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. Для данного представления рассматривается численное решение уравнения переноса с помощью метода объема жидкости (Volume of Fluid, VoF). Ключевым моментом реализации данного метода является аппроксимация скачка, которая является разрывом характеристической функции и наблюдается на границе двух сред. Цель данного исследования - программная реализация и сравнение различных схем геометрической реконструкции разрыва. Были исследованы классические методы Годунова (константа), MUSCL (линеная функция), относительно новый метод THINC (гиперболический тангенс), а также представлен метод Jump Reconstruction (JR, восполнение скачком).

Содержание

1 Введение **4**
1.1 Обзор литературы 5

2 Постановка задачи **5**
2.1 Цифровая геометрия 5
2.2 Точное решение: Direct Motion 6
2.3 Уравнение переноса 6
2.4 Jump Reconstruction 7
2.5 Ограничения и допущения 7

3 Точное решение **8**
3.1 Постановка задачи для точного решения 8
3.2 Теоретическая часть 9
3.3 Численное решение 9
3.4 Поле скоростей твердого тела. 12
3.5 Пример расчета 13
3.6 Исследование сходимости 14

4 Численное решение уравнения переноса в одномерном случае со стационарным полем скоростей **16**
4.1 Численное решение 16

5 Численное решение уравнения переноса в одномерном случае с нестационарным полем скоростей **18**
5.1 Численное решение 18
5.2 Метод характеристик 20
5.3 Аппроксимация интегралов 23

6 Исследование различных схем **24**
6.1 Метод Годунова(const) 24
6.2 Метод MUSCL (linear) 25
6.3 Метод THINC (tanh) 25
6.4 Метод JR (jump) 25
6.5 Исследование сходимости 25

7 Многомерное обобщение решения уравнения переноса с нестационарным полем скоростей **25**
7.1 Расщепление по направлениям 25
7.2 Тестовые расчеты 25

8 Заключение **25**

9 Список литературы **25**

10 Приложения **25**

10.1 Пример программного кода нахождения координаты центра твердого тела в моменты времени t_i 25

10.2 Пример программного кода нахождения значений правой функции в моменты времени t_i 26

10.3 Пример программного кода нахождения значений x_k^i 27

10.4 Пример программного кода нахождения значений правой функции в моменты времени t_i 28

1 Введение

В данной работе рассматривается проблема численного расчета движения твердого тела в сплошной среде. 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. Проблема геометрического представления заключается в сложности его совмещения с расчетом воздействия на тело сплошной среды. В данной работе представлен способ цифрового представления 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 метод объема жидкости (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 были разработаны различные схемы геометрической реконструкции, позволяющие при использовании их в расчетах метода объема жидкости получить достаточную точность. В данной работе проводится исследование классических способов аппроксимации скачка с помощью константного восполнения: метод Годунова, с помощью линейного восполнения: метод MUSCL. А также проведено исследование альтернативной схемы восполнения гиперболическим тангенсом: метод THINC. При использовании данной схемы была достигну-

та высокая точность расчетов. Схема THINC - (tangent of hyperbola for INterface capturing) схема отслеживание поверхности с помощью гиперболического тангенса. 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

1.1 Обзор литературы

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 (10.1) 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 (10.2).
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 (10.3).

3.4 Поле скоростей твердого тела.

Поле скоростей твердого тела связано по формуле Эйлера:

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

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 $\overrightarrow{v_c}(t)$ и $\overrightarrow{\omega}(t)$ 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:

