ISSN 0132-3474

Номер 1 Январь - Февраль 2023

ПРОГРАММИРОВАНИЕ

www.sciencejournals.ru

СОДЕРЖАНИЕ

Номер 1, 2023

АНАЛИЗ ДАННЫХ

КОМПЬЮТЕРНАЯ АЛГЕБРА

CONTENTS

No. 1, 2023

DATA ANALYSIS

COMPUTER ALGEBRA

ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 3–31

АНАЛИЗ ДАННЫХ

УДК 004.6

АНАЛИТИКА В РЕАЛЬНОМ ВРЕМЕНИ: ПРЕИМУЩЕСТВА, ОГРАНИЧЕНИЯ И КОМПРОМИССЫ

© 2023 г. С. Д. Кузнецов[a,b,c,d,*] (ORCID: 0000-0002-8257-028X), П. Е. Велихов[e,] (ORCID: 0000-0002-0644-8047), Ц. Фу[f,***] (ORCID: 0000-0003-0244-1718)**

[a]*Институт системного программирования им. В.П. Иванникова РАН, 109004 Москва, ул. А. Солженицына, д. 25, Россия*

[b]*Московский государственный университет имени М.В. Ломоносова, 119991 Москва, Ленинские горы, д. 1, Россия*

[c]*Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

[d]*НИУ "Высшая школа экономики", 101978 Москва, ул. Мясницкая, д. 20, Россия*

[e]*TigerGraph, 94065 Калифорния, Редвуд-Сити, Твин Дельфин Драйв, 3, США*

[f]*Техкомпания Хуавэй. 121614 Москва, ул. Крылатская, д. 17, к. 2, Россия*

**E-mail: kuzloc@ispras.ru*

***E-mail: pavel.velikhov@tigergraph.com*

****E-mail: fqiang.fuqiang@huawei.com*

Поступила в редакцию 10.09.2022 г.
После доработки 19.09.2022 г.
Принята к публикации 24.09.2022 г.

Аналитика в реальном времени – относительно новая ветвь аналитики. Обычное "определение" аналитики в реальном времени заключается в том, чтобы как можно быстрее анализировать данные по самым последним данным. Это определяет суть фундаментальных потребностей пользователей, но никоим образом не является конкретным требованием к соответствующим программным комплексам в силу нечеткости "определения". В результате разные производители систем управления аналитическими данными и исследователи относят к системам аналитики в реальном времени совершенно разные системы, отличающиеся архитектурой, функциональностью и даже временными параметрами. Цель этой статьи – проанализировать различные подходы к предоставлению аналитики в реальном времени, их преимущества и недостатки, а также компромиссы, на которые неизбежно приходится идти как разработчикам систем, так и их пользователям.

DOI: 10.31857/S0132347423010053, **EDN:** GRYYWT

1. ВВЕДЕНИЕ

Традиционная технология анализа данных (On-Line Analytical Processing, OLAP) в основном базируется на концепции хранилищ данных (Data Warehouse), как она описана, например, в книгах пионеров этого подхода Билла Инмона [1] и Ральфа Кимбалла [2]. Согласно их определениям, хранилище данных – это база данных, в которой тесно интегрируются на физическом уровне данные из разных источников с помощью специальной процедуры извлечения, преобразования, загрузки (Extract, Transform, Load, ETL). Хранилище данных содержит как можно больше исторических данных, чтобы предоставить аналитикам возможность обнаружить с помощью статистики скрытые правила или тенденции, которые могут помочь ответственным лицам принимать правильные *стратегические* бизнес-решения.

21-й век принес новую концепцию аналитики – *анализ данных в реальном времени*. Gartner [3] дает следующее определение: "Аналитика в реальном времени – это дисциплина, которая применяет логику и математику к данным, чтобы предоставить информацию для быстрого принятия лучших решений. В некоторых случаях реальное время просто означает, что аналитика завершается в течение нескольких секунд или минут после поступления новых данных". Как видно, Gartner больше всего выделяет одну важную особенность: анализ данных в реальном времени должен выполняться как можно быстрее. Однако не менее важна и другая особенность аналитики в реаль-

ном времени: данные должны быть максимально свежими.

