ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ КВАДРАТУРНЫХ МЕТОДОВ

В.А. Семенов, С.В. Морозов

В статье исследуются возможности применения объектного подхода к программированию квадратурных методов. Проводится объектный анализ задач и методов численного интегрирования функций. Строится единая объектная классификация, включающая широкие семейства квадратурных формул, алгоритмов интегрирования, полиномов и отражающая общую методологию конструирования квадратурных методов с использованием классических многочленов. Обсуждаются вопросы проектирования и реализации библиотеки классов для численного интегрирования функций на основе развитого подхода. Основные объектные парадигмы программирования квадратурных методов иллюстрируются примерами на языке Си++.

1. Введение

Методы численного интегрирования функций играют важную роль как в самой вычислительной математике, так и в ее многочисленных приложениях [1–3]. Существующие пакеты программ для решения данного класса задач [4–7] написаны на процедурно-ориентированных языках и не удовлетворяют требованиям расширяемости и модифицируемости, предъявляемым в настоящее время к математическому обеспечению [8, 9]. Определенный теоретический и практический интерес в связи с этим приобретает применение объектно-ориентированного подхода (ООП) к программированию квадратурных методов, поскольку он предоставляет более широкие инструментальные возможности для развития уже разработанного программного обеспечения. Мотивация настоящей работы тем более убедительна, что известные исследования охватывают в основном вопросы применения объектной технологии к программированию других классов вычислительных задач и, прежде всего, задач линейной алгебры и дифференциальных уравнений в частных производных [8].

Проведенный объектный анализ теории численных методов [1–3] выявил фундаментальную роль классических интерполяционных многочленов в подходах, применяемых при численном интегрировании функций. Причем роль полиномов не ограничивается только теоретическими построениями и соответствующими обос-

нованиями. Они также используются в качестве практических методик для конструирования квадратурных формул, организации алгоритмов численного интегрирования, получения необходимых оценок вычислительных погрешностей. По существу любой приближающий многочлен, для которого определены процедуры интерполяции или аппроксимации и существует аналитическая первообразная, порождает целое семейство квадратурных методов. Данное обстоятельство побудило нас следовать указанной методологической общности как при объектном анализе, так и при объектном проектировании разрабатываемого математического обеспечения.

В разделе 2 проводится общий объектный анализ задач и методов численного интегрирования функций и обосновывается целесообразность включения классов интерполяционных многочленов в проектируемую объектную систему. В разделе 3 приводится возможная объектная классификация полиномов, включающая необходимые семейства интерполяционных многочленов. В разделе 4 обсуждаются известные подходы к организации численных методов интегрирования и дается их объектная интерпретация. Данные методы рассматриваются в качестве объектов алгоритмического суперкласса, реализующего обобщенный итерационный вычислительный процесс. Раздел 5 посвящен вопросам построения объектной классификации квадратурных формул и ее программной реализации на языке Си++.

2. Объектный анализ задач и методов интегрирования функций

Задача численного интегрирования может быть сформулирована следующим образом: вычислить определенный интеграл $\int_a^b f(x)\,dx$ по значениям функции f_k и ее производных $f_k^{(m)}$ в точках $x_k \in [a,b], \quad k = 1, ..., n$ с заданной точностью в рамках отведенных ресурсов (как правило, используются только значения самой функции).

Существуют два различных подхода к организации методов численного интегрирования. Первый основан на применении известных квадратурных формул вида

$$\int_a^b f(x)\,dx \approx \sum_{k=1}^{n} w_k\ f(x_k)\,,$$

где величины w_k называются весами, а x_k — узлами квадратуры. Для вывода подобных формул применяется процедура

интерполяции подынтегральной функции одним из классических полиномов (обычно многочленом Лагранжа), первообразная которого выражается аналитически.

Второй подход заключается в непосредственном построении полинома $\varphi(x)$, интерполирующего подынтегральную функцию: $\varphi(x_k) = f(x_k)$, и вычислении интеграла по следующей формуле

$$\int_a^b f(x)\,dx \approx \int_a^b \varphi(x)\,dx = \Phi(b) - \Phi(a),$$

где $\Phi(x)$ — первообразная $\varphi(x)$. Данный подход часто используется в геометрических приложениях, в которых подынтегральная функция представляется или приближается сплайнами разных видов.

Объектный анализ постановки задачи численного интегрирования и отмеченных подходов к ее решению приводит к объектной системе, представимой классом скалярных функций одной переменной **ScalarUniVariateFunction** и частным, но концептуально важным классом полиномов **Polynomial**. Поскольку в приложениях требуется вычислять интегралы и от векторных функций одной переменной, в данную объектную систему следует включить соответствующий класс **VectorUniVariateFunction**. В дальнейшем будем считать, что для функциональных объектов определены операции вычисления значений функции, ее производных, интегралов, а также поиска нулей и экстремумов. Вопросы проектирования и реализации классов скалярных функций обсуждались нами в работе [10]. Поскольку полиномиальные объекты рассматриваются как частные случаи функций, то к ним применим весь репертуар перечисленных функциональных методов. Вместе с тем, для полиномиальных классов дополнительно определены необходимые процедуры приближений (интерполяции и аппроксимации).

Существенной чертой применяемого подхода к построению математической объектной системы [9] является включение в нее самих алгоритмов численного интегрирования и квадратурных формул, объединяемых классами **IntegrationAlgorithm** и **QuadratureFormula** соответственно. И алгоритмы, и квадратуры рассматриваются в качестве самостоятельных объектов, участвующих как в постановке исходной задачи, так и в методах ее решения.

Таким образом, определена объектная система для решения задач численного интегрирования. Центральное место в ней занимают полиномиальные классы. Рассмотрим особенности их реализации в рамках единой библиотеки.

3. Библиотека полиномиальных классов

Предлагаемая библиотека полиномов является достаточно развитой системой абстрактных и конкретных классов, которые представляют классические семейства многочленов и выражают их фундаментальные математические и вычислительные свойства. Данная библиотека предназначена для разработки различных программных приложений, связанных, прежде всего, с решением задач приближений функций, а также многих других численных проблем, которые редуцируются к ним. Библиотека организована в виде целостной иерархии функциональных и полиномиальных классов (см. рис. 1) в соответствии с известными классификационными признаками, принятыми в вычислительной математике [1–3].

Абстрактный класс **Polynomial**, реализующий понятие обобщенного многочлена, является производным от класса **ScalarUniVariateFunction** и наследует от него следующие виртуальные методы функциональных объектов $f: \mathbf{R} \to \mathbf{R}$:

— вычисление значения функции в заданной точке $f(x)$,

— дифференцирование (вычисление значений первой, второй и *n*-ой производных $f'(x),\ f''(x),\ f^{(n)}(x)$),

— интегрирование $\int_a^b f(x)\,dx$,

— поиск нулей на заданном интервале $f(x) = 0,\ x \in [a, b]$,

— поиск экстремумов на заданном интервале $\underset{[a,b]}{extr}\ f$.

