ТЕХНОЛОГИИ РЕАЛИЗАЦИИ РАЗРЕЖЕННЫХ МАТРИЧНЫХ КЛАССОВ

В.А. Семенов, О.А. Тарлапан

В работе рассматриваются вопросы объектного программирования больших разреженных задач линейной алгебры. Строится единая иерархия матричных объектов, охватывающая широкие классы элементарных, специальных и произвольных матриц, и допускающая ее непосредственную программную реализацию. Особое внимание уделяется вопросам унифицированной разработки матричного обеспечения, ориентированного на существенно различные формы разреженности. Обсуждаются результаты практической реализации векторно-организованных матричных классов на языке Си++.

1. Введение

Имеющиеся сегодня математические библиотеки и пакеты линейной алгебры являются в основном результатом довольно продолжительной эволюции процедурного программирования и не используют возможностей современных объектных технологий. Принципы создания и особенности их внутренней организации отражают прежде всего их проблемную и методическую ориентацию, в определенной степени препятствующую их дальнейшему развитию и адаптации к новым программным приложениям.

Так, пакеты LAPACK [1], LINPACK [2] и EISPACK [3] реализуют прямые методы решения линейных систем и проблем собственных значений с плотными матрицами, пакет ITPACK [4] предназначен для решения линейных проблем итеративными методами, программы ELLPACK [5] ориентированы на работу со структурными матрицами, возникающими в результате разностной дискретизации эллиптических дифференциальных задач, возможности пакета SPARSPAK [6] ограничены главным образом хаотически разреженными матрицами, пакет TOEPLITZ [7] обеспечивает работу с теплицевыми матрицами и т. п.

Естественной, в связи с этим, представляется попытка объектной систематизации матричного программного обеспечения (ПО), результатом которой должна стать единая программная инструментальная среда, допускающая работу с существенно различными матричными объектами при использовании всего арсенала методов линейной алгебры. Объектно-ориентированный подход (ООП) предоставляет важные возможности для подобных программных обобщений. Целесообразно упомянуть коммерческие матричные системы MATLAB [8] и CLAM [9], принципы организации интерактивных интерфейсов которых очень близки положениям ООП.

В настоящей работе на основе выделения групп элементарных, специальных, структурных и параметрических признаков строится общая матричная классификация и обсуждаются общие принципы организации единой иерархии матричных классов. Значительное внимание уделяется технологиям, обеспечивающим высокую степень унификации разрабатываемых классов. Описывается практическая реализация векторно-организованных матричных классов на языке Си++ и приводятся сравнительные результаты их временного тестирования.

2. Объектный подход к организации матричного обеспечения

Центральным элементом теории линейной алгебры [12, 13], с которым связаны практически все методические построения, является, конечно, матрица. Поэтому ее использование в качестве основного объекта при программной реализации является достаточно естественным и вполне соответствует общим принципам ООП [11]. Применение объектных парадигм при разработке матричного ПО, в частности, наследования и полиморфизма базовых операций и методов линейной алгебры, инкапсулируемых соответствующими матричными классами, обеспечивает качественно новый уровень технологии программирования.

Требования интеграции и унификации объектно-ориентированного матричного ПО приводят к необходимости разработки единой иерархии матричных классов. При такой организации матричный суперкласс выражает наиболее общие свойства матрицы как фундаментального математического объекта и определяет или реализует ее базовые операции и методы, а наследуемые классы — свойства, операции и методы, соответствующие частным матричным объектам.

Предположим, что арифметические матричные операции и базовые операции линейной алгебры, составляющие известный пакет BLAS [10], определены для матричного суперкласса в виде множества чисто виртуальных функций и реализуются для всех наследуемых конкретных классов. Тогда сложные вычислительные алгоритмы, включая прямые и итеративные методы решения линейных систем и проблем собственных значений, будучи редуцированными к базовым операциям, могут быть реализованы как методы абстрактных матричных классов на самых верхних уровнях иерархии и обеспечивать желаемую общность постановок и методов решения линейных задач. С другой стороны, вычислительно мощные операции линейной алгебры как методы нижних классов иерархии допускают эффективную реализацию, так как при этом могут учитываться математические свойства частных матричных объектов и особенности конкретных структур данных.

Тем самым достигается важный компромисс между компактностью и эффективностью матричного ПО, необходимые при одновременной работе с матричными объектами различных типов. Заметим, что всякий раз, когда требования к эффективности особенно высоки, а общий метод плохо структурирован и не редуцируется к базовым операциям, возможно его переопределение и целостная реализация на самых нижних уровнях классовой иерархии.

Переопределение общих методов особенно целесообразно, когда наличие специальных матричных свойств, принятых в качестве классификационных, позволяет применить более эффективные алгоритмические версии, предназначенные для решения задач со специальными матрицами. Использование универсальных методов в таких случаях, как правило, крайне неэффективно. Например, для треугольной факторизации симметричной матрицы более разумной являлась бы реализация разложения Холесского, а не общего алгоритма LU-разложения.

С базовыми методами матричных классов будем связывать группы методов, реализующие:

— доступ к элементам матриц,
— унарные и бинарные алгебраические матричные операции,
— базовые операции линейной алгебры BLAS 2, 3.

К общим методам, реализуемым как методы матричного суперкласса, будем относить

— прямые и итеративные методы решения систем линейных уравнений,
— методы решения задачи о наименьших квадратах,
— методы решения проблем собственных значений.

