УДК 519.6

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЧИСЛЕННОГО РЕШЕНИЯ ОПЕРАТОРНО-РАЗНОСТНЫХ СЕТОЧНЫХ ЗАДАЧ ДВУМЕРНОЙ ГАЗОВОЙ ДИНАМИКИ С ИСПОЛЬЗОВАНИЕМ СИСТЕМЫ КЛАССОВ C++

М. Н. Саблин[1]

На базе техники объектно-ориентированного программирования разработаны средства программной реализации операторных алгоритмов численного решения операторно-разностных уравнений, описывающих сеточные задачи газовой динамики. Операторный подход позволяет единообразно формулировать в виде систем операторных уравнений сеточные задачи различной размерности в разных системах координат, с разными типами краевых условий и с использованием разнообразных способов аппроксимации, а также алгоритмов решения этих задач. Техника объектно-ориентированного программирования позволяет реализовать прямую возможность работы с сеточными функциями — элементами конечномерных пространств, с операторами, действующими в этих пространствах, с операторными уравнениями и операторными алгоритмами решения этих уравнений.

Ключевые слова: разностная, схема, сетка, оператор, численное, газовая динамика, класс, расчет, программный, комплекс.

Введение. В соответствии с принципами операторного подхода теории разностных схем [1], решение начально-краевой задачи математической физики сводится к решению сеточной задачи, представляющей собой систему операторно-разностных уравнений в конечномерных линейных пространствах сеточных функций. Обоснование устойчивости, построение и исследование итерационных методов и формулировка алгоритмов решения сеточной задачи проводятся в операторном виде на основе хорошо развитой теории операторов в конечномерных линейных пространствах [2 – 4]. При этом для сеточных задач, описываемых такой системой операторно-разностных уравнений, теоретические результаты и алгоритмы являются универсальными в том смысле, что различные характеристики задач (размерность, топология сетки, способ аппроксимации, краевые условия, используемая система координат) содержатся лишь в определениях сеточных операторов и пространств сеточных функций.

Таким образом, при использовании операторного подхода сеточная задача и алгоритм ее решения формулируются на высоком иерархическом уровне в терминах универсальных операторных уравнений в конечномерных пространствах сеточных функций, а многочисленные детали, значимые с точки зрения приложений, содержатся в определениях сеточных операторов и множеств сеточных функций. При выполнении настоящей работы ставилась задача разработки программ, в которых напрямую реализуются иерархические и универсальные свойства операторного подхода.

В существующих программных комплексах решения сеточных задач операторный подход не реализуется в указанном выше смысле. Обычно программный комплекс представляет собой набор алгоритмических модулей для заранее выбранного конечного набора готовых вариантов задач и алгоритмов их решения. Пользователь выбирает подходящий для него алгоритмический модуль. При таком способе организации программы возможность модернизации алгоритмов затруднена тем, что изменение готового модуля требует значительных усилий и временны́х затрат на отладку даже от специалиста, хорошо знакомого с кодом. Реализованный программно-операторный подход предоставляет квалифицированному пользователю возможность быстрого изменения характеристик задачи и алгоритмов на различных иерархически изолированных уровнях программы. Это важно при численном решении серии сложных задач, отличающихся от стандартных, особенно когда требуется выбрать наиболее оптимальный способ решения путем оперативного внесения изменений в программу и последующего тестирования. Поэтому задача построения программного комплекса, который минимизирует усилия квалифицированного пользователя при изменении варианта задачи или алгоритма при разработке новых версий комплекса, является актуальной. Актуальность задачи повышается при рассмотрении неявных методов решения нестационарных задач, так как в этом случае алгоритмы решения сеточных задач на шаге по времени являются нетривиальными в отличие от случая использования явных методов.

[1] Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, Ленинские горы, 119992, Москва; e-mail: mark@garant.ru

Построение такого программного комплекса с использованием средств объектно-ориентированного и обобщенного программирования [5] и описано в данной работе. Этот комплекс реализует операторные алгоритмы численного решения сеточных задач газовой динамики. В качестве примеров использованы фрагменты реальной программы в варианте, адаптированном для решения сеточных задач, сформулированных в [6, 7].

В [6, 7] построены двумерные операторно-разностные схемы газовой динамики на нерегулярных треугольных сетках в случае аксиальной симметрии, обладающие свойством локальной аппроксимации вблизи оси симметрии. При применении операторного подхода теории разностных схем к решению задач механики сплошной среды [8 – 16] численному моделированию на нерегулярных непрямоугольных сетках уделяется повышенное внимание. Использование таких сеток целесообразно, если расчетная область имеет сложную форму или в задаче присутствуют сильные пространственные локальные неоднородности, требующие применения адаптивных сеток.

Комплекс программ построен с помощью специально разработанной системы классов, написанной на языке C++ с использованием возможностей стандартной библиотеки этого языка. Эта система классов описывает иерархические и структурные свойства операторного подхода теории разностных схем через пользовательские типы данных. Ее применение позволяет разделить программу на две части. В первой (неуниверсальной) части реализуются сеточные операторы, которые зависят от указанных выше особенностей задачи. Вторая (универсальная) часть программы не зависит от этих особенностей и предназначена для работы с алгоритмами решения операторных уравнений.

В универсальной части программы алгоритмы решения систем операторных сеточных уравнений представлены в обобщенной форме, не зависящей от размерности задачи, топологии сетки, способа аппроксимации, набора краевых условий и вида системы координат. В этой части программы обеспечена прямая возможность работы с операторами и сеточными функциями, что облегчает модификацию кода при изменении алгоритма решения.

Система классов спроектирована таким образом, что изменения, которые необходимо внести в неуниверсальную часть программы, сводятся к программированию сеточных аналогов дифференциальных операторов исходной задачи. При этом с помощью специальных вспомогательных классов сеточные операторы реализованы с учетом структуры определяющих формул, что позволяет свести любые изменения к локальным изменениям в коде ограниченного набора функций.