Существует множество различных вариантов использования аналитики в реальном времени, но все они относятся к *повседневной* деловой деятельности: такой анализ данных помогает оптимизировать рутинные бизнес-операции. В этом случае решения должны приниматься *быстро*, поэтому аналитические операции должны быть *быстрыми,* а данные должны отражать текущую ситуацию в бизнесе, поэтому они должны быть *свежими*. Понятия быстрой аналитической обработки данных и свежести данных не абсолютны, а скорее относительны: программная система, обеспечивающая аналитику в реальном времени, делает все возможное для поддержки быстрых сколь угодно сложных аналитических запросов к свежим данным, однако пользователям всегда придется искать компромисс между их потребностями, преимуществами аналитики в реальном времени и реальные возможностями системы.

Продолжим приведенную выше цитату из Глоссария Gartner: "*Аналитическая система по требованию (on-demand) в реальном времени* ожидает, пока пользователи или системы предоставят запрос, а затем выдают аналитические результаты. *Непрерывная (continuous) аналитика в режиме реального* времени более активна и предупреждает пользователей или инициирует реакцию по мере возникновения событий". В частности, в настоящее время существует два основных подхода к аналитике в реальном времени. Первый подход обеспечивает практически в реальном времени ответы на произвольно сложные аналитические запросы к почти свежим сохраненным данным. Второй предлагает ограниченные аналитические операции в настоящем режиме реального времени над свежими потоковыми данными. По аналогии с общепринятым понятием системы реального времени первый подход можно назвать аналитикой *мягкого (soft)* реального времени, а второй – аналитикой *жесткого (hard)* реального времени.

Первая категория решений, в свою очередь, может быть разделена на два подкласса: так называемые современные хранилища данных (*modern data warehouses*) и системы гибридной транзакционно-аналитической обработки данных (*hybrid transactional/analytical data processing, HTAP*). Системы первого подкласса обычно ориентированы на облачную среду (хотя некоторые поставщики также предоставляют версии своих систем для использования в локальной среде (*on premise*)). Они обеспечивают быструю аналитическую обработку за счет использования массивно-параллельной архитектуры (massively-parallel processing, MPP), хранененения таблиц по столбцам и современных методов оптимизации запросов. Свежесть данных обеспечивается за счет возможности динамического добавления внешних данных, часто в потоковом режиме.

Решения HTAP поддерживают в рамках одной системы как транзакционную, так и аналитическую обработку данных. Оперативные данные, генерируемые транзакциями, становятся почти немедленно доступными для анализа данных. Это общий принцип обеспечения свежести данных в системах HTAP. Быстрая аналитическая обработка запросов обеспечивается за счет хранения данных в основной памяти и распараллеливания запросов на современных многоядерных процессорах.

В мировой литературе имеется множество публикаций, посвященных аналитике в реальном времени, включая, в частности, обзоры (например, [4–7]). Однако, насколько нам известно, в каждой из этих работ рассматриваются технологии и решения, относящиеся к одному конкретному подходу (например, потоковой обработке [4, 5] или HTAP [6, 7]), в предположении, что этот и только этот подход предоставляет аналитику в режиме реального времени.

Напротив, цель нашей статьи состоит в том, чтобы использовать обобщенное понятие анализа данных в (почти) реальном времени, чтобы рассмотреть и представить примеры наиболее успешных реализаций всех соответствующих современных подходов и технологий, которые могут быть использованы в этой области аналитики. В каждом случае мы также предоставим наиболее известные и очевидные варианты использования соответствующего подхода.

Следует отметить, что наш обзор не является исчерпывающим из-за обширности темы и ограниченного размера статьи. Мы также ограничиваем себя, рассматривая только аналитику над обычными структурированными данными реляционной модели данных (точнее, ее воплощения на основе SQL). Важные вопросы аналитики полуструктурированных или неструктурированных данных в режиме реального времени выходят за рамки этой статьи.

Оставшаяся часть статьи имеет следующую структуру. В разд. 2 мы приводим краткую историю потоковой обработки данных (главным образом, анализа), описываем наиболее важные компоненты этой технологии и представляем наиболее известные решения как коммерческих вендоров, так и сообщества открытого кода. В разд. 3 описываются основные функции и архитектурные принципы современных хранилищ данных с упором на облачные решения крупных поставщиков. Мы также анализируем, какие основные изменения в традиционных технологиях хранилищ данных обеспечивают свежесть данных и быструю обработку запросов. Разд. 4 посвящен анализу данных почти в реальном времени

на основе технологии HTAP. Мы рассмотрим корни этой технологии (в основном управление транзакциями в системах, хранящих базы данных в основной памяти, и аналитика с использованием хранения таблиц по столбцам), опишем историю подхода и представим наиболее интересные коммерческие и академические решения с учетом возможностей новой энергонезависимой памяти. Разд. 5 подводит итоги и завершает статью.