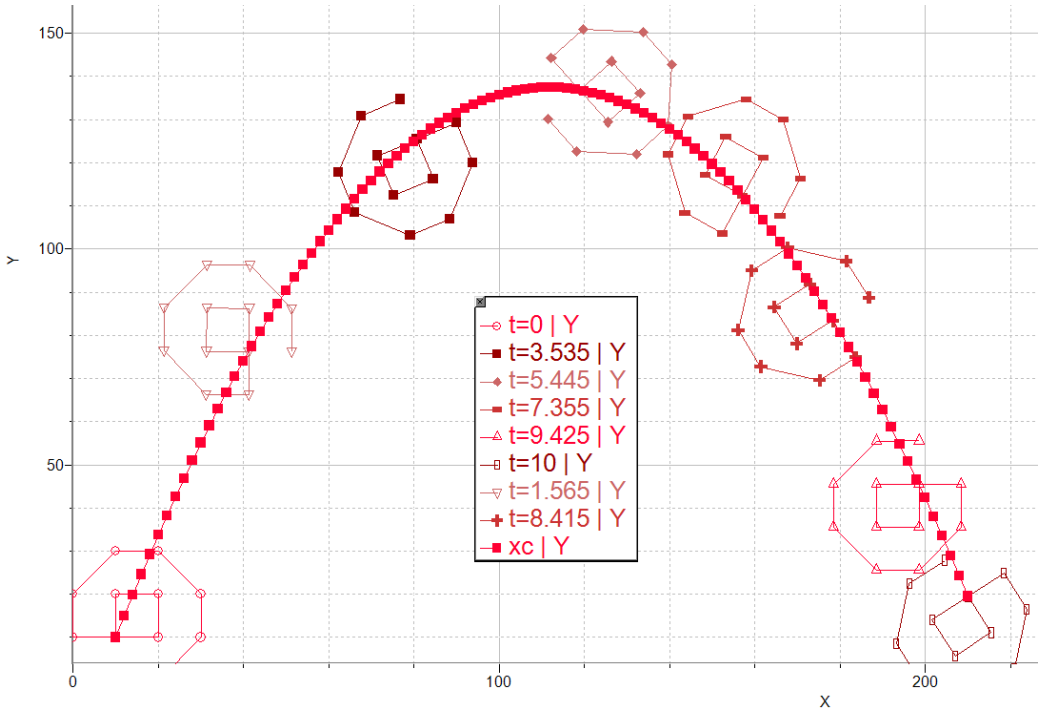


Рис. 1: Пример расчета движения точек твердого тела

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 $\overrightarrow{x_{real}}_k^i$ – 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:
.

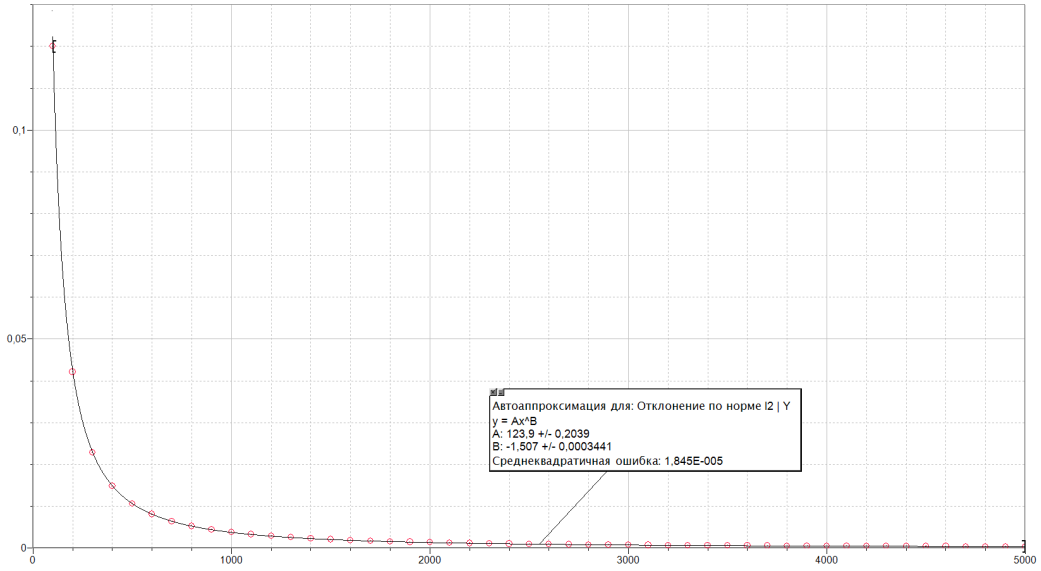


Рис. 2: Сходимость метода DM. Среднеквадратичное отклонение при измельчении шага по времени

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

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:

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

∇ – 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 $[0;T]$.

4.1 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 $\int_{t_n}^{t_{n+1}} Fdt$ используем квадратурную формулу прямоугольников:

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

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:

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

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 $[t_n; t_{n+1}]$:

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

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:

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

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

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

Интегралы

$$\Phi_{i+\frac{1}{2}} = \int_{t_n}^{t_{n+1}} u f_{i+\frac{1}{2}} d\tau \tag{18}$$

$$\Phi_{i-\frac{1}{2}} \int_{t_n}^{t_{n+1}} u f_{i-\frac{1}{2}} d\tau \tag{19}$$

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

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

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 методов MUSCL и THINC).