Естественно, принятое деление на частные и общие методы достаточно условно, поскольку базовые матричные операции могут быть реализованы и на абстрактном уровне с использованием методов доступа к элементам, а реализация общих методов — переопределена для конкретных классов. Тем не менее, оно отражает достаточно общий подход к организации объектно-ориентированного матричного ПО, при котором реализация любой матричной операции может быть перенесена на необходимый иерархический уровень в соответствии с предъявляемыми требованиями универсальности и эффективности. Заметим, что подобная организация алгоритмов и программ не препятствует реализации машинно-зависимых версий, обычно использующих структуризацию в рамках пакета BLAS.

Обсудим особенности реализации и применения методов доступа к элементам, как ключевых при работе с разреженными матричными объектами. По-видимому, возможны два основных метода доступа к элементам матриц, соответствующих способам передачи параметров по ссылке и по значению. Поскольку для полноценной работы с матричными объектами требуется возможность модификации их элементов, реализация метода, возвращающего ссылку на элемент матрицы по заданным индексам, является необходимой. Однако в случае отсутствия элемента в структуре разреженной матрицы такой метод должен предварительно зарезервировать элемент, что требует крайней осторожности и анализа характера его алгоритмического применения с учетом возможностей заполнения.

В дальнейшем будет показано, что выполнение алгоритмов линейной алгебры, выражаемых в конечном счете элементарными аддитивными и мультипликативными

операциями, всегда возможно при итерировании ненулевых элементов соответствующих разреженных матричных и векторных операндов. В связи с этим, важной оказывается организация специальных классов матричных итераторов, реализующих необходимые методы итерирования по строкам, столбцам и диагоналям соответствующих матриц. Организация таких классов в виде самостоятельной параллельной иерархии позволяет унифицировать работу с итераторами, соответствующими различным матричным объектам, и достаточно компактно реализовать все многообразие алгебраических операций над разнотипными матричными объектами.

Таким образом, применение ООП к разреженному матричному программному обеспечению выражается в организации параллельных иерархий классов матриц и родственных им классов матричных итераторов, оптимизирующих доступ к ненулевым элементам.

3. Матричная классификация

Обсудим характерные вопросы построения единой целостной иерархии матричных классов. Возможности для классификационных построений здесь самые разнообразные. Тем не менее, описываемое ниже множество матричных признаков является достаточно широким и охватывает наиболее важные и практически содержательные случаи матричных объектов. В дальнейшем будем использовать терминологию, принятую в теории матриц [12, 13].

Будем подразделять все матричные признаки условно на четыре группы, а именно на элементарные, специальные, параметрические и структурные. Принимая во внимание программный аспект реализации матричных объектов, определим и пятую группу, выражающую возможные способы организации матричных структур данных для математически эквивалентных объектов.

Под параметрическими признаками здесь будем понимать нетривиальные условия корреляции значений элементов матрицы. К параметрическим признакам будем относить условия ортогональности матриц Orthogonal, нормальности Normal и унитарности Unitary в комплексном случае, положительности Positive, диагонального преобладания DiagonallyDominant, знакоопределенности Definite, регулярности Regular, обусловленности Conditioned, диагонализуемости Diagonalizable, дефектности Defective, разложимости Reducible, цикличности Cyclic, идемпотентности Idempotent, стохастичности Stochastic.

Довольно часто параметрические признаки выражаются в форме явной корреляции значений элементов и порождают матрицы специального вида. В этих случаях их естественная математическая классификация может быть явным образом перенесена на классовую матричную иерархию. Специальные матрицы носят обычно вспомогательный характер, но играют заметную конструктивную роль как в теории матриц, так и в практике матричных вычислений. По-видимому, выделение специальных матричных классов со своими уникальными свойствами позволило бы придать программам линейной алгебры алгоритмически более выразительный характер.

Специальную группу составляют симметричные матрицы Symmetric, эрмитовы Hermitian в комплексном варианте, циркулянтные матрицы Circulant, теплицевы Toeplitz, ганкелевы матрицы Hankel, матрицы Вандермонда Vandermonde, матрицы перестановок Permutation, ковариационные матрицы Covariance, корреляционные матрицы Correlation, жордановы клетки JordanBlock и матрицы канонической жордановой формы Jordan, матрицы Грамма Gram, индикаторные матрицы Indicator, матрицы инерции Inertion, а также частные варианты матрицы Гильберта Hilbert.

К элементарным будем относить признаки специальных матриц, определяющие известные элементарные матричные преобразования. Поскольку почти все алгоритмы линейной алгебры редуцируются в конечном счете к элементарным матричным преобразованиям, использование данных признаков в качестве классификационных и организация соответствующих специальных классов позволяет, в частности, исключить функциональное представление базовых алгебраических операций и реализовать сложные алгоритмы в наглядной операторной форме, близкой к математической нотации.

К элементарным отнесем единичные матрицы Identity, перъединичные матрицы BackwardIdentity, матрицы транспозиций (элементарных перестановок) Transposition, матрицы масштабирования Scal, Axpy, основные циркулянтные матрицы BasicCirculant, матрицы Хаусхолдера Householder, матрицы вращений Якоби Jacobi и Гивенса Givens.