В результате структурно-иерархические возможности и универсальные свойства операторного подхода прямо реализуются на программном уровне. Тем самым обеспечивается возможность удобной и быстрой модификации программы пользователем.

Применение предлагаемой системы классов имеет смысл в многовариантных развиваемых комплексах программ, предназначенных для численного моделирования широкого класса задач.

1. Особенности операторного подхода, требующие отражения на уровне системы классов. Сеточная задача, сформулированная на основе операторного подхода теории разностных схем, описывается с помощью системы операторно-разностных уравнений, инвариантных относительно способа аппроксимации неизвестных и уравнений, размерности задачи, системы координат, вида сетки и краевых условий.

В данной работе в качестве примера рассматривается случай неявной по времени операторно-разностной схемы в лагранжевых переменных для изоинтропического движения идеального газа.

Задача, решаемая на шаге по времени, имеет вид

$$\begin{aligned} &F_\eta(\eta, \boldsymbol{v}) = \eta - \tau A_{21}\boldsymbol{v} + f_1 = 0, \\ &\boldsymbol{F}_v(\eta, \boldsymbol{v}) = 2\boldsymbol{v} + \tau A_{12}P(\eta) + A_\nu \boldsymbol{v} + \boldsymbol{f}_2 = 0, \\ &\boldsymbol{x} = \breve{\boldsymbol{x}} + \tau \boldsymbol{v}. \end{aligned} \tag{1}$$

Здесь τ — шаг сетки по времени; η, $\boldsymbol{v}$, $\boldsymbol{x}$ — искомые сеточные функции, являющиеся элементами конечномерных евклидовых пространств; буквой A с индексами обозначены линейные сеточные операторы; P — нелинейный оператор, индуцируемый заданной скалярной функцией поточечно; $\breve{\boldsymbol{x}}$, f_1, $\boldsymbol{f}_2$ — известные сеточные функции. По физическому смыслу, η — удельный объем газа; $\boldsymbol{v}$ — скорость газа; $P(\eta)$ — давление и $\boldsymbol{x}$ — координаты узлов сетки. Оператор $A_\nu = -A_{21}\nu A_{12} = A_\nu^* \geqslant 0$ обусловлен наличием объемной вязкости, а операторы A_{12}, $A_{21} = -A_{12}^*$ соответствуют дивергенции вектора и градиенту скаляра с учетом включенных в систему краевых условий. Сеточные операторы A с индексами могут зависеть от координат узлов сетки, сеточная функция ν — коэффициент объемной вязкости — вычисляется по известным величинам при решении задачи (1) на фиксированном шаге по времени.

Система операторных уравнений (1) — частный случай описанной в [6] узловой (все сеточные функции определены в узлах треугольной сетки) двумерной по пространству операторно-разностной схемы в случае аксиальной симметрии. В операторной форме (1) записываются также ячеечно-узловая (различные сеточные функции определены в узлах или в ячейках треугольной сетки) схема из [4, 17 – 20], двумерные по пространству схемы в декартовой системе координат, одномерные и трехмерные схемы.

Пример (1) — подвариант операторно-разностных схем из [6, 7], наиболее компактный по объему, однако достаточный для целей данной работы.

В соответствии со способом, описанным в [4, 6, 7], операторы A_{12} и A_{21} задаются с помощью операторов $[\bigtriangledown_x \cdot]$, $[\bigtriangledown_x]$ — сеточных аналогов дивергенции вектора и градиента скалярной функции, оператора $[\Phi_{\gamma_2} \cdot]$, обусловленного учетом краевых условий и аппроксимирующего операции умножения функции на нормаль к границе, и операторов проектирования сеточных функций на границу (или ее часть) сеточной области, обозначаемых далее символом δ (возможно, с индексами).

Система (1) решается модифицированным итерационным методом Ньютона, взятым из [4].

На каждой итерации (k — номер итерации) сначала находим $\overset{k+1}{\boldsymbol{v}}$, решая относительно поправки $\overset{k}{\boldsymbol{z}} = \overset{k+1}{\boldsymbol{v}} - \overset{k}{\boldsymbol{v}}$ задачу

$$B \overset{k+1}{\boldsymbol{z}} + \boldsymbol{F}_v(\overset{k}{\eta}, \overset{k}{\boldsymbol{v}}) = 0, \quad B \equiv 2I + A_\nu + \tau^2 A_{21} P_\eta A_{12}, \tag{2}$$

а затем находим удельный объем и координаты узлов сетки:

$$\overset{k+1}{\eta} = f_1 - \tau A_{12} \overset{k+1}{\boldsymbol{v}}, \quad \overset{k+1}{\boldsymbol{x}} = \breve{\boldsymbol{x}} + \tau \overset{k+1}{\boldsymbol{v}}. \tag{3}$$

Здесь B — линейный оператор, $B = B^* > 0$, P_η — линейный оператор, производная оператора P по η.

Первое требование, предъявляемое операторным подходом к иерархически структурированной программе, — возможность непосредственной реализации операторных формул и алгоритмов вида (1) – (3). На этом иерархическом уровне мы имеем дело фактически с несколькими относительно простыми уравнениями, в которых участвуют такие объекты, как сеточные функции и операторы.