2. ПОТОКОВАЯ АНАЛИТИКА ДАННЫХ

Похоже, что понятие потока данных впервые было введено в статье Моники Хензингер (Monika Henzinger), Прабхакара Рагхавана (Prabhakar Raghavan) и Шридара Раджагопалана (Sridar Rajagopalan) [8]. Они определили поток данных как такую последовательность элементов данных $x_1 \dots x_i \dots x_n$, что элементы считываются один раз в порядке возрастания индексов i. Авторы выбрали два параметра – количество проходов по потоку (P) и размер требуемой оперативной памяти (S) (в зависимости от n) – и изучили алгоритмы решения различных задач с малыми значениями P и значениями S, меньшими размера входных данных.

2.1. Первые годы потоковой обработки

Вышеупомянутая статья была скорее теоретической и не предлагала никаких идей анализа потоковых данных, но концепция потоковой передачи данных была настолько жизненно важной, соответствовала стольким важным вариантам использования (например, сенсорным сетям, биржевым рынкам, управлению трафиком и т.д.), что с начала нового века многие исследователи начали работать в этой новой области управления данными – области потоковой обработки данных. Первые результаты этих исследований обсуждались на нескольких встречах в Стэнфордском университете в 2002–2003 гг. [9] и были опубликованы в [10]. На наш взгляд, наиболее важными для будущих исследований и разработок были статьи об Aurora и Medusa [11], TelegraphCQ [12] и Stanford Stream Data Manager [13]. Мы не будем подробно останавливаться на деталях предлагавшихся архитектур и кратко упомянем лишь наиболее важные результаты этих работ.

Мы считаем, что наиболее важной особенностью потоковой аналитики является понятие *непрерывного запроса* (*continuous query*). Эта концепция была введена в 1992 году Дугласом Терри (Douglas Terry) и др. [14], гораздо раньше, чем концепция потоковой передачи данных, и в совершенно другом контексте. Как говорят авторы, “непрерывные запросы аналогичны обычным запросам к базе данных за исключением того, что они выдаются один раз и в дальнейшем выполняются “непрерывно” над базой данных”. Очевидно, что в этом контексте такие запросы имеют смысл тогда и только тогда, когда состояние базы данных изменяется с течением времени, чтобы предоставить релевантные результаты такого запроса в разные моменты времени. Чтобы пользователи могли писать непрерывные запросы, авторы использовали в своей системе Tapestry специальный язык запросов TQL, но во время предварительной обработки запросов система преобразовывала запросы TQL в стандартный SQL.

Другой контекст использования непрерывных запросов был продемонстрирован в проекте NiagaraCQ [15]. Под руководством Дэвида Девитта (David Dewitt) группа из Университета Висконсин-Мэдисон разработала распределенную систему, поддерживающую непрерывные запросы к распределенному набору данных XML. Для формулировки запросов они использовали расширение языка запросов XML-QL, одного из предшественников XQuery. Основной целью этого проекта было обеспечение масштабируемости, достаточной для одновременной поддержки миллионов непрерывных запросов. Для достижения этой цели разработчики пытались группировать похожие запросы и обрабатывать не отдельные запросы, а группы запросов.

Идея непрерывных запросов оказалась очень актуальной в контексте потоковых данных, и эта идея интенсивно использовалась во всех трех проектах, упомянутых в начале этого подраздела и стартовавших почти одновременно в начале 2000-х годов.

2.1.1. TelegraphCQ. Проект TelegraphCQ [12] (более подробное описание можно найти в [16]) был выполнен командой Калифорнийского университета в Беркли, в которую входили Майкл Франклин (Michael Franklin), Джозеф Хеллерштейн (Joseph Hellerstein) и Сэмюэл Мэдден (Samuel Madden). Проект включал в себя несколько этапов прототипирования и редизайна, а TelegraphCQ – это название окончательной архитектуры, изображенной на рис. 1.