Вычисление значений полинома осуществляется оператором ( ), объявленным как чисто виртуальный метод абстрактного класса **ScalarUniVariateFunction**. Данный оператор должен переопределяться в каждом конкретном полиномиальном классе. Остальные методы допускают реализацию уже на функциональном уровне с использованием известных численных алгоритмов. Вместе с тем, может оказаться целесообразным их переопределение в частных полиномиальных классах, когда существует возможность их более эффективной или точной реализации. Например, процедуры дифференцирования или интегрирования, реализуемые в функциональном классе только численным образом, могут быть переопределены в классах степенных многочленов аналитически. Особенности подобных частных реализаций обсуждаются в следующих подразделах.

- **ScalarUniVariateFunction**
  - **Polynomial**
    - **Canonical**
      - **Linear**
      - **Quadratic**
      - **Cubic**
      - **Quartic**
    - **Interpolational**
      - **Irregular**
        - **Lagrange**
        - **Newton**
        - **Hermite**
      - **Uniform**
        - **UniformLagrange**
        - **ForwardNewton**
        - **BackwardNewton**
        - **Gauss**
        - **Gauss2**
        - **Stirling**
        - **Bessel**
      - **Rooted**
      - **ChebyshevRooted**
    - **OrthogonalSystem**
      - **Classical**
        - **Hermite**
        - **GeneralizedLaguerre**
          - **Laguerre**
        - **Jacobi**
          - **Chebyshev Of First, Second, Third, Fourth Kind**
          - **Legendre Of First, Second Kind**
          - **Gegenbauer**
      - **NonClassical**
    - **Special**
      - **BernsteinBezier**
      - **Rational**
      - **Fractional**
      - **RationalBezier**
    - **Trigonometric**
      - **Fourier**
    - **Spline**
      - **LinearSpline**
      - **CubicSpline**
      - **BSpline**
      - **RationalBSpline (NURBS)**

Рис. 1. Иерархия полиномиальных классов

Наряду с вышеперечисленными методами, интерфейс класса **Polynomial** включает:

— методы, оперирующие с числом параметров многочлена и осуществляющие доступ к ним,

— методы интерполяции и аппроксимации.

Последние группы методов специфицируют вычислительные процедуры как чисто виртуальные и реализуются на нижних уровнях

иерархии многочленов. Такой способ позволяет в наибольшей степени сблизить функциональные интерфейсы и тем самым достичь желаемой унификации в организации родственных полиномиальных классов.

Рассмотрим кратко особенности реализации конкретных полиномиальных классов. Мы намеренно опускаем некоторые детали частных реализаций, поскольку для обсуждаемых вопросов программирования методов интегрирования важно многообразие полиномиальных объектов, на основе которых может строиться тот или иной метод. Прежде всего, остановимся на классах канонических, интерполяционных и ортогональных многочленов, которые являются основой для построения квадратурных формул. Другие семейства полиномов могут применяться при непосредственном интегрировании функций в рамках второго упомянутого в предыдущем разделе подхода.

3.1. Класс канонических полиномов

Канонические полиномы являются наиболее распространенными в приложениях функциональными объектами. К семейству **Canonical** будем относить обычные степенные многочлены, представленные в канонической форме $P_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$. Обсудим некоторые наиболее важные стороны реализации класса **Canonical**, следуя известным особенностям канонического степенного представления.

Во-первых, репертуар методов данного класса следует расширить алгебраическими операциями сложения, вычитания, умножения, деления и нахождения остатка от деления со скалярными и полиномиальными операндами. Так как канонические полиномы образуют группу не только относительно алгебраических операций, но и относительно операций дифференцирования и интегрирования, то в набор методов могут быть включены также операции конструирования полинома, являющегося производной или первообразной данного многочлена.

Во-вторых, поскольку производные и интеграл канонических многочленов выражаются аналитически, а для поиска нулей и экстремумов существуют специальные численные подходы, основанные на теоретических оценках числа нулей и способах их практической локализации, целесообразно переопределить соответствующие методы родительского функционального класса более эффективным образом.

В-третьих, в классе **Canonical** можно реализовать методы интерполяции и аппроксимации, определяемые полиномиальным

суперклассом. Следует заметить, что подобная реализация не является формальной, поскольку канонические полиномы невысокой степени иногда непосредственно применяются для решения задач приближения. В общем случае приближающего многочлена произвольной степени используются другие методики, связанные с применением специальных степенных представлений, обсуждаемых ниже. Основой возможной реализации процедуры интерполирования в данном классе может служить метод неопределенных коэффициентов, а процедуры аппроксимации — метод наименьших квадратов.

Таким образом, полный интерфейс класса **Canonical** включает методы, реализующие:

— бинарные арифметические операции +, –, *, / со скалярными и полиномиальными операндами,

— операцию % нахождения остатка от деления двух многочленов,

— унарные арифметические операции,

— операции понижения степени многочлена (его деление на линейный или квадратичный полином),

— простые и арифметические операции присваивания,

— нахождение многочлена, являющегося *n*-ой производной или первообразной данного,

— интерполяцию и аппроксимацию,

а также переопределенные методы функционального суперкласса, осуществляющие:

— доступ к коэффициентам полинома a_k, $k = 1, ..., n$,

— вычисление, дифференцирование, интегрирование, поиск нулей и экстремумов многочлена.

Отметим некоторые детали внутренней организации данного класса. Коэффициенты многочлена хранятся внутри класса **Canonical** как вектор класса **Vector**, поддерживающий необходимые векторные арифметические операции и операции BLAS 1. Программирование полиномиальных операций через векторные позволяет достичь определенной унификации в использовании важной части алгоритмического ядра линейной алгебры.

Наряду с самостоятельной реализацией, методы приближения оформляются как конструкторы класса **Canonical**. Это является достаточно естественным, поскольку целью решения любой задачи приближения является конструирование соответствующего полиномиального объекта. В качестве аргумента в данные методы передается приближаемая сеточная функция и требуемый порядок приближающего многочлена.

В иерархии приведены четыре частных класса канонических полиномов, наследуемых непосредственно от **Canonical**. Классы **Linear**, **Quadratic**, **Cubic** и **Quartic** реализуют канонические многочлены первого, второго, третьего и четвертого порядков соответственно. Необходимость введения полиномиальных классов специальных порядков связана с тем, что процедуры вычисления их корней и экстремумов могут быть переопределены на нижнем уровне более эффективным образом, поскольку корни многочленов до четвертой степени включительно выражаются аналитически на основе известных формул.