Задание краевых условий проводится также на операторном уровне путем задания формул, определяющих операторы A_{12} и A_{21} сеточной задачи. Например, при использовании краевых условий прилипания, непротекания и условий свободной границы эти операторы выглядят следующим образом [6]:

$$\begin{gathered} A_{12} = (I - \delta_{\gamma_1})\big([\bigtriangledown_{x,m} \cdot] - [\Phi_{\gamma_2,m} \cdot]\big)(I - \delta_{\gamma_0}), \quad A_{21} = (I - \delta_{\gamma_0})[\bigtriangledown_{x,m}](I - \delta_{\gamma_1}), \\ [\bigtriangledown_{x,m} \cdot] = \frac{V_x}{m_x}[\bigtriangledown_x \cdot], \quad [\Phi_{\gamma_2,m} \cdot] = \frac{V_x}{m_x}[\Phi_{\gamma_2} \cdot], \quad [\bigtriangledown_{x,m}] = \frac{V_x}{m_x}[\bigtriangledown_x]. \end{gathered} \tag{4}$$

Здесь V_x, m_x — сеточные функции, имеющие смысл объема и массы узла соответственно; индексы γ_0, γ_1, γ_2 обозначают части границы сеточной области, на которых задаются краевые условия прилипания, свободной границы и непротекания соответственно. Операторный подход предполагает возможность прямой реализации в программах, определяющих операторы A_{12}, A_{21}, A_v и B, формул вида (4), содержащих элементарные операции с базовыми операторами. В рассматриваемом примере базовыми являются операторы

$$[\bigtriangledown_{x,m} \cdot], \quad [\Phi_{\gamma_2,m} \cdot], \quad [\bigtriangledown_{x,m}]. \tag{5}$$

При изменении топологии сетки, способа аппроксимации дифференциальных операторов или размерности задачи код изменяется только в той его части, которая непосредственно относится к реализации базовых операторов (5). Модули, относящиеся к этой части кода, должны быть легко и удобно модифицируемыми за счет максимального учета в системе классов структуры определяющих формул.

Рассмотрим особенности структуры формул задания базовых операторов на примере оператора $\bigtriangledown_x$ — сеточного аналога градиента скалярной функции.

В соответствии с принципами операторного подхода для построения разностной схемы необходимо задать в расчетной области сетку, определить на этой сетке конечномерные линейные пространства сеточных функций и действующие в этих пространствах сеточные операторы.

Двумерные операторные разностные схемы, построенные в [6, 7], используют треугольные сетки. Это означает, что сеточная расчетная область представляет собой объединение треугольных ячеек. Окрестность ячейки составляет множество узлов, являющихся ее вершинами. Под ячеечной окрестностью узла понимается множество ячеек, которым он принадлежит, а под узловой окрестностью узла понимается множество узлов (вершин треугольных ячеек), содержащих данный узел. Кроме того, для граничных узлов определяется граничная окрестность узла: множество граничных узлов, смежных с заданным. Узлы и ячейки в окрестностях нумеруются в порядке обхода узла против часовой стрелки.

На заданной сетке вводятся конечномерные линейные пространства сеточных функций, определенных в узлах или в ячейках сетки. Для узловой аппроксимации системы уравнений газовой динамики с первым порядком точности используется набор кусочно-линейных базисных функций [21], а для ячеечно-узловой аппроксимации — системы кусочно-линейных и кусочно-постоянных базисных функций [21]. Двумерные по пространству ячеечно-узловые операторно-разностные схемы [4, 15, 16] в случае аксиальной симметрии теряют локальную аппроксимацию вблизи оси симметрии, поэтому были разработаны чисто узловые схемы [6, 7].

Пусть $\{r, \varphi, z\}$ — ортогональная система координат в $\mathbb{R}^3$, G — двумерная ограниченная область с кусочно-гладкой границей, лежащая в плоскости $\varphi = \text{const}$. В рассматриваемом двумерном случае зависимость функций от переменной φ отсутствует. В области G введена треугольная сетка ω и системы $\{\varphi_\omega\}$, $\{\psi_\omega\}$ финитных базисных функций метода конечных элементов на этой сетке. Любую сеточную формулу, с помощью которой аппроксимируется непрерывный аналог инвариантного дифференциального оператора A первого порядка, полученную разностным, интегро-интерполяционным, проекционным или проекционно-сеточным методами, можно вывести на основе формулы вида

$$\frac{1}{V_\omega} \int\limits_G A(f) \varphi_\omega \sqrt{g}\, dG. \tag{6}$$

Здесь g — определитель метрического тензора, $V_\omega = \int\limits_G \varphi_\omega \sqrt{g}\, dG$ — сеточный объем, f — функция, к которой применяется оператор. Если в качестве базисных берутся системы кусочно-линейных функций (как в случае узловой аппроксимации), то сеточный оператор вычисляется путем подстановки в (6) кусочно-линейной функции f_{int}, получаемой интерполяцией функции f базисными функциями, и последующего интегрирования. Если используется система кусочно-линейных базисных функций $\{\varphi_\omega\}$ и система кусочно-постоянных базисных функций $\{\psi_\omega\}$ (как в случае ячеечно-узловой аппроксимации), то перед подстановкой в формулу (6) кусочно-постоянного интерполянта f_{int} выражение (1) преобразуется с использованием формулы Гаусса–Остроградского к виду, в котором отсутствует дифференцирование функции f [4].

В [6, 7] приведены формулы для сеточных аналогов инвариантных дифференциальных операторов первого порядка, полученных для случая аксиально симметричной двумерной задачи и узловой аппроксимации на треугольных сетках. Для градиента скалярной сеточной функции ρ в узле x аппроксимационная формула имеет вид

$$\bigtriangledown_x \rho = \frac{1}{V_x} \sum_{\Delta_i \in \mathrm{O}_x} \left(\left(\sum_{k=1}^{3} -\boldsymbol{N}_{k+1,k-1} \rho_k \right)_l \left(\frac{1}{2 s_l} \int\limits_{\Delta_l} \varphi_x r\, dr\, dz \right) \right). \tag{7}$$

Здесь V_x — сеточный объем узла x, O_x — ячеечная окрестность узла x, Δ_l — ячейка из окрестности O_x, $\sum\limits_{k=1}^{3}$ — сумма по узлам окрестности ячейки, $\boldsymbol{N}_{k+1,k-1}$ — внешняя нормаль к стороне ячейки Δ_l, s_l — площадь ячейки.