Проект родился в Беркли и, естественно, базировался на СУБД PostgreSQL с открытым исходным кодом. Однако для поддержки запросов потоковых данных и достижения других целей проекта команда существенно изменила архитектуру PostgreSQL. Главной из других целей была адаптивность системы. Это мы здесь обсуждать не будем.

Для достижения первой, самой важной цели они в первую очередь адаптируют и расширяют идею *окон* в потоке данных, впервые представленную Йоханнесом Герке (Johannes Gehrke) и др. в [17]. Авторы этой статьи предложили два вида окон – раздвижные (landmark) и скользящие (slading). В раздвижном окне фиксируется старый

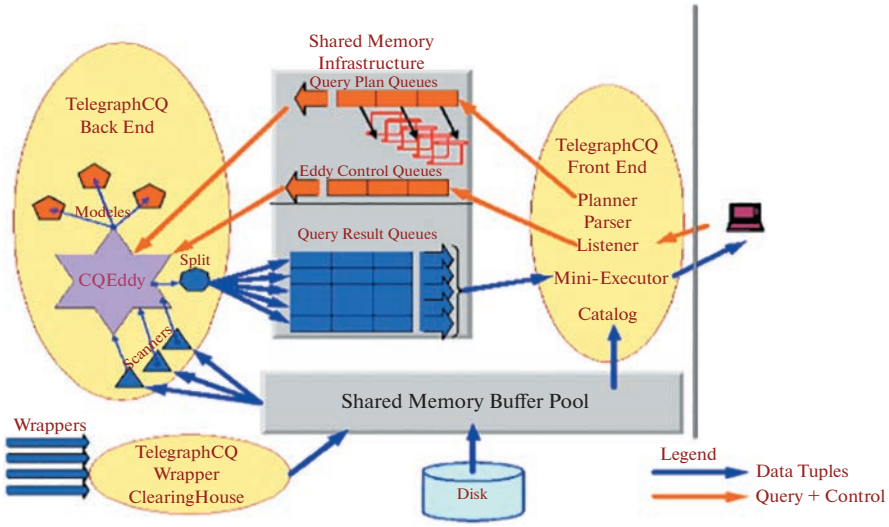

Рис. 1. Архитектура TelegraphCQ.

(левый) конец, а новый (правый) конец перемещается вперед при поступлении в потоке новых кортежей, в то время как в скользящем окне при прибытии новых кортежей оба конца окна перемещаются вперед. В TelegraphCQ использовались гораздо более общие окна, чем раздвижные и скользящее. Была введена специальная конструкция *for-loop* для объявления последовательности окон, в которых пользователь хочет получить ответы на свой запрос: переменная *t* перемещается по временной шкале по мере выполнения цикла, а левый и правый концы каждого окна в последовательности и условие остановки выполнения запроса могут быть определены относительно этой переменной *t*.

Во-вторых, TelegraphCQ обеспечивает возможность одновременного выполнения нескольких запросов, как непрерывных, так и выполняемых однократно, как над потоковыми, так и над историческими (сохраняемыми в базе данных) данными. Для поддержки такой возможности был разработан специальный многопоточный исполнитель запросов, который активно использовал разделяемую память, работал без блокировок и пытался совместно обрабатывать похожие запросы.

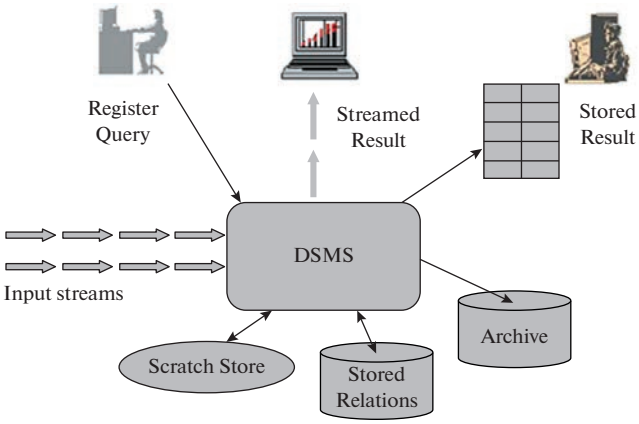

Рис. 2. Высокоуровневое представление STREAM.