Важные для практических реализаций частные параметрические свойства равенства нулю определенным образом расположенных элементов матрицы отнесем в отдельную группу и будем называть структурными. Фактически, структурные признаки определяют локализацию ненулевых элементов в портрете матрицы. Формально к этой группе признаков будем относить также и условия на размерности матриц. Основные структурные признаки составляют свойства регулярного заполнения в рамках диагональной Diagonal, трехдиагональной TriDiagonal, верхней треугольной UpperTriangular, ленточной Band, верхней хессенберговой UpperHessenberg, фронтальной Frontal, блочной Block, а также свойство произвольного хаотического заполнения Chaotic.

Заметим, что выше перечислены лишь основные матричные свойства. Некоторые из них допускают широкое толкование и порождают целые группы близких понятий. Например, для знакоопределенных матриц целесообразно ввести признаки полуопределенности и положительной определенности. Аналогично, свойство матричной обусловленности, выражаемое известной параметрической мерой, может уточняться соответствующими понятиями хорошей, плохой, совершенной обусловленности.

Наконец, существует довольно широкое многообразие способов реализации структур данных или матричных форматов, ориентированных, как правило, на определенные виды разреженности матриц и характер оперирования с ними. Обычно плотные матрицы и регулярно структурируемые матрицы организуются как одномерные массивы, индексирование в которых соответствует последовательному итерированию элементов матриц по строкам, по столбцам и по диагоналям.

Часто в приложениях, например, в задачах вычислительной геометрии, возникает необходимость быстрого решения линейных задач невысокого порядка. Применение общих матричных классов для этих целей было бы крайне неэффективным. Полезной, в связи с этим, оказывается реализация плотных квадратных матриц Matrix22, Matrix33 порядка 2 и 3 соответственно, учитывающая данный размерный фактор и использующая более эффективные алгоритмические варианты.

Для хаотически разреженных матриц разработано гораздо больше форматов [14], хотя, как правило, используются координатный формат Coordinate, формат Кнута Knut и строчный формат Row с теми же вариантами, что и в случае плотных структур. В любом случае важным представляется то обстоятельство, что алгоритмы линейной алгебры, ориентированные на определенные способы матричного представления, претерпевают заметные модификации, которые необходимо принимать во внимание при их программной реализации и, в частности, при выделении и организации частных матричных классов.

Признаки всех перечисленных групп могут быть использованы в качестве классификационных критериев и тем самым определять архитектуру матричной иерархии в зависимости от представляющейся степени важности и от очередности их выделения. На наш взгляд требования неизбыточности матричных представлений и эффективности использования численных методов, хотя бы для самых распространенных постановок задач линейной алгебры, являются ключевыми и в значительной степени определяют структуру иерархии.

Требование неизбыточности представлений главным образом возникает в связи со структурными, специальными и элементарными свойствами матриц, допускающим хранение элементов особым экономичным способом. При этом элементарные матрицы используются только как внутренние базовые средства для реализации необходимых матричных преобразований и совсем не связаны с постановками проблемных задач. Структурные свойства в методическом отношении являются вспомогательными и появляются в основном на промежуточных этапах алгоритмов. Для решения же наиболее важных и общих задач линейной алгебры, к которым следует отнести решение линейных систем и проблему собственных значений с произвольными матрицами, ключевую роль играют специальные свойства и прежде всего свойства симметричности и эрмитовости в случае определения матриц на кольце комплексных чисел.