3.2. Класс интерполяционных полиномов

Специальные интерполяционные полиномы являются особой формой степенного многочлена, по существу представляя ту же группу математических объектов, что и канонические полиномы. Вместе с тем, отличия в способе представления затрагивают существенные свойства интерполяционных семейств, что делает целесообразным выделение их в отдельную группу. Мотивацией такого классификационного деления могут служить следующие аргументы.

Во-первых, большинство перечисленных процедур, которые реализуются методами классов канонических полиномов, не может непосредственно распространяться на специальные степенные представления. Для их применения исходный многочлен нужно предварительно привести к каноническому виду, что является вычислительно трудоемкой операцией и кажется довольно искусственным для библиотечных реализаций. Отмеченный способ при необходимости может использоваться в самих приложениях.

Во-вторых, в отличие от канонических полиномов интерполяционные многочлены не образуют групп относительно арифметических операций. Формальное же определение подобных операций для объектов данных классов не отражает их алгебраические свойства и представляется нецелесообразным.

В-третьих, область практического применения данных полиномов ограничивается в основном интерполяционными и смежными с ними задачами. Желание в наибольшей степени унифицировать спецификации методов, осуществляющих решение эквивалентных задач интерполирования, приводит к организации самостоятельной подиерархии родственных интерполяционных классов.

Обсудим особенности программирования задач интерполяции с помощью данных полиномиальных классов. Как правило, процесс

применения интерполяционных многочленов включает два отдельных этапа:

— построение приближающего многочлена (или, что то же самое, определение его параметров), которое составляет собственно задачу интерполирования,

— использование построенного многочлена при решении задач численного анализа или прикладных задач, включающее вычисление его значений, производных, интегралов, нулей и экстремумов.

Инкапсуляция интерполяционными классами методов приближения и численного анализа, а также необходимых данных внутреннего представления (коэффициентов и узлов интерполирования) позволяет достичь желаемой технологичности в их практическом применении.

В связи с этим кажется естественным включение в интерфейс класса интерполяционных полиномов **Interpolational** следующих дополнительных методов:

— доступ к коэффициентам интерполяционной формулы и значениям узлов,

— преобразование в каноническую форму,

— интерполяция,

— оценка остаточного члена интерполяционной формулы.

Все перечисленные процедуры данного класса определяются чисто виртуальными и реализуются в наследуемых конкретных классах.

Многообразие методов интерполяции, определяемое наличием большого числа интерполяционных формул, а также возможностью применения различных способов выбора узлов и разнообразных алгоритмических схем, приводит к необходимости более развитой классификации интерполяционных объектов. Поскольку построение интерполяционных формул в существенной мере зависит от расположения узлов интерполирования, удобным оказывается деление полиномов на три группы, которые соответствуют наиболее часто используемым вариантам выбора узлов, а именно, произвольного (нерегулярного), равномерного и в нулях многочленов.

Данная классификация положена в основу подиерархии специальных интерполяционных полиномов. Абстрактный класс **Interpolational** имеет четырех непосредственных наследников: **Irregular**, **Uniform**, **Rooted** и **ChebyshevRooted**. Первые два класса представляют абстрактные интерполяционные формулы с нерегулярным и равномерным расположением узлов интерполирования соответственно. Третий класс реализует особую форму представления произвольного степенного полинома по

значениям его корней и фактически задает приближение функции с нулевыми значениями в узлах. Класс **ChebyshevRooted** реализует оптимальный интерполяционный многочлен с узлами в корнях ортогонального полинома Чебышева.

Такое деление позволяет реализовать методы доступа к коэффициентам и значениям узлов уже в абстрактных классах **Irregular** и **Uniform**. Вместе с тем, оно не исключает возможности дальнейшей более глубокой классификации интерполяционных формул, построение которых диктуется способом выбора узлов. Так, в представленной полиномиальной иерархии группа интерполяционных многочленов с произвольным выбором узлов **Irregular** представлена классами полиномов Лагранжа **Lagrange**, Ньютона **Newton** и Эрмита **Hermite**. Группа интерполяционных многочленов с равномерным выбором узлов **Uniform** образуется семействами полиномов Лагранжа **UniformLagrange**, интерполяционных схем Ньютона **ForwardNewton**, **BackwardNewton**, а также полиномов Гаусса, Стирлинга и Бесселя **Gauss**, **Gauss2**, **Stirling**, **Bessel**.

3.3. Класс ортогональных полиномов

Ортогональные полиномы являются одним из важнейших семейств многочленов, используемых для решения задач численного анализа, включая задачи приближения. Рассмотрим наиболее существенные вопросы проектирования данной группы классов, затрагивающие фундаментальные свойства ортогональных многочленов.

Класс **OrthogonalSystem** реализует наиболее распространенную форму выбора приближающей функции в виде линейной комбинации абстрактных ортогональных полиномов

$$\varphi(x) = \sum_{i=0}^{m} c_i \, \varphi_i(x) .$$

При этом с данным классом мы будем связывать и обобщенный многочлен, представленный в ортогональном базисе, и соответствующую ему ортогональную систему. Представление базисных полиномиальных объектов $\varphi_k(x)$ осуществляется путем выбора частных значений коэффициентов: $c_k = 1; \quad c_i = 0, i \neq k$.

Единые правила построения ортогональных систем полиномиальных функций приводят к возможности обобщенной программной реализации численных методов, оперирующих с ортогональными многочленами. Она может строиться на основе выделения ядра базовых процедур, к которым редуцируются все основные алгоритмы. Механизм виртуальных функций

обеспечивает поддержку подобной редукции при определении базовых операций как чисто виртуальных методов класса **OrthogonalSystem** и их реализации в конкретных классах. Тогда общие процедуры, использующие эквивалентные численные методики для всех ортогональных семейств, могут быть реализованы один раз как методы данного абстрактного класса.

С базовыми процедурами ортогональных систем многочленов будем связывать следующие методы:

— выдачу границ интервала ортогональности [*a*, *b*],

— вычисление значений весовой функции $\rho(x)$ и ее производных,

— вычисление коэффициентов отдельного многочлена $\varphi_k(x)$ в каноническом степенном базисе,

— определение взвешенной нормы d_k полиномиального элемента $\varphi_k(x)$,

— определение коэффициентов α_k, β_k рекуррентного соотношения, связывающего три последовательных полинома ортогональной системы

$$\varphi_{k+1}(x) = (x - \alpha_k)\varphi_k(x) - \beta_k \varphi_{k-1}(x), \quad \varphi_0(x) = 1, \varphi_{-1}(x) = 0, k = 0,1,2,\ldots.$$

К общим процедурам, реализуемым в классе **OrthogonalSystem**, будем относить:

— доступ к коэффициентам полинома c_k,

— вычисление, дифференцирование, интегрирование, поиск нулей и экстремумов полинома $\varphi(x)$ и базисных многочленов $\varphi_k(x)$,

— интерполяцию и аппроксимацию,

— преобразование в каноническую форму полинома $\varphi(x)$ и базисных многочленов $\varphi_k(x)$,

— конструирование ортогонального полинома $\varphi(x)$ по заданной канонической форме.