Наконец, хотя авторы [12, 16] не акцентируют на этом внимание, но похоже, что они использовали в качестве языка запросов расширенную версию диалекта SQL PostgreSQL.

2.1.2. Stanford Stream Data Manager. Stanford Stream Data Manager (Stream, позже переименованный в Stanford Data Stream Management System) [13] (более подробное описание см. в [18]) естественным образом разрабатывался в Стэнфордском университете при участии (может быть, под руководством) всем известной Дженнифер Видом (Jennifer Widom). Высокоуровневое представление STREAM показано на рис. 2.

Входящие входные потоки бесконечно производят данные и управляют обработкой запросов. Для обработки непрерывных запросов обычно требуется сохранять в основной или внешней памяти промежуточное состояние данных (Scratch Store). Хотя основной задачей Stream являлась оперативная обработка непрерывных запросов, во многих приложениях потоковые данные можно было скопировать в архив для сохранения и возможной автономной обработки аналитических запросов. Пользователи или приложения регистрируют непрерывные запросы, которые остаются активными в системе до тех пор, пока их регистрация явно не будет отменена. Результаты непрерывных запросов обычно передаются в виде выходных потоков данных.

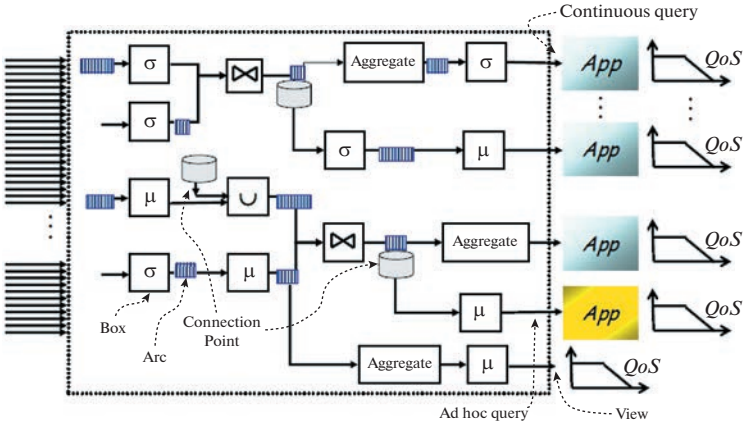

Рис. 3. Сеть обработки в Aurora.

Stream был действительно работающим прототипом системы потоковой обработки, но наиболее важным результатом проекта стал язык CQL (*Continuous Query Language*). Хотя синтаксически CQL является незначительным расширением SQL, его семантика сильно отличается от семантики SQL. В своих статьях о Stream авторы описывают абстрактную семантику SQL, приводят примеры запросов и обсуждают особенности их оптимизации и обработки. Сама команда Stream считала дизайн CQL настолько важным, что представила отдельную длинную статью [19], полностью посвященную возможностям этого языка. Между прочим, эта статья явно демонстрирует, что Дженнифер Видом была основным разработчиком CQL. Язык жив и поддерживается почти во всех современных системах обработки потоков данных.

2.1.3. Aurora, Borealis, StreamBase. Проект Aurora [11] (более подробное описание см. в [20, 21]) был инициирован Майклом Стоунбрейкером, который привлек к участию в проекте профессоров и студентов из MIT, университетов Brown и Brandeis. Отметим, что, насколько нам известно, [11] была первой публикацией, в которой высказывалось предположение, что потоковая обработка данных может служить основой для аналитики в реальном времени. Авторы пишут: “Распространенной целью является разработка технологии “предприятия реального времени”, с помощью которой можно было бы производить бизнес-анализ данных в режиме реального времени. Такой тактический анализ требует перехвата потоков данных из оперативных систем, их объединения и последующей обработки в режиме реального времени. Поддержка предприятий реального времени – одна из целей специализированных систем потоковой обработки, таких как Aurora и Medusa”.

В Aurora непрерывные запросы определялись с помощью визуального графического интерфейса с прямоугольниками и дугами, а систему Aurora можно рассматривать как направленный подграф диаграммы потока работ, который выражает все одновременно выполняемые вычисления результатов запросов. Эта диаграмма потока работ называется сетью Aurora (см. рис. 3). Запросы строятся из стандартного набора четко определенных операторов (блоков). Дуги обозначают очереди кортежей, представляющие потоки. Каждый блок по расписанию обрабатывает один или несколько кортежей из своей входной очереди и помещает результаты в свою выходную очередь. Когда кортежи поступают в очереди, подключенные к приложениям, они оцениваются в соответствии с QoS (quality of service) приложения.