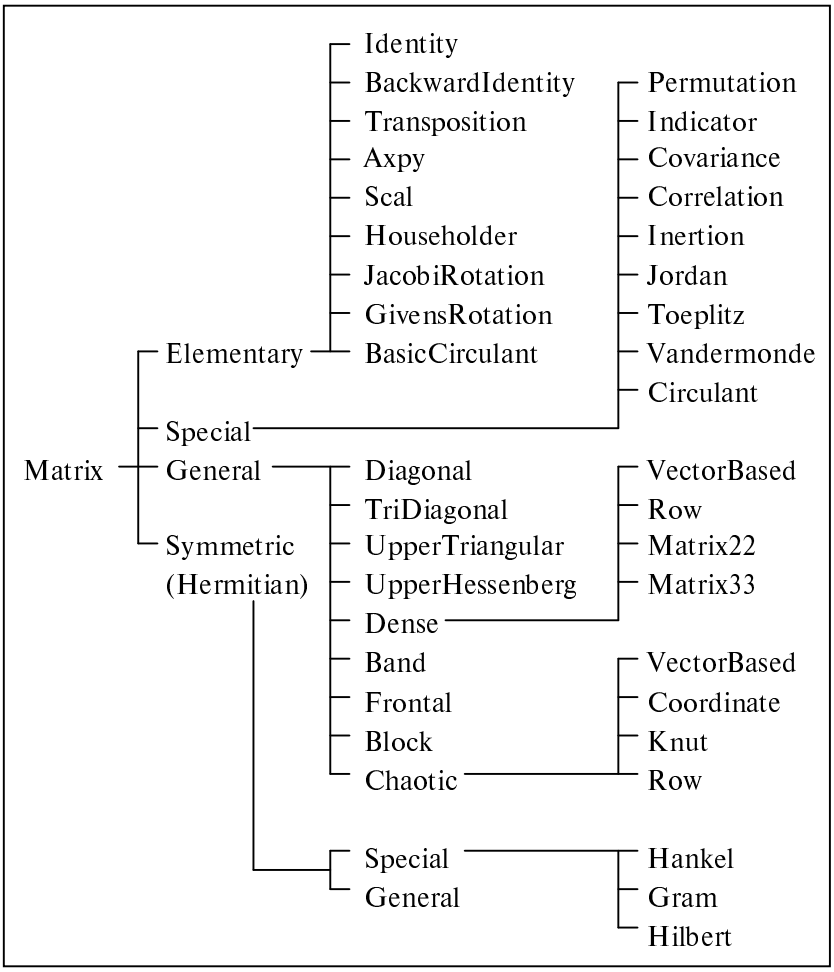

Рис. 1. Иерархия матричных классов

Таким образом, предлагается следующая иерархия матричных объектов, допускающая непосредственную программную реализацию (см. рис. 1). На рисунке не приведены симметричные элементарные и структурные матрицы, поскольку их перечень в значительной степени повторяет соответствующие несимметричные варианты. Построенная общая математическая классификация матриц учитывает специальные, элементарные и структурные свойства. Параметрические свойства не связаны с особенностями организации матричных данных и при построении данной иерархии во внимание не принимаются. Классификация может служить конструктивной основой для построения единой целостной иерархии матричных классов.

4. О реализации различных форм разреженности

Обсудим более подробно вопросы реализации разреженных матричных классов, следуя построенной выше классификации. Учет разреженности позволяет в большинстве случаев добиться значительной экономии памяти, а также заметно снизить время решения задач. Рассмотрим два возможных подхода к организации разреженных матричных классов.

Первый подход реализует обычную технологию программирования классов с использованием базовых типов данных, встроенных в язык программирования. В этом случае возможно достижение высокой эффективности разрабатываемых программ за счет использования непосредственной адресации элементарных скалярных операндов и исключения лишних вызовов функций. Серьезным недостатком данного подхода является его большая трудоемкость. Принимая во внимание достаточно широкое многообразие матричных объектов и необходимых для приложений методов линейной алгебры, надежная разработка развитой универсальной матричной системы только таким способом выглядит довольно проблематичной.

В качестве альтернативного подхода используем возможности алгоритмической редукции методов линейной алгебры к векторным операциям и представления различных моделей матричной разреженности векторной. В самом деле, пусть разреженная матрица представляется массивом разреженных векторов. Тогда любой метод линейной алгебры, будучи редуцированным к базовым векторным операциям, может быть реализован как метод общего матричного класса без конкретизации типов векторов, из которых он составлен. В рамках данного подхода нет принципиальных различий между организацией плотной матрицы и матрицы с хаотическим или регулярным характером заполнения. Тем самым возникает возможность обобщенной матричной реализации с использованием механизма абстрактных типов данных или шаблонов классов.

Очевидно, применение векторных операций BLAS 1 при реализации плотных задач является менее эффективным, чем использование более мощных операций BLAS 2, 3 или программирование методов линейной алгебры целиком [15]. Однако простота и полная унификация разработки всего спектра матричного ПО делают второй подход вполне реальной альтернативой первому и при таких реализациях. Для хаотически разреженных матриц, доступ к элементам которых возможен только через косвенную многоуровневую адресацию, а иногда и только путем поиска, применение обоих подходов, по-видимому, даст приблизительно одинаковые результаты по эффективности.

Рассмотрим принципы разработки векторно-организованных матричных классов. Основной задачей здесь видится выработка базового набора векторов, позволяющего эффективно представлять все формы матричной разреженности. Предлагается использовать четыре базовых типа векторов, обеспечивающих такое представление, а именно: плотный вектор DenseVector, ленточный вектор BandVector, индексный вектор IndexVector и списочный вектор ListVector.

DenseVector реализует классическое представление плотного вектора в виде линейного числового массива заданной размерности, хранящего все значения, в том числе и нулевые. Очевидно, данный вектор может эффективно представлять плотные матрицы Dense и ленточные матрицы Band в случае их диагональной организации.

Класс ленточного вектора BandVector плотно хранит только ненулевой сегмент вектора. Сегмент характеризуется двумя индексами, определяющими его нижнюю и верхнюю границы. Ленточный вектор можно использовать для хранения регулярно структурируемых матриц TriDiagonal, UpperTriangular, Band, Frontal, Block. Недостатком является необходимость хранения всех элементов сегмента, в том числе и нулевых.

Индексный вектор IndexVector хранит ненулевые элементы вектора и их индексы как плотные массивы. Класс индексного вектора целесообразно использовать в качестве базового для хранения хаотически разреженных матриц большой размерности со статическим, редко меняющимся портретом. Использование упорядоченного по индексу представления вектора позволяет резко сократить среднее время поиска элемента в портрете матрицы. Уместна аналогия со строчным форматом Row, широко применяемым для разреженных матриц.

Наконец, списочный вектор ListVector хранит ненулевые элементы вектора и их индексы в виде списка. Данное представление может эффективно применяться при работе с хаотическими, сильно разреженными матрицами с динамически меняющимся портретом, поскольку списочная организация позволяет легко добавлять и удалять элементы, расположенные в динамической памяти. В определенном смысле матричный формат, построенный на списочных векторах, подобен известному формату Кнута Knut с тем исключением, что последний поддерживает одновременно строчные и столбцовые списки элементов, а векторный формат — только одни из них.

Таким образом, предлагаемый базовый набор из четырех векторных классов поддерживает различные формы регулярной и хаотической разреженности, а следовательно, может обеспечить эффективную работу с произвольными матрицами. Следует однако заметить, что при работе с матричными объектами, имеющими специфические особенности, может быть рациональным расширение этого набора. При этом все разработанные ранее процедуры векторно-организованных матричных классов останутся в неизменном виде.