Заметим, что методы, оперирующие с отдельным базисным многочленом $\varphi_k(x)$, являются перегруженными вариантами методов, относящихся ко всей ортогональной линейной комбинации $\varphi(x)$, и отличаются от последних наличием дополнительного параметра, определяющего порядок заданного многочлена.

К наиболее часто применяемым системам ортогональных функций следует отнести семейства классических полиномов Якоби, Лагерра и Эрмита, представляемые в иерархии классами **Jacobi**, **GeneralizedLaguerre** и **Hermite** соответственно. Заметим, что частные семейства многочленов Чебышева **Chebyshev**, Лежандра **Legendre** и Гегенбауэра **Gegenbauer** строятся как классы, наследуемые от общего класса **Jacobi**, отражая тем самым

параметрическую общность полиномов Якоби. Неклассические ортогональные системы имеют более узкое распространение, прежде всего, из-за отсутствия эффективных методов их конструирования. Поскольку классические ортогональные многочлены имеют особые математические свойства, а конструирование неклассических полиномов основывается на специальных численных подходах, выделение их в различные группы носит принципиальный характер. Данные группы ортогональных многочленов представляются в рассматриваемой иерархии классами **Classical** и **NonClassical**.

3.4. Классы специальных полиномов

Особые группы многочленов в приведенной иерархии образуют классы специальных полиномов **Special**, тригонометрических полиномов **Trigonometric** и сплайнов **Spline**. Они представляют интерес в рамках настоящей статьи, прежде всего, потому, что могут использоваться при организации методов непосредственного интегрирования функций на основе их предварительной интерполяции с применением данных полиномиальных семейств. Опуская детали их внутренней организации, отметим лишь, что для объектов данных классов определены необходимые процедуры интерполяции и интегрирования.

Специальная группа **Special** представлена конкретными классами **BernsteinBezier**, **Rational**, **Fractional**, **RationalBezier**, реализующими специфическое степенное представление Бернштейна-Безье, семейства рациональных полиномов, цепной дроби и рациональные многочлены Безье. Группу сплайнов **Spline** составляют кусочно-линейные функции **LinearSpline**, кубические сплайны **CubicSpline**, а также обычные и рациональные B-сплайны **BSpline**, **RationalBSpline**. Группа тригонометрических полиномов **Trigonometric** формально представлена единственным классом многочленов Фурье **Fourier**, хотя допускает достаточно развитую классификацию.

4. Объектная организация алгоритмов численного интегрирования

Обсудим вопросы построения целостной иерархии классов, реализующих различные алгоритмы интегрирования функций. Поскольку методы численного интегрирования являются алгоритмами итерационного типа, имеет смысл начать построение данной иерархии от суперкласса **IterativeAlgorithm**, реализующего

обобщенный итерационный вычислительный процесс. Далее при построении алгоритмов класса **IntegrationAlgorithm** будем следовать двум рассмотренным в разделе 2 подходам к численному интегрированию, а именно, подходу, основанному на использовании квадратурных формул и осуществляемому классом **QuadratureAlgorithm**, и подходу, использующему непосредственную интерполяцию подынтегральной функции в рамках алгоритмической стратегии **InterpolationalAlgorithm**. Квадратурные методы допускают дальнейшее деление на адаптивные и неадаптивные алгоритмические схемы **Adaptive**, **NonAdaptive**, а также на экстраполяцию Ромберга **RombergExtrapolation** [1, 2]. Таким образом, предлагается следующая иерархия алгоритмов численного интегрирования функций, представленная на рис. 2.

- **IterativeAlgorithm**
  - **IntegrationAlgorithm**
    - **QuadratureAlgorithm**
      - **Adaptive**
        - **Horizontal**
        - **Vertical**
      - **NonAdaptive**
      - **RombergExtrapolation**
    - **InterpolationalAlgorithm**

Рис. 2. Объектная классификация алгоритмов численного интегрирования

Рассмотрим особенности реализации классов, образующих данную иерархию.

4.1. Класс IterativeAlgorithm

Поскольку основные численные методы являются алгоритмами итерационного типа, имеет смысл построить суперкласс обобщенных итерационных алгоритмов **IterativeAlgorithm**, определяющий основные алгоритмические компоненты итерационного вычислительного процесса. При этом область применения данного класса не должна ограничиваться квадратурными задачами и его проектирование обязано обеспечивать достаточную общность, которая позволила бы использовать **IterativeAlgorithm** при программировании любых других итерационных алгоритмов численного анализа, линейной алгебры и т.п. Обсудим возможную организацию алгоритмического суперкласса.

Методы абстрактного класса **IterativeAlgorithm** соответствуют основным стадиям итерационного цикла и виртуально определяют следующие алгоритмические компоненты:

— инициализацию вычислений цикла,

— выполнение одного итерационного шага,

— проверку условий останова вычислений в цикле в случаях достижения точности, невозможности обеспечить сходимость вычислительного процесса или исчерпания отведенных задаче ресурсов,

— возможные априорные (теоретические) оценки ресурсов, необходимых для достижения заданной точности, а также обратные оценки точности по имеющимся ресурсам,

— возможные апостериорные (практические) оценки достигнутой точности,

— переход к новому шагу цикла,

— постобработку решения.

Достаточно общим можно признать также включение в алгоритмический суперкласс членов-данных, определяющих требуемую (задаваемую пользователем) и достигнутую (реальную или оцениваемую в момент решения) точность вычислений. Поскольку о вычислительной точности имеет смысл говорить только в контексте затрат, необходимых для решения задачи, в **IterativeAlgorithm** включены также данные, определяющие вычислительные ресурсы задачи, отведенные для нее пользователем и реально потраченные при ее решении. Таким образом, будем считать, что класс **IterativeAlgorithm** определяет необходимые методы доступа к перечисленным данным и оперирования с ними и непосредственно инкапсулирует следующие объекты:

— требуемую точность,

— достигнутую точность,

— пользовательские ресурсы,

— затраченные ресурсы.

Поскольку анализ точности вычислений и оценка ресурсов задач могут проводиться довольно различными способами, целесообразно рассматривать их в качестве самостоятельных объектов и определить соответствующие классы **Tolerance** и **Resource**. Их обобщенная реализация позволяет при необходимости изменить стратегию ведения расчетов путем переопределения части методов в наследуемых классах.

Для объектов класса **Tolerance** определим методы, реализующие:

— доступ к значениям абсолютной, относительной и пороговой точности,

— анализ точности, включающий элементарные операции сравнения.

Для объектов класса **Resource** определим методы, реализующие:

— доступ к значениям числа итераций, числа элементарных арифметических операций, числа вычислений функции, ее производных,

— анализ ресурсов, включающий элементарные операции сравнения.