Хотя вид аппроксимационных формул для различных инвариантных дифференциальных операторов первого порядка изменяется в зависимости от способа аппроксимации, типа оператора и вида аргумента оператора, эти формулы имеют некоторые общие структурные и алгоритмические особенности. В частности, как и в (7), для них характерно выполнение одних и тех же операций со значениями аргумента во всех элементах окрестности фиксированного элемента сетки. В описываемой в данной работе системе классов такие структурные особенности определяющих формул реализованы через специальные операции типа “во всех узлах окрестности ячейки умножить значение одной сеточной функции на другую” или “просуммировать значения сеточной функции по всем элементам ячеечной окрестности узла”.

В целом, разработанная система классов упрощает процесс модификации программы, если этого требует решение конкретной сеточной задачи, за счет того, что

— сеточные функции определяются с помощью единого шаблона, описывающего в обобщенном виде любую сеточную функцию;

— выражения, в которых участвуют сеточные функции, записываются в операторном виде, аналогичном (4);

— в этих выражениях реализована встроенная проверка типов значений аргументов;

— часть программы, непосредственно не относящаяся к конкретным формулам, определяющим базовые операторы вида (7), и связанная с использованием операторов типа (4), учитывающих краевые условия и структуру алгоритма решения задачи на шаге по времени, написана в виде обобщенных алгоритмов и не зависит от деталей реализации базовых сеточных операторов;

— программирование конкретных формул типа (7) для базовых сеточных операторов (5) осуществляется с учетом структуры определяющих формул.

2. Определение классов для сетки и сеточных функций. Для работы с сеточными функциями предназначен специальный шаблонный класс-контейнер Function <Type_Function, Type_Grid>, позволяющий сгенерировать класс в зависимости от значений своих аргументов. По определению (см. [5]), в контейнерах хранятся данные и определены базовые операции с ними. При объявлении класса для конкретной сеточной функции задается тип ее элементов через параметр шаблонного класса Type_Function $\in \{\text{Scalar}, \text{Vector}, \text{Tensor}\}$ (т.е. задается скаляр, вектор или тензор соответственно); через параметр шаблонного класса Type_Grid $\in \{\text{Cell}, \text{Knot}, \text{Cell\_Templ}, \text{Knot\_Templ}\}$ указывается, в каких элементах сетки определена сеточная функция — в ячейках, узлах, на сеточной окрестности ячейки или сеточной окрестности узла соответственно. Сеточные функции, определенные на сеточных окрестностях, являются вспомогательными; они используются при программировании формул для вычисления значений сеточных операторов в фиксированном элементе сетки по формулам вида (7).

Для хранения значений сеточной функции используется объект, определяемый в контейнере шаблонным классом template <class T> class valarray стандартной библиотеки языка C++. Класс valarray предназначен для хранения и управления последовательностью элементов типа T переменной длины и оптимизирован для повышения эффективности вычислений.

В качестве элементов типа T могут выступать действительные числа (для скалярной функции), элементы типа vector_component (для векторной функции) и tensor_component (для тензора второго ранга). Классы vector_component и tensor_component определены через структуру, в которой хранятся три действительных числа для вектора и девять действительных чисел для тензора. В этой структуре и системе классов определены операции, позволяющие использовать эти элементы в арифметических выражениях. Такими операциями являются присваивание, сложение, вычитание элементов одного типа, умножение элемента на число, деление элемента на число, отличное от нуля, а также операции внутреннего умножения вектора на тензор, внутреннего умножения тензора на тензор, свертка тензора, внешнее произведение двух векторов и модуль элемента. Кроме того, в соответствии со стандартом программирования на языке C++ в структуре определены конструкторы по умолчанию и инициирующие конструкторы, а также деструктор. Для элемента-тензора дополнительно определены функции-члены, позволяющие получить нужную строку или столбец матрицы в виде элемента векторного типа.

Классы, в которых определяется элемент сеточного вектора или тензора и операции над этим элементом, находятся в специальных заголовочных файлах. Это позволяет более эффективно использовать память, необходимую для программы, учитывая специфику конкретного расчета. Например, для двумерного расчета иногда достаточно хранить только два числа для вектора и четыре — для тензора, а для одномерной — вообще по одному числу. При изменении размерности решаемой задачи все изменения кода, касающиеся элементов сеточных функций, сводятся к замене соответствующих заголовочных файлов на нужные.

Для работы с сеточными функциями нецелесообразно напрямую использовать valarray и классы, производные от него, так как в этом случае придется отказаться от неявной проверки в арифметических выражениях типов сеточных функций относительно типа их области определения. На практике встречаются ситуации, когда функции с разным типом области определения хранятся в массивах одинакового размера, а это единственное условие, которое накладывается на возможность выполнения операций с аргументами типа valarray над данными одного типа. Поэтому в выражениях, использующих эти операции, возможны ошибки, когда, например, к ячеечной функции прибавляется узловая или скалярная функция складывается с векторной. Необходимость обеспечения неявной проверки типов в программе обусловила введение второго параметра (Type_Grid) в шаблон Function, что позволило обнаруживать такие ошибки на этапе компиляции.

В полном соответствии с концепцией обобщенных алгоритмов [5], которые должны формулироваться в виде, не зависящем от деталей представления, для задания операций над объектами, определенными с помощью Function, в программе определены итераторы, т.е. классы для пользовательских типов данных, обеспечивающих последовательное движение и доступ к элементам контейнера без необходимости всякий раз изменять программу при учете специфики конкретной задачи. Эти итераторы предназначены для единообразного определения циклов по узлам сетки, по ячейкам сетки, по граничным узлам, по ячеечной

окрестности узла, по узловой окрестности узла и для получения значения сеточной функции в отдельном элементе ее области определения.

