](double t)->VectorXd {
        function<VectorXd(double)> integrand = [=](double tau)->VectorXd {
```

```
            VectorXd xcTau = xc(tau);
            Vector3d xcTau3(xcTau(0), xcTau(1), dimN < 3 ? 0 : xcTau(2));
            Vector3d crossProd = xcTau3.cross(omega(tau));
            VectorXd crossProdX(dimN);
            for (int i = 0; i < dimN; i++) crossProdX(i) = crossProd(i);
            return v(tau) - crossProdX;
        };
        return xi0 + integrateVector(integrand, 0, t, pow(10, -5), pow(10, -5));
    };
}

function<VectorXd(double)> getRightFuncNoMargin(function<VectorXd(double)> v, funct
    return [=](double t)->VectorXd {
        function<VectorXd(double)> integrand = [=](double tau)->VectorXd {
            VectorXd xcTau = xc(tau);
            Vector3d xcTau3(xcTau(0), xcTau(1), dimN < 3 ? 0 : xcTau(2));
            Vector3d crossProd = xcTau3.cross(omega(tau));
            VectorXd crossProdX(dimN);
            for (int i = 0; i < dimN; i++) crossProdX(i) = crossProd(i);
            return v(tau) - crossProdX;
        };
        return integrateVector(integrand, 0, t, pow(10, -5), pow(10, -5));
    };
}
```

9.3 Пример программного кода нахождения значений x_k^i

```
Array<VectorXd, Dynamic, 1> getXi(function<VectorXd(double)> v,
VectorXd xi0,
function<Vector3d(double)> omega,
function<VectorXd(double)> xc, int stepsN, double h) {

    Array<VectorXd, Dynamic, 1> Xn(stepsN + 1);

    Array<VectorXd, Dynamic, 1> rightFuncCalc(stepsN + 1);
    function<VectorXd(double)> rightFunc = getRightFunc(v, xi0, omega, xc);
    for (int i = 0; i <= stepsN; i++)
        rightFuncCalc(i) = rightFunc(i * h);

    cout << "rightFuncCalc calculated" << endl;

    Xn(0) = rightFuncCalc(0);

    for (int k = 1; k <= stepsN; k++) {
        double sk = k * h;

        VectorXd innerSum(dimN);
```

```
        for (int i = 0; i < dimN; i++) innerSum(i) = 0;


        for (int j = 0; j < k; j++) {
            double sj = j * h;
            double Aj = (j == 0 ? 1. / 2. : 1.);

            VectorXd prodResult(dimN);
            Vector3d Xj(Xn(j)(0), Xn(j)(1), dimN < 3 ? 0 : Xn(j)(2));
            Vector3d crossProd = Xj.cross(omega(sj));
            for (int i = 0; i < dimN; i++) prodResult(i) = crossProd(i);

            innerSum += Aj * prodResult;
        }

        // Xk - h * innerSum(K-1) - h * Ak * [Xk ; OmegaK]  = rightFuncK
        // Xk - [Xk ; h * Ak *  OmegaK]  = rightFuncK + h * innerSum(K-1)
        // a - [a ; b] = c, a=?
        // ax - ay*bz = cx
        // ay + ax*bz = cy
        // (1 -bz)(ax)=(cx)
        // (bz  1)(ay)=(cy)
        // Ba=c, a = invB c

        Vector3d b = h * 0.5 * omega(sk);

        Vector2d rf2(rightFuncCalc(k)(0), rightFuncCalc(k)(1));
        Vector2d is2(innerSum(0), innerSum(1));
        Vector2d c = rf2 + h * is2;

        Matrix2d B; B <<
            1, -b(2),
            b(2), 1;

        Vector2d a = B.inverse() * c;

        VectorXd Xk(dimN);
        for (int i = 0; i < dimN; i++) Xk(i) = a(i);

        Xn(k) = Xk;
    }

    return Xn;

}
```

9.4 Пример программного кода решения уравнения переноса в одномерном случае со стационарным полем скоростей

```
double flow(const function<double(double)>& PsyL,
            const function<double(double)>& PsyR,
            double x, double u, double dt){
    return u>=0 ?
    PsyL(getXforInterpolation(x, u, dt/2)) :
    PsyR(getXforInterpolation(x, u, dt/2));
}

double fNext(F1D fi,
             U1D ui,
             C1D ci,
             double dt,
             const vector<function<double(double)>> &Psy){ // 3 functions: {Psy/i-
    double fiR = flow(Psy[1], Psy[2], ci.xR, ui.uR, dt);
    double fiL = flow(Psy[0], Psy[1], ci.xL, ui.uL, dt);
    return fi.fi - 1.0 / ci.dx * (fiR*ui.uR - fiL*ui.uL) * dt;
}

void TESolverStep(LineInterface &f,
                  LineInterface &u,
                  TESolver1DParams &p){
    vector<function<double(double)>> Psy(p.getCellCount()+2);
    for(int i=-1; i<p.getCellCount()+1; i++){
        int sgnUi = 0;
        if(i==-1) //TODO extend u outside area like f
            sgnUi = sgn(u[0]);
        else if(i==p.getCellCount())
            sgnUi = sgn(u[i]);
        else
            sgnUi = sgn((u[i]+u[i+1])/2.);
        F1D fi = getFi(f, i);
        C1D ci = getCi(p.getDx(), i);
        if(sgnUi<0){
            F1D invFi = inverse(fi);
            C1D invCi = inverse(ci);
            Psy[i+1] = fInverseX(p.getFlowInterpolationFunctionBuilder()(invFi, inv
        } else {
            Psy[i+1] = p.getFlowInterpolationFunctionBuilder()(fi, ci);
        }
    }


    double fiPrev = f[-1];
    double fiAfterLast = f[p.getCellCount()];
```

```
    for (int i = 0; i < p.getCellCount()-1; i++) { // calculating all cells except
        F1D fi = getFi(f, i);
        fi.fiPrev = fiPrev; // using saved fi as fiPrev in the next cell
        fiPrev = fi.fi;
        f.set(i, fNext(fi, getUi(u, i), getCi(p.getDx(), i), p.getDt(),
                       {Psy[i], Psy[i+1], Psy[i+2]}));
    }
    // fiNext for last cell is old f0, so we calc this separately
    int iLast = p.getCellCount()-1;
    F1D fiLast = {f[iLast], fiPrev, fiAfterLast};
    f.set(iLast, fNext(fiLast, getUi(u, iLast), getCi(p.getDx(), iLast), p.getDt()
}

void SolveTransportEquation1D(Area1D &f,
                              VectorField1D &u,
                              TESolver1DParams &p,
                              TESolver1DOutput &output) {
    output.print(f, 0, p.getDx());
    for (int n = 0; n < p.getNTimeSteps(); n++) {
        TESolverStep(f, u, p);
        output.print(f, n+1, p.getDx());
    }
}
```