Ниже приведен возможный вариант построения класса **IterativeAlgorithm** на языке Си++. В методе инициализации вычислений Initialize( ) производится отключение значений достигнутой точности и обнуление затраченных ресурсов, а также оценивается предельное число итераций по заданным пользователем значениям ресурсов. Метод Run( ) выполняет сам итерационный алгоритм. Метод проверки условий окончания цикла While( ) использует три критерия останова вычислений — при достижении заданной точности результата, при исчерпании ресурсов, отведенных решаемой задаче, а также при отсутствии сходимости в итерационном процессе, что является признаком некорректного применения алгоритма в конкретной постановке.

```
class IterativeAlgorithm
{
protected:
    Tolerance toler;        // tolerance — требуемая точность
    Tolerance accur;        // accuracy — достигнутая точность
    Resource requir;        // required — заданные ресурсы
    Resource expend;        // expended — затраченные ресурсы
public:
…
    virtual void CalcNumberOfIteration( ) = 0;
    virtual Bool isAchievedTolerance( ) { return (accur>toler) ? False : True; }
    virtual Bool isExceededResource( ) { return (expend<requir) ? False : True; }
    virtual Bool isDisConvergence( ) { return False; }
    virtual void Initialize( )
        { accur.ExcludeAccuracy( ); expend.ExcludeResource( ); CalcNumberOfIteration( ); }
    virtual Bool While( )
        { return (isAchievedTolerance( ) ||
                  isExceededResource( ) ||
                  isDisConvergence( )) ? False : True; }
    virtual void Next( ) = 0;
    virtual void MakeIteration( ) = 0;
    virtual void Completion( ) = 0;
    void Run( );
    // теоретические априорные оценки
    virtual Tolerance ToleranceEstimation(Resource& r) = 0;
    virtual Resource ResourceEstimation(Tolerance& t) = 0;
```

```
    // практические апостериорные оценки
    virtual Tolerance AchievedTolerance( ) = 0;
    virtual Resource ExpendedResource( ) = 0;
…
};
```

Реализация самого итерационного алгоритма тогда выглядит тривиальным образом:

```
void IterativeAlgorithm::Run( )
    { for (Initialize( ); While( ); Next( ))   MakeIteration( );
      Completion( ); }
```

Заметим, что основные методы класса **IterativeAlgorithm**, определяющие ключевые этапы итерационного алгоритма и, прежде всего, этапы вычисления самой итерации и перехода к следующему шагу, оформлены как чисто виртуальные процедуры, которые необходимо реализовать в конкретных алгоритмических классах. Использование базового класса в данном случае позволяет унифицировать интерфейсы всех классов итерационных алгоритмов, а также реализовать на самом верхнем уровне алгоритмической иерархии часть методов, связанных с анализом достижения точности, сходимости, контролем вычислительных ресурсов.

4.2. Классы алгоритмов интегрирования функций

Абстрактный класс **IntegrationAlgorithm** представляет обобщенный алгоритм численного интегрирования функций с заданной точностью и заданными вычислительными ресурсами. Он является производным от суперкласса обобщенных итерационных алгоритмов **IterativeAlgorithm** и наследует от него данные и методы, необходимые для организации итерационного процесса. Интерфейс данного класса определяет следующие дополнительные группы процедур:

— вычисление интегралов скалярных и векторных функций одной переменной на заданном интервале,

— определение порядка метода интегрирования,

— апостериорная оценка порядка метода интегрирования,

— апостериорная оценка погрешности результата.

По существу интерфейс класса специфицирует общие пользовательские процедуры, связанные с решением задачи интегрирования. Детали внутренней организации алгоритмов на данном уровне скрываются, и их конкретизация переносится в частные классы.

Рассмотрим возможную реализацию класса **IntegrationAlgorithm** на языке Си++.

```
class IntegrationAlgorithm : public IterativeAlgorithm
{
protected:
    ScalarUniVariateFunction* pf;      // указатель на интегрируемую функцию
    double a;                          // нижний и верхний пределы интегрирования
    double b;
    double result;                     // приближенные значения интеграла
    double res1;                       // на текущей и двух предыдущих итерациях
    double res2;
public:
…
    virtual double ApplyTo(ScalarUniVariateFunction& f, double lower, double upper);
    virtual int OrderOfMethod( ) = 0;
    virtual int AchievedOrderOfMethod( );
    virtual Tolerance AchievedTolerance( );
…
};
```

Для организации взаимодействия между отдельными стадиями итерационного цикла, которые реализуются методами Initialize( ), While( ), Next( ), MakeIteration( ) и Completion( ) класса **IterativeAlgorithm**, класс **IntegrationAlgorithm** инкапсулирует приведенные члены-данные. Вышеперечисленные процедуры должны быть переопределены в каждом конкретном алгоритмическом классе в соответствии с конкретной методикой организации процесса интегрирования. Метод ApplyTo( ) предназначен для вычисления интеграла от указанной функции, и его реализация сводится к инициализации данных класса и перевызову метода Run( ).

Заметим, что процедуры апостериорной оценки погрешности и порядка метода интегрирования могут быть реализованы уже в абстрактном классе с использованием известных формул Рунге и Эйткена, которые не зависят от типа и порядка конкретной применяемой квадратуры или интерполяционного многочлена. Процедура апостериорной оценки погрешности интегрирования AchievedTolerance( ) строится на формуле Рунге

$$R = \frac{w_h - w_{kh}}{k^p - 1},$$

а процедура определения порядка метода AchievedOrderOfMethod( ) — на формуле Эйткена

$$p \ln k = \ln \frac{w_{kh} - w_{k^2 h}}{w_h - w_{kh}}.$$

В данных формулах w_h, w_{kh}, w_{k^2h} — величины интеграла, вычисленные с шагом h, kh, k^2h соответственно (т.е. на текущей и двух предшествующих итерациях), а *p* — порядок метода.

Абстрактный класс **QuadratureAlgorithm** реализует наиболее важную группу алгоритмов численного интегрирования функций, основанных на применении квадратурных формул. Принимая во внимание разнообразие известных квадратур, а также возможность конструирования новых с учетом специфики интегрируемых функций, проектирование данного класса должно обеспечивать его полную инвариантность по отношению к различным квадратурным типам **QuadratureFormula**. Для этого в классе **QuadratureAlgorithm** предусмотрен указатель на объект класса **QuadratureFormula** и инкапсулированы процедуры конструирования алгоритма по ссылке на произвольную квадратуру. Тогда для определения порядка метода интегрирования OrderOfMethod( ) и априорной оценки его погрешности ToleranceEstimation(Resource& r) на основе формул для остаточного члена квадратуры могут использоваться соответствующие процедуры, инкапсулируемые классом **QuadratureFormula**.