В существующей программе определены и используются пять типов итераторов: ячеечный по всей сетке (iterator_cell), узловой по всей сетке (iterator_knot), узловой по граничным узлам (iterator_border), по окрестности ячейки (iterator_cell_template) и по окрестности узла (iterator_knot_template). Для всех итераторов определены следующие функции-члены:

— size( ) (общее число элементов в области определения);

— begin( ) (проверка установки на начало — первый элемент области определения);

— end( ) (проверка на достижение конца — последнего элемента области определения);

— reset( ) (установка текущего индекса на начало — первый элемент области определения);

— current( ) (номер текущего элемента);

— постфиксный и префиксный операторы инкремента ++ (переход к следующему элементу области определения);

— копирующий конструктор;

— деструктор;

— current_templ( ) — получение массива элементов типа итератор для текущего элемента итератора в следующих случаях: для ячейки (iterator_cell) или ячейки из окрестности узла (iterator_knot_template) — получение массива узлов (типа iterator_knot), которые относятся к вершинам текущей ячейки; в случае узла (iterator_knot) или узла из окрестности ячейки (iterator_cell_template) — получение массива ячеек (типа iterator_cell), которые принадлежат ячеечной окрестности текущего узла; в случае граничного узла (iterator_border) — получение массива узлов (iterator_knot), принадлежащих граничной окрестности этого узла.

Кроме того, для итераторов определены следующие функции-члены:

— для итератора iterator_knot по узлам всей сетки: internal( ) (для проверки, является ли узел внутренним или граничным);

— для итератора iterator_border по граничным узлам: sort( ) (для определения сорта границы для того, чтобы отличать разные участки друг от друга при постановке на них различных краевых условий) и неявное приведение к типу iterator_knot (преобразование в узловой итератор);

— для итераторов iterator_cell, iterator_knot и iterator_border: конструктор по умолчанию.

Для итераторов, предназначенных для работы с сеточными окрестностями, определены:

— конструктор с аргументом, который является значением итератора соответствующего типа (значение типа iterator_cell является аргументом конструктора для iterator_cell_template, а значение типа iterator_knot — для iterator_knot_template);

— source_element( ), где элемент, его породивший, имеет тип iterator_cell для iterator_cell_template или тип iterator_knot для iterator_knot_template;

— current_element( ) — значение типа итератор, соответствующее текущему элементу окрестности; для iterator_cell_template — iterator_knot и для iterator_knot_template — iterator_cell).

В программе используется вспомогательный класс Grid_Data_And_Structure, в котором содержится информация о сетке: список ячеек, список узлов, список граничных узлов, список элементов окрестностей ячеек, список элементов окрестностей узлов. Информация о структуре сетки используется при определении итераторов. В целях оптимизации использования памяти массивы и переменные, описывающие структуру сетки, хранятся как static, т.е. во время работы программы они создаются в единственном экземпляре независимо от числа созданных экземпляров класса.

Данные в классе Grid_Data_And_Structure хранятся в виде, который позволяет обмениваться данными с внешними программами, например, с программным комплексом для перестройки сетки [18] или с какой-либо программой визуализации расчетов.

Определенные выше операции с итераторами достаточны для программирования сеточных операторов в терминах обобщенных алгоритмов, однако для повышения удобства работы список этих операций расширен за счет часто встречающихся операций, характерных только для сеток определенной топологии. Например, в одномерных и двумерных расчетах при определении сеточных операторов в узлах сетки часто используются линейные комбинации значений сеточной функции в соседних узлах, поэтому в соответствующем итераторе с помощью специальных функций-членов реализовано получение ссылок на соседние узлы. Так, в конкретной программе для итераторов iterator_cell_template и iterator_knot_template определены функции current_knot0( ), current_knot1( ), current_knot2( ) (значение типа итератор iterator_knot; текущий, предыдущий и последующий узлы окрестности соответственно).

С использованием вышеперечисленных итераторов определяются следующие унарные и бинарные

операции с объектами класса Function <Type_Function,Type_Grid>, представляющими сеточные функции: присваивание; взятие обратного знака; сложение и вычитание функций с элементами одного типа и с одинаковой областью определения; почленное умножение функции на число, вектор и тензор; почленное деление функции на число, отличное от нуля; поэлементное умножение функции на скаляр; почленное прибавление к функции константы типа элемента; почленное умножение на скалярную, векторную и тензорную сеточные функции; покомпонентное умножение на число; внешнее почленное произведение двух векторных сеточных функций; свертка для векторов и тензоров. Кроме того, определены следующие функции-члены класса Function:

— конструктор: по умолчанию копирующий и инициирующий через valarray;

— деструктор;

— size( ): определение общего числа элементов;

— operator[ ]: аргумент в квадратных скобках; если функция определена в узлах, а аргумент имеет тип iterator_knot, то возвращается значение функции в соответствующем узле; если же аргумент имеет тип iterator_cell, то возвращается массив из значений функции в вершинах текущей ячейки; аналогично и для ячеечной функции с аргументом типа iterator_cell возвращается значение функции в ячейке, а с аргументом iterator_knot — массив со значениями функции в вершинах ячейки;

— фортранный стиль вызова элемента сеточной функции (аргумент не в квадратных, а в круглых скобках);

— max_abs( ): поточечный максимум модуля сеточной функции;

— sum( ): сумма всех элементов сеточной функции.

Благодаря введенным классам, например, вычисление в программе удельного объема газа в операторной формуле (3) будет выглядеть следующим образом:

$$\text{nuk} = \text{f1 - tau * A12(vk)}.$$

Здесь величина nuk, соответствующая сеточному скаляру $\overset{k+1}{\eta}$, определена как Function <Scalar, Knot>, а величина vk, соответствующая сеточному вектору $\overset{k+1}{\boldsymbol{v}}$, определена как Function <Vector, Knot>. Величина tau соответствует шагу по времени τ, A12 — функция-член для применения оператора A_{12}. Если определить f1 не как скалярную функцию, а как, например, векторную или скалярную, но определенную в ячейках, ошибка будет выявлена уже на этапе компиляции.