5. Классы разреженных векторов и итераторов

Большая частота использования методов векторных классов выдвигает особые требования к эффективности с одной стороны и простоте и надежности обращения с ними

с другой. При этом важной является эффективная реализация не только векторных операций, но и методов доступа к элементам, упрощающих и унифицирующих разработку самих операций BLAS 1. С учетом необходимости реализации операций со смешанными векторными операндами, данная задача может потребовать значительных усилий по программированию и заслуживает особого внимания.

Пусть доступ к элементам вектора осуществляется скобочным оператором [ ], возвращающим ссылку на значение элемента по заданному индексу. Применение такого оператора для реализации векторных операций практически невозможно, поскольку при отсутствии элемента необходимо предварительно зарезервировать его, что неприемлемо для заполнения. Другим недостатком является его неэффективность при последовательном сканировании элементов вектора, поскольку его выполнение связано с большим числом адресных операций и часто сопряжено со значительным поиском в разреженной структуре.

Указанная проблема решается путем введения вспомогательных классов итераторов, родственных соответствующим векторным, и определения для них методов сканирования только ненулевых элементов. В дальнейшем будем считать, что векторный оператор [ ] перегружен для объектов соответствующего класса итератора, а сам итератор в этом случае будем называть псевдоиндексным, поскольку его применение напоминает работу с обычным индексом.

На основании сделанных замечаний целесообразна следующая организация классов векторов и векторных итераторов, изображенная на рис. 2. Вершинами параллельных иерархий являются абстрактные классы вектора Vector и итератора VectorIterator, интерфейсы которых определяют общие свойства векторов, не затрагивая особенностей конкретных реализаций.

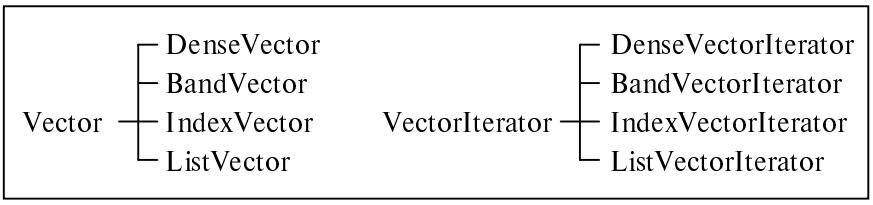

Рис. 2. Иерархия классов векторов и векторных итераторов

Интерфейс класса Vector включает группы методов, реализующих:

— определение и изменение размерности вектора,

— проверку равенства значения элемента нулю и установление наличия элемента в разреженной структуре,

— определение ненулевого сегмента вектора и числа ненулевых элементов в нем,

— доступ к элементам, включая перегруженный вариант оператора [ ] для индексного и псевдоиндексного операнда,

— конструирование эквивалентного векторного объекта и родственного ему итератора,

— унарные и бинарные арифметические операции,

— простые и арифметические операции присваивания и логические операции сравнения,

— операции BLAS 1, расширенные процедурами перестановки, удаления и обнуления элементов и допускающие возможность работы с отдельными сегментами.

Заметим, что большинство векторных операций может быть непосредственно реализовано уже на этом уровне с использованием методов доступа к элементам и прежде всего псевдоиндексного итератора. Однако еще более эффективным, но более трудоемким являлось бы переопределение векторных операций в конкретных классах с учетом деталей их внутренней организации. По-видимому, оптимальная стратегия должна состоять в реализации операций в абстрактном классе, обеспечивающей необходимую универсальность, и в переопределении только той части общих методов, к которым предъявляются повышенные требования эффективности.

Отметим особенность в реализации стандартного пакета базовых операций BLAS 1, которые вычисляют Ахру, скалярное произведение векторов, сумму и абсолютную сумму элементов вектора, евклидову норму, а также осуществляют вращение векторов, копирование и перестановку элементов. Использование векторов при организации матричных классов приводит к необходимости перегрузки данных операций для сегментов

векторов. Для этого список параметров в спецификации методов расширяется парой или тройкой индексов, соответствующих началу, концу сегмента вектора и смещению сегмента второго векторного операнда относительно первого.

Интерфейс абстрактного класса VectorIterator определяет методы, реализующие

— связь с итерируемым векторным объектом,

— установку на произвольный элемент, в частности, на первый, на последний ненулевой элемент, на элемент, индекс которого превышает заданный и т. п.,

— итерирование по ненулевым элементам,

— операции присваивания и логические операции сравнения с индексом и псевдоиндексом.

Уточним значение некоторых операций. При присваивании итератору индекса, итератор устанавливается на элемент вектора с заданным индексом. В случае отсутствия элемента он предварительно заводится в структуре вектора. При присваивании итератора, указывающего, возможно, на элемент другого векторного объекта, текущий итератор устанавливается на элемент с совпадающим индексом. Аналогичный смысл имеют логические операции сравнения итераторов и индексов.

Все перечисленные методы, естественно, реализуются только в конкретных классах итераторов. При этом удобна операторная форма реализации методов, повышающая их наглядность и упрощающая использование классов. Например, на языке Си++ операции итерирования, присваивания и сравнения могут быть реализованы с использованием стандартных операторов, а именно, + +, − −, =, <, > соответственно. Такая техника полностью унифицирует работу с обычными индексами и рассматриваемыми псевдоиндексными итераторами.