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:

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

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

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:

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

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

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:

6.2 Метод MUSCL (linear)

6.3 Метод THINC (tanh)

6.4 Метод JR (jump)

6.5 Исследование сходимости

7 Многомерное обобщение решения уравнения переноса с нестационарным полем скоростей

7.1 Расщепление по направлениям

7.2 Тестовые расчеты

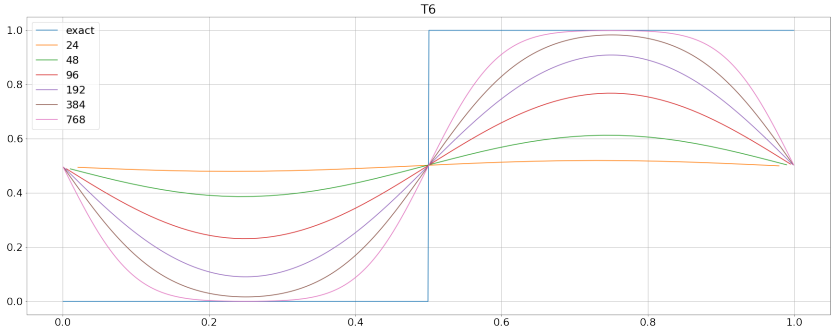
8 Заключение

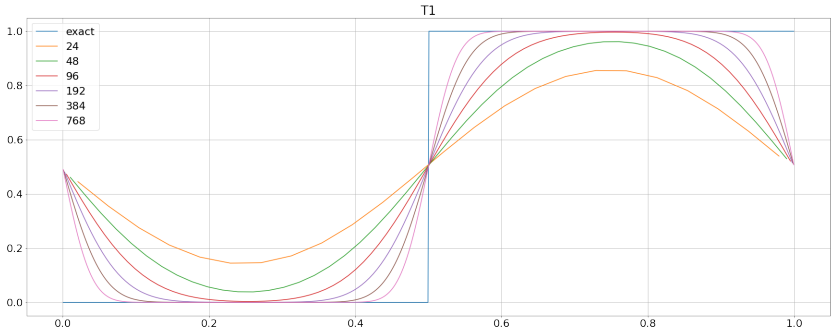
9 Список литературы

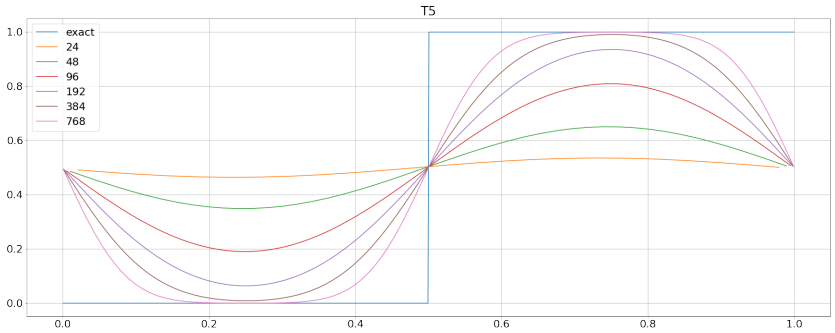
10 Приложения

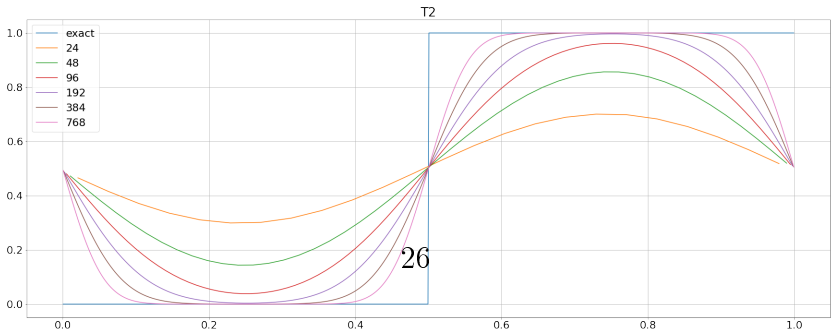
10.1 Пример программного кода нахождения координаты центра твердого тела в моменты времени t_i

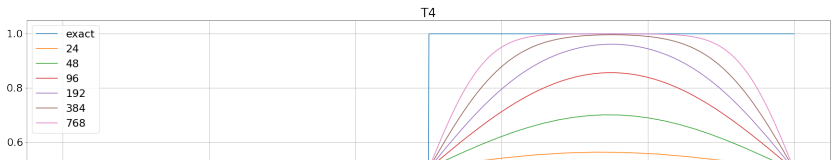
```
function<Vector2D(vector<Vector2D>, double)> trapezoidQuadrature(){
    return [&](vector<Vector2D> f, double dx) -> Vector2D {
        return (f[0]+f[1])/2.*dx;
    };
}

void EESolver2DCenterStep(Vector2D& xc, const vector<Vector2D>& vc, int it,
const EESolver2DParams& params){
    if(it<=0) return;
```


T6
exact
24
48
96
192
384
768
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


T1
exact
24
48
96
192
384
768
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


T5
exact
24
48
96
192
384
768
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


T2
exact
24
48
96
192
384
768
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


T4
exact
24
48
96
192
384
768
1.0
0.8
0.6

```
    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);
    }
}
```

10.2 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));
    };
}
```

10.3 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;

    }
```

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