Оператор operator[ ] для получения значения сеточной функции в конкретном элементе сетки реализован таким образом: если переменная i (типа iterator_cell) — текущая ячейка, переменная j (типа iterator_knot) — текущий узел, а nuk — узловая скалярная функция из предыдущего примера, то значением nuk[j] будет значение $\overset{k+1}{\eta}_j$, а nuk[i] — значения $\overset{k+1}{\eta}$ во всех вершинах ячейки i, хранящиеся в виде функции типа Function <Scalar, Cell_Templ>. При этом определены также выражения вида nuk[i] * vk[i], что означает “умножить значения $\overset{k+1}{\eta}$ на значения $\overset{k+1}{\boldsymbol{v}}$ во всех вершинах ячейки i”. Такие выражения удобно использовать при программной реализации сеточных аналогов инвариантных дифференциальных операторов первого порядка. Эти сеточные операторы определяются в классе Grid с учетом особенностей конкретной задачи: размерности сетки и того, к каким элементам сетки, ячейкам или узлам относятся газодинамические величины.

3. Описание класса Grid. Класс Grid предназначен для того, чтобы с учетом топологических особенностей сетки и способа аппроксимации газодинамических величин задать необходимые для определения операторов операторно-разностной схемы сеточные аналоги инвариантных дифференциальных операторов первого порядка (базовые сеточные операторы). В существующей реализации класс содержит сеточные аналоги дифференциальных операторов

$$\bigtriangledown a, \quad \bigtriangledown \boldsymbol{b}, \quad \bigtriangledown \cdot \boldsymbol{b}, \quad \bigtriangledown \cdot (a\boldsymbol{b}), \quad \bigtriangledown \cdot (a\boldsymbol{bd}), \quad \bigtriangledown \cdot (a\boldsymbol{b}c), \quad \boldsymbol{b} \cdot \bigtriangledown a, \quad a \bigtriangledown \cdot \boldsymbol{b}, \quad a\boldsymbol{b} \bigtriangledown c, \quad a\boldsymbol{b} \cdot \bigtriangledown \boldsymbol{c}, \quad c \bigtriangledown \cdot (a\boldsymbol{b}),$$

где $\bigtriangledown$ — инвариантный дифференциальный оператор “набла”; a, c — скалярные функции; $\boldsymbol{b}$, $\boldsymbol{d}$ — векторные. Кроме того, в классе определен граничный оператор $[\Phi_{\gamma_2} \cdot]$. В отличие от рассматриваемого в работе примера, в комплексе программ присутствуют все базовые операторы, необходимые для работы с сеточной задачей газовой динамики в произвольной подвижной системе координат.

Каждому оператору соответствует две функции-члена: одна открытая, которая возвращает результат применения оператора на всей сетке, и вторая закрытая, которая возвращает результат применения оператора в конкретном элементе сетки. Функции имеют возвращаемое значение и аргументы типа Function <Type_Function, Type_Grid>, при этом тип Type_Grid зависит от того, каким способом аппроксимируется задача. В конкретном варианте программы используется узловая аппроксимация, поэтому

Type_Grid везде равен Knot. В случае ячеечно-узловой аппроксимации используется Type_Grid = Cell.

Вычисление результата применения оператора во всех узлах сетки в открытых функциях-членах реализовано в виде обобщенного алгоритма с помощью цикла по всей сетке, где вызывается соответствующая закрытая функция-член, в которой содержатся все особенности реализации этого оператора.

Кроме вышеназванных функций-членов, класс Grid содержит вспомогательные функции, с помощью которых вычисляются операторы, такие как сеточный объем, площадь ячейки и нормали к граням ячейки.

Информация о сетке хранится в классе Grid в виде ссылок на два экземпляра описанного выше класса Grid_Data_And_Structure. Два экземпляра класса Grid_Data_And_Structure необходимы для манипулирования связанными сетками (на верхнем или нижнем временны́х слоях, двух соседних итерациях и т.п.); в функциях-членах за это отвечают аргументы weight (вес со значением из отрезка [0, 1], с которым берутся координаты для вычисления нормалей и других величин) и layer $= 1, 2$ (временной слой, на котором берутся координаты для вычисления сеточного объема).

В качестве примера приведем фрагмент кода определения класса Grid, в котором по формуле (7) определен сеточный аналог градиента скалярной функции в узлах сетки.

Для программирования этого оператора используются вспомогательные функции, вычисляющие сеточный объем узла V_x (Vi(j, layer)), нормали к граням ячейки $[-\boldsymbol{N}_{k+1,k-1}]_i$ (Ni(i, weight)) и r-компоненту радиус-векторов ячеек из ячеечной окрестности узла $\left[\dfrac{1}{2s_l}\displaystyle\int\limits_{\Delta_l}\varphi_x r\,dr\,dz\right]_x$ (ri_cells(j, weight)). Вспомогательные функции могут возвращать одно значение в конкретном элементе сетки (например, Vi(i, layer) вычисляет сеточный объем узла i) или несколько значений сразу для всех элементов окрестности элемента сетки (например, Ni(i, weight) вычисляет все нормали к граням ячейки i, а ri_cells(i, weight) — r-компоненты всех ячеек ячеечной окрестности узла i). Если необходимо рассчитывать значения сеточных величин на окрестности, как отмечалось выше, возвращаемое значение функции имеет тип сеточной функции Function, определенной на окрестности соответствующего элемента, а операции с такими значениями определены таким образом, что при работе с ними через Function можно избежать необходимости явно программировать циклы по элементам окрестности. Например, формула temp $= \left[\displaystyle\sum_{k=1}^{3} -\boldsymbol{N}_{k+1,k-1}\rho_k\right]_i$, где ρ — сеточная узловая функция, записывается следующим образом:

$$\text{temp} = (\text{Ni(i, weight)*ro[i]}).\text{sum( )}.$$

В правой части этого равенства стоит выражение “во всех узлах окрестности ячейки i умножить соответствующую им нормаль на значение ro в этом узле, а затем просуммировать полученные значения”.