Перечисленные функции составляют на наш взгляд необходимый базовый набор методов, необходимых для работы с произвольными векторными объектами. При необходимости репертуар методов может быть легко расширен.

6. Некоторые примеры использования векторных итераторов

Использование разработанного псевдоиндексного итератора позволяет представить процедуры последовательного и поэлементного сканирования вектора синтаксически и семантически близкими программными циклами. Рассмотрим следующие два программных фрагмента на языке Си++, реализующих умножение вектора на скаляр.

```
// последовательное сканирование элементов вектора
for(Index i=0; i<vector.SizeOf(); i++)
  vector[i]*=scalar;
// сканирование ненулевых элементов вектора
for(ListVectorIterator i(vector); i<vector.SizeOf(); i++)
  vector[i]*=scalar;
```

Несмотря на кажущуюся эквивалентность программных циклов, они выполняются различным образом. В первом случае все элементы вектора vector умножаются на скаляр scalar, а во втором — только ненулевые, что является более эффективным. Заметим, что в данном случае применение итератора может оказаться более предпочтительным и для обычного плотного вектора, поскольку выполнение обычной операции индексирования требует одного сложения и одного умножения в индексной арифметике, а использование псевдоиндекса при последовательном итерировании элементов — только одного сложения.

Другим преимуществом псевдоиндекса является автоматическая поддержка разреженности. Если первый цикл применяется для работы с разреженным вектором, то результатом его выполнения будет полностью заполненный, что приводит к лишним затратам памяти и неприемлемо при решении больших задач. Удовлетворить данному требованию можно и при использовании обычного индекса, например, с помощью предварительной проверки нулевых элементов:

```
// vector*=scalar
for(Index i=0; i<vector.SizeOf(); i++)
  if(vector.isNonZero(i)==True)
    vector[i]*=scalar;
```

Однако поэлементный анализ наличия нулей в векторе часто сопряжен с дополнительными расходами, связанными с необходимостью многократного просмотра векторных данных.

Рассмотрим некоторые характерные примеры использования псевдоиндексного итератора при реализации алгебраических векторных операций. При этом ограничимся случаями реализации аддитивной и мультипликативной операции, к которым сводятся практически все остальные. Необходимо отметить, что обычное программирование операций над разреженными структурами является довольно трудоемким.

Следующая программа реализует операцию векторного сложения с присваиванием:

```
// vector1+=vector2
ListVector vector1,vector2;
ListVectorIterator i1(vector1),i2(vector2);
for(i1=i2=0; i2<vector2.SizeOf(); i1++) {
  if(i1==i2) {
    vector1[i1]+=vector2[i2];
    i2++;
  }
  else if(i1>i2) {
    i1=i2;
    vector1[i1]=vector2[i2];
    i2++;
  }
}
```

По-другому реализуется операция векторного умножения с присваиванием. В отличие от операции векторного сложения, при выполнении которой в разреженной структуре возможно лишь появление новых ненулевых элементов, при поэлементном умножении векторов ненулевые элементы могут только исчезнуть, что и реализуется соответствующими ветвями программ.

```
// vector1*=vector2
IndexVector vector1;
ListVector vector2;
IndexVectorIterator i1(vector1);
ListVectorIterator i2(vector2);
for(; i1<vector1.SizeOf(); ) {
  if(i1==i2) {
    vector1[i1]*=vector2[i2];
    i1++;
    i2++;
  }
  else if(i1>i2)
    i2++;
  else
    vector1.ZeroElement(i1);
}
```

Как и в предыдущем случае, программный фрагмент является компактным, а его понимание достаточно прозрачным. Важно отметить, что технология программирования (буквально, текст программы) не меняется при переходе к другим классам разреженных векторов. Более того, приведенные фрагменты могут использоваться для реализации алгебраических операций над разреженными объектами, принадлежащими разным векторным классам, что может активно использоваться при программировании разреженных задач линейной алгебры.

Таким образом, использование псевдоиндексного итератора позволяет существенно упростить и унифицировать разработку достаточно эффективных программ, реализующих алгебраические операции над разреженными объектами.

7. Векторно-организованные матричные классы

Рассмотрим вопросы реализации векторно-организованных матричных классов. Возможны три основных способа организации таких матричных объектов, соответствующих хранению элементов матрицы по строкам, по столбцам и по диагоналям. Это позволяет разделить семейство векторно-организованных матричных классов на три подсемейства и образовать иерархию, представленную на рис. 3.

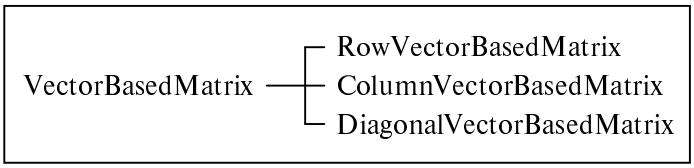

Рис. 3. Иерархия векторно-организованных матричных классов

Следующим уровнем иерархии должны были бы стать конкретные матричные классы, построенные на различных типах векторов. Однако значительная общность, достигнутая при разработке векторных классов, позволяет использовать механизм шаблонов. В данном случае RowVectorBasedMatrix, ColumnVectorBasedMatrix, DiagonalVectorBasedMatrix — это шаблоны классов, параметризуемые типами используемого вектора и векторного итератора. Достоинством этого механизма является не только значительное сокращение исходных текстов и полная унификация интерфейсов родственных матричных классов, но и обеспечение более высокой эффективности генерируемого кода, поскольку исключается виртуальное использование методов векторных классов. Например, реализация матричного шаблона ColumnVectorBasedMatrix имеет вид

```
template <class VectorType, class VectorIteratorType>
class ColumnVectorBasedMatrix : public VectorBasedMatrix {
  VectorType *vector;
  // описание класса
};
```