В приведенной иерархии квадратурные алгоритмы подразделяются на адаптивные и неадаптивные. Класс **Adaptive** реализует обобщенный адаптивный алгоритм вычисления определенного интеграла по указанной квадратурной формуле, который автоматически выбирает величину шага интегрирования *h* так, чтобы результат удовлетворял предписанной точности. При реализации адаптивных алгоритмов используются различные схемы разбиения интервала интегрирования и порядка его обхода. Классическими вариантами являются, горизонтальная и вертикальная схемы [1], реализуемые классами **Horizontal** и **Vertical** соответственно.

Рассмотрим построение класса квадратурных методов на более простом примере неадаптивных алгоритмов **NonAdaptive**. Программирование данного класса сводится к реализации двух методов родительского класса **IterativeAlgorithm**, определяющих основные стадии итерационного цикла — вычисление итерации и переход к следующему шагу. Ниже приводится фрагмент программы на языке Си++.

```
class NonAdaptive : public QuadratureAlgorithm
{
    // от класса QuadratureAlgorithm наследуется указатель на применяемую квадратуру
    // QuadratureFormula* pquad;
protected:
    int n;     // число разбиений отрезка интегрирования
```

```
public:
…
    virtual void MakeIteration( );
    virtual void Next( );
…
};

void NonAdaptive::MakeIteration( ) {
    res2=res1;
    res1=result;
    double h=(b–a)/n;
    double a1=a;
    double b1=a+h;
    result=0.0;
    for(int i=0; i<n; i++) {
        result+=pquad–>Calculate(*pf, a1, b1);
        a1=b1;
        b1+=h;
    }
}

void NonAdaptive::Next( ) {
    accur= AchievedTolerance( );
    expend= ExpendedResource( );
    n*=2;
}
```

Класс **InterpolationalAlgorithm** реализует алгоритм численного интегрирования, основанный на непосредственной интерполяции подынтегральной функции. Он инкапсулирует указатель на объект абстрактного класса **Polynomial**, задающий тип приближающего многочлена, а также методы конструирования алгоритма по ссылке на заданный полином. Передавая в конструктор многочлены конкретных типов, можно строить разнообразные варианты интерполяционных алгоритмов численного интегрирования. На практике чаще всего применяются алгоритмы, основанные на интерполяции сплайнами или классическими степенными многочленами. Методы полиномиального класса (интерполяция, вычисление и интегрирование полинома) используются при реализации процедуры интегрирования произвольной функции в классе **InterpolationalAlgorithm**.

Поскольку вычисление интегралов таблично заданных функций тесно связано с описанным классом алгоритмов, представляется целесообразной перегрузка метода интегрирования для табличных функциональных объектов. Ниже приведена простейшая реализация данного метода на языке Си++.

```
double InterpolationalAlgorithm::ApplyTo(ScalarUniVariateTableFunction& f, double a, double b) {
    // класс InterpolationalAlgorithm инкапсулирует член Polynomial* poly;
    poly–>Interpolate(f) ;
    return poly–>Integrate(a, b);
}
```

Интегрируемая табличная функция передается по ссылке на объект класса **ScalarUniVariateTableFunction**, содержащий массивы значений аргумента и самой функции. Для вычисления интеграла используются методы интерполяции Interpolate( ) и интегрирования многочлена Integrate( ), инкапсулируемые полиномиальным классом. Погрешность результата в данном случае оценивается по остаточному члену интерполяционного многочлена и значению старшей производной предположительно гладкой функции.

5. Объектно-ориентированное программирование квадратурных формул

Перейдем к центральному вопросу программирования квадратурных методов в рамках проводимого подхода — построению самих квадратур как объектов соответствующих классов. Рассмотренные выше обобщенные классы алгоритмов интегрирования были разработаны таким образом, чтобы обеспечить возможность вычислений без какой-либо конкретизации вида используемой квадратурной формулы. Такая программная общность может быть поддержана путем организации различных квадратур в виде целостной классовой иерархии с единой вершиной, представляющей квадратурный суперкласс.

5.1. Иерархия классов квадратурных формул

Рассмотрим представленную на рис. 3 иерархию классов, включающую широкие семейства известных квадратурных формул [1–3].

Вершиной данной иерархии является класс **QuadratureFormula**. Он определяет и частично реализует вычислительные процедуры, связанные с использованием обобщенной квадратурной формулы

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\sum_{i=1}^{n} w_i\, f(x_i) + C(b-a)^{p+1} f^{(p)}(\xi) + O((b-a)^{p+2}),$$

где $x_i \in [a,b]$ — узлы квадратуры, w_i — ее весовые коэффициенты. Узлы и веса квадратурной формулы определяются принятым для нее способом приближения подынтегральной функции. В качестве приближающей функции применяется полином порядка *n*–1 (обычно интерполяционный многочлен Лагранжа). Величина $R_p = C(b-a)^{p+1} f^{(p)}(\xi),\ \ \xi \in [a,b]$ является главным членом погрешности квадратурной формулы. Параметр *C* есть некоторый коэффициент,

зависящий только от конкретной квадратуры. Число *p*, которое определяется порядком приближающего многочлена *n*–1, называется порядком квадратурной формулы.

- **QuadratureFormula**
  - **NewtonCotes**
    - **Rectangle**
    - **Trapezium**
    - **Simpson**
    - **Newton**
  - **GaussType**
    - **GaussChristoffel**
      - **Gauss**
      - **Hermite(Mehler)**
    - **GaussRadau**
    - **GaussLobatto**
  - **Chebyshev**
  - **Euler**
  - **Gregori**
  - **Patterson**
  - **Filon**
  - **MonteCarlo**

Рис. 3. Объектная классификация квадратурных формул

Интерфейс абстрактного класса **QuadratureFormula** виртуально определяет следующие группы методов:

— доступ к значению числа узлов квадратуры *n*,

— определение порядка квадратурной формулы *p*,

— доступ к значениям узлов и весовых коэффициентов квадратуры $x_i, w_i, i = 1,...,n$,

— конструирование квадратурной формулы заданного порядка (вычисление значений узлов и весов),

— вычисление квадратур скалярных и векторных функций одной переменной на заданном интервале,

— оценка главного члена погрешности квадратуры R_p с использованием константы *C*,

— оценка вычислительных ресурсов, необходимых для проведения расчетов.

Заметим, что в суперклассе квадратур может быть реализована процедура вычисления интеграла по обобщенной квадратурной формуле путем использования виртуальных методов доступа к значениям узлов и весов. В том случае, если имеется более экономичная методика вычисления квадратуры, например, при равенстве значений весов квадратурной формулы Чебышева, данная процедура будет переопределяться на нижних уровнях иерархии. Переопределение целесообразно и с практической точки зрения для исключения нежелательных вызовов виртуальных

функций, особенно методов доступа к узлам и коэффициентам, представляющим элементарные операнды квадратурного выражения.