Данный прием используется и в приведенном ниже фрагменте кода для вычисления градиента скалярной функции в узле j по формуле (7) (закрытая функция-член).

```
vector_component Grid::grad(const Function<Scalar,Knot>&a,iterator_knot j,double weight,size_t layer)const
{
    // Вспомогательная векторная функция res определяется на ячейках окрестности узла j
    Function<Vector> res(j.current_templ( ).size( ));
    for(iterator_knot_template i(j); i.end( ); i++) // Цикл по всем ячейкам окрестности узла j
        // res[i] = \sum_{k=1}^{3} -N_{k+1,k-1} a_k :
        res[i] = (Ni(i.current_element( ), weight)*a[i.current_element( )]).sum( );
    // Для каждой ячейки i окрестности узла j res[i] умножаем на :
    res *= ri_cells(j,weight);
    // Суммируем все элементы res и делим сумму на сеточный объем узла j:
    return res.sum( )/Vi(j,layer);
}
```

Остальные сеточные операторы определяются аналогичным способом. Базовые сеточные операторы из класса Grid используются для определения сеточных операторов типа A_{12}, A_{21}, A_v, B (по формулам вида (4)), учитывающих краевые условия и участвующих в алгоритме непосредственно. Эти операторы определены в виде обобщенных алгоритмов в функциях-членах описываемого ниже класса Step, предназначенного для реализации алгоритма решения сеточной задачи на шаге по времени в универсальной части программы, не зависящей от конкретных деталей реализации.

4. Определение класса Step для проведения расчета. Для организации расчета временно́го шага сеточной задачи предназначен класс Step. В нем содержатся все необходимые для этого данные, а через обобщенные алгоритмы определены операторы, участвующие в расчете. Детали конкретной реализации скрыты в специально определенных для этого классах, описанных выше.

Таким образом, при изменении размерности задачи, системы координат или топологии сетки нет необходимости изменять часть программы, относящуюся к расчету временно́го шага.

Класс Step содержит:

— решение с предыдущего временно́го слоя;

— решение с текущего временно́го слоя;

— ссылку на экземпляр класса Grid, в котором определяются базовые операторы — сеточные аналоги дифференциальных операторов первого порядка;

— функции-члены, реализующие сеточные операторы типа A_{12}, A_{21}, A_ν, B, которые учитывают краевые условия и участвуют в алгоритме непосредственно;

— функции-члены, реализующие уравнения состояния и другие характеристики среды: формулы для давления и для внутренней энергии, их частные производные по аргументам, скорость звука, коэффициент вязкости, другие величины, используемые в алгоритме;

— функции-члены, реализующие индикаторы типа границы (граничных условий);

— функции-члены, реализующие граничные значения сеточных функций и операторы проектирования на конкретную границу;

— SMAIN( ): программу с циклом для расчета временно́го шага;

— ссылку на экземпляр специального вспомогательного класса it_parameters, содержащего параметры итерационного процесса;

— конструктор класса.

Если опустить стандартные для всех расчетных программ детали, относящиеся к считыванию и сохранению информации, в конкретной реализации программы расчет организован следующим образом. В цикле по шагам временно́й сетки сначала начальные значения для расчета временно́го шага заносятся в предварительно определенный экземпляр класса Step, затем вызывается функция SMAIN, в которой временной шаг рассчитывается, а затем после сохранения полученного решения осуществляется переход к расчету следующего временно́го шага.

Применение техники обобщенного программирования позволило использовать в этой части программы форму записи операторных уравнений, близкую к использующейся в теоретических работах, что очень удобно при последующей поддержке программного комплекса. Например, итерационный процесс (2), (3) для решения (1) в SMAIN будет выглядеть следующим образом.

```
// ...
// Определяем вспомогательные величины
bool crk = false;
// Вычисляем неизменные на временно́м шаге части невязок
// ...
// Задаем начальное приближение — решение с предыдущего временно́го шага
eta = eta0;
v = v0;
z = 0.0;
for(size_t k=0; !(*params).endk(crk, k); k++)
  {
  // Вычисляем невязки для проверки, сошлись ли итерации
  Feta = eta — tau*A12(v) + f1;
  Fv = 2.0*v + tau*A21(Pressure(eta)) + Av(v) - f2;
  // Проверяем, сошлись ли итерации, и, если сошлись, прерываем итерации
  if((*params).endk(Feta.max_abs( ), Fv.max_abs( ), 0, k+1))
  {
      crk = true;
      break;
  }
  // Находим скорость, решая систему (2)
  z = inverse(z, Fv, end_it((*params).value_epsNEWT1( ), (*params).value_maxNEWT1( )), B, inverse_D);
```

```
  v = z + v;
  // Находим удельный объем по формуле из (3)
  eta = f1 - tau*A12(v);
  // Находим новые координаты узлов сетки по формуле из (3)
  recalculate_x(v, tau);
  }
// ...
```

В этом фрагменте в подпрограмме inverse решается система линейных алгебраических уравнений (2).

SMAIN имеет аналогичный вид и в более сложных случаях. Однако и из этого варианта видно, что вид кода достаточно прост и нагляден, чтобы свести к минимуму время, необходимое для его модификации при изменении итерационного процесса.

Достаточно просты и подпрограммы, в которых реализуются сеточные операторы типа A_{12}, A_{21}, A_ν и B. Например, функции-члены для последних двух выглядят следующим образом.

```
Function<Vector, Knot> Step::Av(const Function<Vector,Knot> & a) const
{
    return -A21(nu*A12(a));
}


Function<Vector,Knot> Step::B(const Function<Vector,Knot>& a) const
{
    return 2.0*a + Av(a) + tau*tau*A21(Peta*A12(a));
}
```

Здесь величина tau соответствует шагу по времени τ, величина Peta соответствует сеточному скаляру, реализующему применение линейного оператора P_η, величина nu соответствует коэффициенту вязкости ν. Видно, что функции не зависят от деталей реализации и что исходные формулы из (1), (2) отображаются в коде фактически в явном виде.