Одной из особенностей работы с векторно-организованными матричными классами является возможность двухуровневого индексирования элементов, т. е. использование доступа к элементу a_{ij} в виде a[i][j]. При обычном программировании это возможно лишь в случае использования многомерных массивов с жестко заданными размерами. Предлагаемая организация позволяет задавать матрицы с динамически изменяемыми размерами и использовать индексный доступ к элементам, при котором первый оператор индексирования возвращает ссылку на соответствующий векторный объект, а второй — на необходимый элемент. Этот вид доступа особенно удобен при RowVectorBasedMatrix, поскольку он полностью соответствует математической записи элементов матрицы.

Работа с разреженными матрицами предполагает обеспечение минимального уровня заполнения, что достигается путем применения специальных матричных алгоритмов и соответствующей реализации операций векторного класса. Операции индексирования, как указывалось ранее, являются неэффективным для практического использования. Одним из предлагаемых средств минимизации заполнения на программном уровне является применение матричных итераторов, обеспечивающих упорядоченное сканирование ненулевых элементов по строкам, столбцам и диагоналям. Реализация подобных классов полностью аналогична рассмотренной выше организации векторных итераторов. Собственно, пары конкретных классов матриц и соответствующих им итераторов и составляют набор методов, необходимый для полноценной работы с матричными объектами.

Для векторно-организованных матриц создание вспомогательных классов матричных итераторов представляется излишним, поскольку, как правило, удается применить специальные алгоритмические версии, ориентированные на строчное, столбцовое или диагональное элементное представление [15]. В этом случае оказывается достаточным иметь лишь класс векторного итератора. При сканировании матрицы в направлении, совпадающем со способом векторной организации, он обеспечивает требуемую эффективность. При итерировании в другом направлении вряд ли возможно ускорить

операцию обычного индексирования без дополнительных расходов, сопоставимых с организацией самого матричного класса. По этим причинам матричные итераторы в данной работе не рассматриваются.

В заключение перечислим основные базовые методы матричных классов, декларируемые в интерфейсе класса Matrix и реализуемые в рассмотренных векторно-организованных классах. Методы матричных классов осуществляют

— определение и изменение размерности,

— доступ к элементам матрицы, включая использование матричных итераторов,

— доступ к составляющим векторам для векторно-организованных матриц,

— конструирование эквивалентных объектов и дружественных им матричных итераторов,

— проверку параметрических признаков, в частности, положительности, диагонального преобладания, ортогональности, и вычисление определителей, матричных норм, ранга и других матричных мер,

— проверку структурных признаков и определение параметров локализации ненулевых элементов в рамках лент, блоков и т. п.,

— проверку специальных и элементарных признаков,

— арифметические операторы, определенные для скалярных, векторных и матричных аргументов,

— базовые процедуры BLAS 2, в частности, умножение матрицы на вектор, решение треугольных систем, одноранговая модификация,

— базовые процедуры BLAS 3, в частности, умножение матриц, множественное решение треугольных систем, многоранговая модификация.

Укажем некоторые особенности объектной реализации операций BLAS 2, 3. Прежде всего, все методы перегружены для случаев операций над блочными и ленточными сегментами, что необходимо для реализации прямых и итеративных методов линейной алгебры. Другим важным отличием является реализация большинства процедур BLAS в операторной форме с использованием элементарных матриц, рассмотренных выше при построении единой матричной классификации. Использование матричных операций с элементарными и специальными матрицами позволяет улучшить наглядность и понимание разрабатываемых программ и приблизить их к математической записи. Например, переупорядочивание строк и столбцов матрицы, используемое в процедурах выбора главного элемента, естественнее реализуется путем умножения слева и справа на матрицу перестановок, чем при непосредственной перестановке элементов, затрудняющей ее окончательное восстановление.

На уровне векторно-организованных классов или их шаблонов все операции BLAS 2, 3 могут быть реализованы с использованием методов BLAS 1 соответствующего векторного класса, либо с помощью методов доступа к элементам, что менее эффективно. Естественно, можно попытаться переопределить соответствующие мощные операции и более эффективным образом на основе непосредственного использования матричных структур данных. Однако подобное программирование всегда требует значительных усилий, а приводит к ощутимым результатам лишь при учете особенностей вычислительной архитектуры. По-видимому, наилучший вариант состоит в указанной обобщенной реализации операций BLAS как виртуальных методов матричных классов, допускающих при необходимости переопределение более эффективным образом.

8. Некоторые результаты временного тестирования

Для сравнения эффективности применения индексов, векторных итераторов и непосредственно операций BLAS 1 рассмотрим три следующих варианта реализации операции умножения матрицы **A** типа ColumnVectorBasedMatrix на вектор **X**:

```
// с использованием индекса
for(Index j=0; j<a.MSizeOf(); j++)
  for(Index i=0; i<a.NSizeOf(); i++)
    y[i]+=x[j]*a[j][i];
// с применением векторного итератора
for(VectorIteratorType j(x); j<a.MSizeOf(); j++)
  for(VectorIteratorType i(a[j]); i<a.NSizeOf(); i++)
    y[i]+=x[j]*a[j][i];
```

```
    // процедура a[j] определена как a[j.Index()]
// с применением BLAS 1
for(VectorIteratorType j(x); j<a.MSizeOf(); j++)
  y.Axpy(a[j],x[j]);
```

Результаты временного тестирования операции индексирования для векторно-организованных матричных классов, построенной на векторах типа DenseVector, BandVector, IndexVector, приведены в табл. 1. В качестве тестовой взята плотно заполненная квадратная матрица размерности 200.