Наследники класса **QuadratureFormula** реализуют конкретные квадратурные формулы и инкапсулируют необходимые данные для хранения значений узлов и весовых коэффициентов. Инкапсуляция этих данных в квадратурном суперклассе является нецелесообразной, поскольку большинство квадратур использует совершенно различные методики определения весов и узлов, что влечет за собой необходимость применения разных способов их хранения.

Наиболее часто для интегрирования функций применяются семейства квадратур Ньютона–Котеса и квадратурные формулы гауссовского типа, представленные в иерархии классами **NewtonCotes** и **GaussType** соответственно. Рассмотрим подробно особенности реализации данных классов и их наследников.

Класс **NewtonCotes** реализует семейство квадратурных формул Ньютона–Котеса, основанное на равномерной интерполяции подынтегральной функции многочленом Лагранжа степени *n*–1

$$\int_a^b f(x)\,dx = \frac{(b-a)}{2}\sum_{i=1}^{n} w_i\, f(x_i) + C(b-a)^{p+1}\, f^{(p)}(\xi),$$

в которой порядок квадратуры определяется выражением $p = 2\left\lfloor \frac{n+1}{2} \right\rfloor$. Веса и коэффициент главного члена погрешности квадратуры Ньютона–Котеса вычисляются по следующим обобщенным формулам

$$w_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{t-d_j}{d_i-d_j}\,dt, \quad C = \frac{1}{p!\,2^{p+1}} \int_{-1}^{1} (t-d_1)\ldots(t-d_n)\,dt,$$

в которых узлы d_i приведены к интервалу [–1, 1] линейным отображением. Веса хранятся в векторе внутри класса **NewtonCotes**.

Квадратурные формулы внутри данного семейства отличаются друг от друга степенью используемого интерполяционного полинома. Частными случаями квадратур Ньютона–Котеса являются широко известные формулы средних прямоугольников (интерполяция константой), трапеций (интерполяция линейным многочленом), Симпсона (интерполяция квадратичным многочленом) и Ньютона (интерполяция кубическим многочленом), реализуемые классами **Rectangle**, **Trapezium**, **Simpson** и **Newton** соответственно. Использование приведенных выше общих формул для вычисления частных квадратур и оценки их погрешности крайне неэффективно. Например, вычисление интеграла в классе **Simpson** могло бы осуществляться по более простой формуле

$$\int_a^b f(x)\,dx = \frac{(b-a)}{6}\left(f(a) + 4\, f\left(\frac{a+b}{2}\right) + f(b)\right) - \frac{(b-a)^5}{2880}\, f^{(4)}(\xi)\,.$$

Поэтому основные методы класса **NewtonCotes** переопределяются в наследуемых квадратурных классах.

Абстрактный класс **GaussType** представляет семейство квадратурных формул гауссовского типа, основанное на использовании неравноотстоящих узлов, расположенных в нулях ортогональных многочленов. Данные формулы называются также квадратурами наивысшей алгебраической точности, поскольку подобная методика выбора узлов обеспечивает минимальную погрешность результата. В данном классе инкапсулируются два вектора, необходимые для хранения значений узлов и весов гауссовых квадратур, а также реализуются виртуальные методы доступа к этим значениям. Кроме того, класс **GaussType** инкапсулирует указатель на класс **OrthogonalSystem** и процедуру конструирования квадратурной формулы на основе заданной системы ортогональных многочленов.

Класс **GaussChristoffel** реализует практически важное семейство квадратур Гаусса–Кристоффеля

$$\int_a^b f(x)\rho(x)\,dx = \frac{b-a}{2}\sum_{i=1}^{n} w_i\, f(x_i) + C(b-a)^{2n+1} f^{(2n)}(\xi)\,,$$

где узлы $x_i, i = 1,\text{K},n$ являются нулями полинома *n*-го порядка, ортогонального с весом $\rho(x)$ на интервале [*a*, *b*], возможно, бесконечном. Веса и коэффициент остаточного члена квадратуры определяются посредством интегрирования интерполяционного многочлена Лагранжа с весом $\rho(x)$ и выражаются следующим образом

$$w_i = \frac{2}{b-a}\int_a^b \left(\prod_{\substack{j=1\\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}\right)^2 \rho(x)\,dx\,,\; C = \frac{1}{(2n)!\,(b-a)^{2n+1}}\int_a^b (x-x_1)^2 \ldots (x-x_n)^2 \rho(x)\,dx\,,$$

На практике обычно применяется другая методика вычисления весовых коэффициентов квадратур гауссовского типа, основанная на решении проблемы собственных значений для матрицы, составленной из коэффициентов рекуррентных формул для ортогональных многочленов [7]. Подобная методика может использоваться в процедуре конструирования гауссовых квадратур для произвольно заданной весовой функции, которая реализуется классом **GaussType**.

Частными классическими случаями квадратурных формул Гаусса–Кристоффеля являются квадратуры Гаусса и Эрмита, реализуемые классами **Gauss** и **Hermite** соответственно. Первая

формула конструируется на основе ортогональных многочленов Лежандра ($\rho(x)=1, [a,b]=[-1,1]$), а вторая — на основе ортогональных многочленов Чебышева ($\rho(x)=1/\sqrt{1-x^2}, [a,b]=[-1,1]$). В обоих классах переопределяются методы конструирования квадратур и оценки погрешности, поскольку веса и коэффициенты остаточных членов вычисляются аналитически ($w_i=\pi/n$ для квадратуры Эрмита; формула для вычисления весов квадратуры Гаусса приводится в следующем подразделе).

Классы **GaussRadau** и **GaussLobatto** реализуют соответственно квадратурные формулы Гаусса–Радау и Гаусса–Лобатто. Квадратуры Гаусса–Радау отличаются от квадратур Гаусса–Кристоффеля наличием дополнительного узла, совпадающего с нижней границей интервала *a*. В квадратурных формулах Гаусса–Лобатто в качестве дополнительных узлов используются обе границы *a*, *b*.

Классы **Patterson**, **Chebyshev**, **Euler**, **Gregori**, **Filon** реализуют квадратурные формулы Паттерсона, Чебышева, Эйлера, Грегори и Филона соответственно. Организация данных классов близка организации рассмотренных выше классов, поскольку все эти квадратуры имеют одну и ту же интерполяционную природу. Заметим, что квадратурные формулы Филона используются для интегрирования быстро осциллирующих функций и базируются на тригонометрической интерполяции.

Класс **MonteCarlo** реализует статистическую квадратурную формулу Монте-Карло

$$\int_a^b f(x)\,dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i),$$

где узлы x_i выбираются с помощью генератора случайных чисел. Очевидно, что данная квадратура организована аналогично квадратурам интерполяционного типа (имеет узлы, равные весовые коэффициенты). Поэтому оказалось возможным ее включение в общую иерархию квадратурных формул.