Заключение. Положенные в основу описанной системы классов структурно-иерархические свойства операторного подхода теории разностных схем, в сочетании с методами объектно-ориентированного и обобщенного программирования, позволили представить алгоритм решения сеточной задачи в обобщенном виде, не зависящем от деталей задания сетки и базовых сеточных операторов. В результате пользователь может разделять работу по программированию на части, учитывающие особенности конкретной постановки задачи с точки зрения сеточной аппроксимации конкретной физической задачи, и на части, отвечающие за решение системы операторно-разностных уравнений. При этом стандартная форма записи операторных уравнений, специально определенные операции над сеточными функциями и проверка соответствия типов на этапе компиляции значительно сокращают время отладки как при внесении изменений в алгоритм итерационного процесса или постановку физической задачи, так и при изменении размерности задачи или топологии сетки.

Отметим, что в случае необходимости данный комплекс может быть применен и для решения трехмерных задач. Потребуется только запрограммировать формулы применения сеточных аналогов дифференциальных операторов в узлах сетки и необходимые для этого вспомогательные формулы (такие как формулы для нормалей к граням ячеек) в классе Grid.

Адаптация комплекса, отлаженного для одного варианта способа аппроксимации, размерности и системы координат, к другому варианту сводится к изменениям только в определениях класса Grid. Модификация задачи и алгоритма сводится лишь к изменениям класса Step. Рассмотренные особенности предложенной в работе системы классов, наряду с наличием в ней возможностей для удобного внесения изменений, значительно облегчают процесс модификации задачи и алгоритма в рамках операторно-ориентированного комплекса программ.

Автор выражает глубокую благодарность Н. В. Арделяну за постоянное внимание к работе.

СПИСОК ЛИТЕРАТУРЫ

1. *Самарский А.А.* Теория разностных схем. М.: Наука, 1977.
2. *Самарский А.А., Гулин А.В.* Устойчивость разностных схем. М.: Наука, 1973.
3. *Самарский А.А., Николаев Е.С.* Методы решения сеточных уравнений. М.: Наука, 1978.

4. *Арделян Н.В., Космачевский К.В., Черниговский С.В.* Вопросы построения и исследования полностью консервативных разностных схем магнитной газодинамики. М.: Изд-во Моск. ун-та, 1987.
5. *Страуструп Б.* Язык программирования C++. СПб.; М.: Невский Диалект–Издательство БИНОМ, 1999.
6. *Саблин М.Н., Арделян Н.В.* Двумерная операторно-разностная схема газовой динамики в лагранжевых координатах на нерегулярной треугольной сетке, обладающая свойством локальной аппроксимации вблизи оси симметрии // Прикл. матем. и информатика. 2002. № 10. 15–33.
7. *Саблин М.Н., Арделян Н.В.* Операторная сеточная аппроксимация задач двумерной газовой динамики в подвижных координатах на нерегулярной сетке // Прикл. матем. и информатика. 2002. № 11. 5–37.
8. *Самарский А.А., Тишкин В.Ф., Фаворский А.П., Шашков М.Ю.* Операторные разностные схемы // Дифф. уравнения. 1981. **17**, № 7. 1317–1327.
9. *Попов Ю.П.* Разностные методы решения задач газовой динамики. М.: Наука, 1992.
10. *Самарский А.А., Колдоба А.В., Повещенко Ю.А., Тишкин В.Ф., Фаворский А.П.* Разностные схемы на нерегулярных сетках. Минск: ЗАО “Критерий”, 1996.
11. *Галанин М.П., Попов Ю.П.* Квазистационарные электромагнитные поля в неоднородных средах. М.: Наука, 1995.
12. *Михайлова Н.В., Тишкин В.Ф., Тюрина Н.Н., Фаворский А.П., Шашков М.Ю.* Численное моделирование двумерных газодинамических течений на сетке переменной структуры // ЖВМ и МФ. 1986. **26**, № 9. 1392–1406.
13. *Четверушкин Б.М.* Математическре моделирование задач динамики излучающего газа. М.: Наука. 1985.
14. *Головизнин В.М., Самарский А.А., Фаворский А.П.* Вариационный подход к построению конечно-разностных моделей в гидродинамике // ДАН СССР. 1977. **235**, № 6. 1285–1288.
15. *Арделян Н.В., Гущин И.С.* Об одном подходе к построению полностью консервативных разностных схем // Вестник МГУ. Вычисл. матем. и киберн. 1982. № 3. 3–10.
16. *Арделян Н.В., Космачевский К.В.* Неявный свободно-лагранжевый метод расчета двумерных магнито-газодинамических течений // Математическое моделирование. М.: Изд-во Моск. ун-та, 1993. 25–44.
17. *Арделян Н.В.* Об использовании итерационных методов при реализации неявных разностных схем двумерной магнитной гидродинамики // ЖВМ и МФ. 1983. **23**, № 6. 1417–1426.
18. *Арделян Н.В., Космачевский К.В., Козлов Н.П., Попов Ю.П., Протасов Ю.С., Самарский А.А., Чувашев С.Н.* Численное моделирование и теоретические исследования излучающих плазмодинамических разрядов // Радиационная плазмодинамика. Ч. 1. М.: Энергоиздат, 1991. 191–250.
19. *Ardeljan N.V.* Iterative methods for solving implicit difference schemes of MHD // Z. Angew. Math. Mech. 1996. **76**. 123–126.
20. *Арделян Н.В., Саблин М.Н.* Итерационный метод для системы операторных уравнений, возникающих при решении неявных разностных схем газовой динамики // Вестник МГУ. Вычисл. матем. и киберн. 1993. № 4. 7–12.
21. *Зенкевич О., Морган К.* Конечные элементы и аппроксимация. М.: Мир, 1986.

Поступила в редакцию
07.02.2006