Данные результаты получены при использовании компилятора GNU G++ 2.5.8 (опции −O, −ansi) на рабочей станции SunSPARCstation 1+ с тактовой частотой 20 МГц и имеют только сравнительную ценность. Для подсчета времени использовалась стандартная функция ОС UNIX times с разрешающей способностью 0,01 с.

Для сравнения, данная операция была запрограммирована максимально эффективным образом для данных, представленных обычными линейными массивами без использования объектного подхода. Время выполнения такой операции составило 0,06 с.

Из данных результатов можно сделать следующие выводы. Прежде всего использование базовых векторных операций позволяет, несмотря на применение объектного подхода, достигать столь же высокой эффективности, что и при обычном программировании. Вместе с тем, появляются важные преимущества, обусловленные наглядностью и простотой разработки матричного обеспечения.

Таблица 1

Результаты временного тестирования операции индексирования для векторно-организованных матричных классов, построенной на векторах разного типа (в секундах)

Тип итерирования	Dense Matrix	Band Matrix	Index Matrix
Индекс	0,16	0,19	3,37
Итератор	0,13	0,14	0,32
BLAS	0,06	0,07	0,31

Как мы видим, использование обычного индекса наименее эффективно для всех типов матриц, поскольку обрабатываются все элементы матрицы, в том числе и нулевые. Различия особенно ощутимы для матриц, построенных на индексных векторах, что объясняется необходимостью многократного сканирования каждого вектора при поиске очередного элемента.

Следующим по эффективности способом реализации является использование векторного итератора. Его применение в случае разреженных матриц позволяет достичь практически эффективности операций BLAS. При этом исключаются адресные операции и обращение к необходимым элементам происходит непосредственно по ссылке.

Наиболее эффективным, естественно, является использование операций BLAS. В приведенном примере вызывается функция Axpy, работающая непосредственно с внутренней структурой конкретных векторных классов. Заметим, что достаточно трудоемкой оказывается реализация всех процедур пакета BLAS 1, оперирующих с разнотипными векторными операндами. Поэтому при отсутствии необходимой конкретной версии процедуры используется обобщенный вариант, реализованный как метод абстрактного векторного класса с привлечением описанной выше псевдоиндексной технологии.

Таким образом, на основе разработанной единой матричной классификации исследованы возможности унифицированной разработки объектно-ориентированного матричного ПО. Анализ различных форм матричной разреженности обнаружил возможность представления произвольных матриц в виде массивов разреженных векторов, принадлежащих четырем базовым классам. Применение псевдоиндексных матричных и векторных итераторов позволило унифицировать разработку достаточно эффективных

разреженных матричных классов, что и подтверждают результаты их практической реализации на языке Си++. Данное обстоятельство обуславливает внедрение данной программной технологии в практику разработок матричного ПО.

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

ЛИТЕРАТУРА

1. Demmel J. LAPACK: A portable linear algebra library for supercomputers, Proceeddings of 1989 IEEE CACSO, Tampa, FL.
2. Dongarra J.J., Bunch J.R., Moler C.B., Stewart G.W. LINPACK Users' Guide, SIAM Publications, Philadelphia, 1979.
3. Garbon B.S., Boyle J.J., Dongarra J.J., Moler C.B. Matrix Eigensystem Routines — EISPACK Guide Extension, Lecture Notes in Computer Science, Vol. 51, Springer-Verlag, Berlin, 1977.
4. Kincaid D.R., Respess J.R., Young D.M. ITPACK 2C: A Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods, ACM Trans. Math. Software, 8 (1982), pp. 302–322.
5. Rice J.R., Boisvert R.F., Solving Elliptic Problems Using ELLPACK, Springer-Verlag, New York, 1985.
6. George A., Liu J.W.H. Computer Solution of Large Sparce Positive Definite Systems. Prentice-Hall: Englewood Cliffs, NJ. [*Русский перевод:* Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. — М.: Мир, 1983.]
7. Arushanian O.B. et al The TOEPLITZ Package Users' Guide, Argonne National Laboratory, ANL-83-16, (1983).
8. PC/MATLAB, Mathworks, 24 Prime Park Way, Natick, MA 01760.
9. Gropp W.D., Foulser D.E., Chang S., CLAM User's Guide: The Computational Linear Algebra Machine, Scientific Computing Associates, Inc., New Haven, Connecticut, USA, 1989.
10. Gallivan K., Jalby W., Meier U. The use of BLAS 3 in linear algebra on a parallel processor with a hierarchical memory, SIAM J.Sci.Statist.Comput. — 1987. — V.8. — P. 1079–1084.
11. Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications, ACM, 1994.
12. Гантмахер Ф.Р. Теория матриц. — М.: Наука, 1988.
13. Хорн Р., Джонсон Ч. Матричный анализ: Пер. с англ. — М.: Мир, 1989.
14. Писсанецки С. Технология разреженных матриц: Пер. с англ. — М.: Мир, 1988.
15. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем: Пер. с англ. — М.: Мир, 1991.