5.2. Пример объектной реализации квадратур

Рассмотрим пример реализации на языке Си++ класса **Gauss**, представляющего известную квадратурную формулу Гаусса. Данная квадратура связана с ортогональной системой многочленов Лежандра $P_n(x)$. Веса w_i и коэффициент остаточного члена *C* вычисляются по формулам [1]:

$$w_i = \frac{2(1-x_i^2)}{n^2\,(P_{n-1}(x_i))^2}, \quad C = \frac{(n!)^4}{((2n)!)^3\,(2n+1)}.$$

Заметим, что реализация конкретной квадратуры сводится к определению класса, наследуемого от класса **QuadratureFormula** или одного из его потомков, и реализации метода вычисления интеграла в соответствии с приведенными формулами. Предполагается, что определение значений весов, узлов и коэффициента главного члена погрешности осуществляется при конструировании соответствующего квадратурного объекта или при изменении порядка квадратуры.

```
class Gauss : public GaussChristoffel
{
    // от класса QuadratureFormula наследуются данные:
    //  int n;                          число узлов квадратуры
    //  double coefterm;                коэффициент главного члена погрешности
    // от класса GaussType наследуются следующие данные:
    // Vector w;                        веса квадратуры
    // Vector x;                        узлы квадратуры
    // OrthogonalSystem* orthpol;       указатель на ортогональную систему
public:
…
    Gauss( ) { NewQuadrature(3); }
    Gauss(int nnod) { NewQuadrature(nnod); }
    virtual void NewQuadrature(int nnod);
    virtual double Calculate(ScalarUniVariateFunction& f, double a, double b);
…
};
```

Поскольку квадратурные формулы гауссовского типа строятся на основе ортогональных многочленов, класс **GaussType** инкапсулирует указатель на ортогональный полином orthpol. Следует заметить, что большую часть методов класс **Gauss** непосредственно наследует от предшествующих классов (методы оперирования с числом узлов квадратуры — от класса **QuadratureFormula**, методы доступа к значениям весов и узлов — от класса **GaussType**, методы определения порядка квадратуры и оценки погрешности — от класса **GaussChristoffel**). Отметим, что для оценки погрешности на основе формулы для остаточного члена используются два перегруженных метода, один из которых оперирует со ссылкой на интегрируемую функцию **ScalarUniVariateFunction**, а второй — с явно заданным пользователем значением старшей производной.

Рассмотрим более подробно реализацию процедур конструирования квадратуры Гаусса и вычисления интеграла заданной функции по данной квадратурной формуле.

```
void Gauss::NewQuadrature(int nnod) {
    n=(nnod>=1)? nnod : 3;
// вычисление весов и узлов квадратуры
    double coef=2.0/(nnod*nnod);
    orthpol=new Legendre(nnod);
    x=orthpol–>Zeros(nnod);
```

```
    for(int i=0; i<nnod; i++) {
        double tmp=(*orthpol)(x[i], nnod–1);
        w[i]=coef*(1–x[i]*x[i])/(tmp*tmp);
    }
// вычисление коэффициента остаточного члена
…
}

double Gauss::Calculate(ScalarUniVariateFunction& f, double a, double b) {
    double coef1=0.5*(a+b);
    double coef2=0.5*(b–a);
    double result=0.0;
    for(int i=0; i<n; i++)
        result+=w[i]*f(coef1+coef2*x[i]);
    result*=coef2;
    return result;
}
```

В процедуре конструирования квадратуры Гаусса строится приближающий ортогональный многочлен Лежандра *n*-го порядка как объект класса **Legendre**, и методы данного класса используются для вычисления узлов и весовых коэффициентов квадратурной формулы. Поскольку класс **Legendre** реализует линейную комбинацию соответствующих полиномов (см. разд. 3.3), в его методах явно указывается необходимый порядок многочлена с помощью одного из параметров.

6. Заключение

Включение в общую классификационную схему объектов, представляющих различные классы алгоритмов интегрирования, различные семейства полиномов и квадратурных формул, позволило достичь желаемой общности как при теоретической объектной систематизации обсуждаемых разделов вычислительной математики, так и при практической реализации соответствующего математического обеспечения.

Разработанная классификация носит конструктивный характер и допускает непосредственную программную реализацию. Приведенные примеры иллюстрируют лишь некоторые парадигмы объектно-ориентированного программирования методов интегрирования с ее использованием. Среди них следует особо выделить рассмотренные способы обобщенной реализации различных алгоритмических стратегий для произвольных (абстрактных) квадратур и приближающих многочленов, причем с необходимыми априорными и апостериорными оценками погрешности. Другим содержательным примером является реализация квадратур гауссовского типа по произвольным (абстрактным) ортогональным полиномам.

Развитый подход к программированию методов численного интегрирования апробирован при практической реализации математического обеспечения. Разработанная на языке Си++ система классов включена в состав математической объектно-ориентированной библиотеки [9]. Математическая и вычислительная наглядность программных средств библиотеки, возможности их расширения, модификации и адаптации с использованием объектных парадигм делают ее достаточно привлекательной инструментальной средой для разработки вычислительных приложений в разнообразных научных и технических областях.

Работа поддержана Российским Фондом фундаментальных исследований (грант 95–01–01239).

ЛИТЕРАТУРА

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. — М.: Наука, 1987.
2. Самарский А.А., Гулин А.В. Численные методы. — М.: Наука, 1989.
3. Никольский С.М. Квадратурные формулы. — М.: Наука, 1988.
4. Piessens R., de Doncker-Kapenga E., Überhuber C.W., Kahaner D.K. QUADPACK, A Subroutine Package for Automatic Integration. Series in Computational Mathematics 1. Springer-Verlag, New York, 1983.
5. Favati P., Lotti G., Romani F. Algorithm 691: Improving QUADPACK Automatic Integration Routines. // ACM Trans. Math. Softw., v. 17, № 2 (June 1991), pp. 218–232.
6. IQPACK: FORTRAN Subroutines for the Weights of Interpolatory Quadratures. // ACM Trans. Math. Softw., v. 13, № 4 (Dec. 1987).
7. Gautschi W. Algorithm 726: ORTHPOL — A Package of Routines for Generating Orthogonal Polynomials and Gauss-Type Quadrature Rules. // ACM Trans. Math. Softw., v. 20, № 1 (March 1994), pp. 21–62.
8. Proceedings of the Second Annual Object-Oriented Numerics Conference. Sunriver, Oregon, 1994.
9. Семенов В.А. Об объектно-ориентированном подходе к разработке численного математического обеспечения. // Вопросы кибернетики. Приложения системного программирования. — М.: НСК РАН, 1995, с. 140–163.
10. Морозов С.В., Семенов В.А. Объектно-ориентированное программирование задач численного анализа. // Вопросы кибернетики. Приложения системного программирования. — М.: НСК РАН, 1995, с. 189–211.