По умолчанию запросы являются непрерывными в том смысле, что они потенциально могут выполняться бесконечно долго для входных данных. Во время работы системы также могут быть определены однократно выполняемые запросы, привязываемые к точкам подключения, которые представляют собой предопределенные дуги в сети, ведущие к хранилищам исторических данных. С точками подключения могут быть связаны спецификации персистентности, которые указывают, как долго должна храниться история. Aurora также позволяет определять обособленные точки подключения, в которых могут храниться статические наборы данных. Наличие точек подключения позволяет направлять запросы к сочетанию традиционных хранимых данных и динамических данных, поступающим в потоках. Aurora также позволяет определять представления, то есть запросы, к которым не подключено ни одно приложение. Представлению разрешается иметь спецификацию QoS для указания на уровень его важности. При наличии потребности приложения могут подключаться к представлениям.

Medusa как отдельный проект упоминается только в [11]. Вкратце, этот проект обеспечил се-

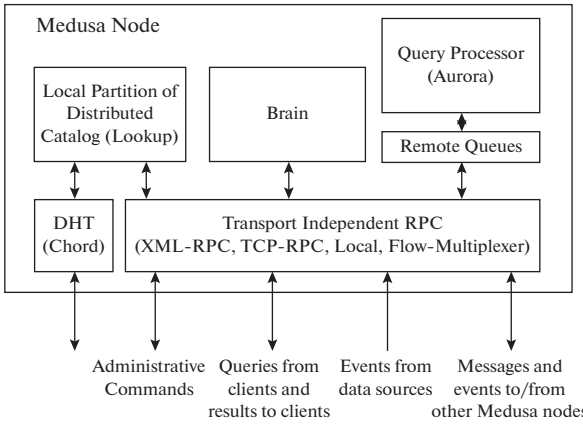

Рис. 4. Высокоуровневая архитектура узла Medusa.

тевую инфраструктуру, которая позволила организовать распределенную обработку потоков на базе Aurora (см. рис. 4).

В конце 2003 года система Aurora была коммерциализирована. Майк Стоунбрейкер и его коллеги основали стартап, который первоначально назывался Grassy Brook, а вскоре был переименован в StreamBase [21]. В то же время межуниверситетская команда в течение следующих трех лет продолжала разработку системы распределенной обработки потоков под названием Borealis [21, 22]. Как показывает рис. 5, архитектура этой системы была фактически объединением Aurora и Medusa.

Компания StreamBase работала самостоятельно десять лет в тесном сотрудничестве с университетами, а затем в 2013 г. была приобретена компанией TIPCO Software, где программное обеспечение StreamBase в настоящее время является основным продуктом для потоковой обработки данных [23]. Вероятно, наиболее интересным вкладом StreamBase в сообщество обработки потоков является язык запросов StreamSQL – диалект SQL, отражающий семантику запросов Aurora [24].

Интересно, что еще в 2008 году Стоугбрейкер инициировал процесс интеграции CQL и StreamSQL с целью определить единый язык потоковых запросов. Эта попытка, результаты которой были опубликованы в [25], не увенчалась успехом, поскольку семантика CQL и StreamSQL слишком различается. Однако обе модели полезны в разных приложениях, поэтому многие современные системы обработки потоков поддерживают оба языка.

Наконец, в результате исследований Майкла Стоунбрейкера и его коллег была опубликована очень разумная, ценная и влиятельная статья [26]. Рекомендации Стоунбрейкера, Четинтемеля (Uğur Çetintemel) и Здоника (Stanley Zdonik) сыграли и продолжают играть важную роль в разработке новых систем потоковой обработки. Мы кратко перечислим здесь эти требования.

1) Система обработки потоковых данных в реальном времени должна обрабатывать сообщения “в потоке”, без необходимости их сохранения для выполнения какой-либо операции или последовательности операций.

2) Система должна поддерживать высокоуровневый язык “StreamSQL” со встроенными расширяемыми примитивами и операторами, ориентированными на потоки.

3) Система должна иметь встроенные механизмы для обеспечения устойчивости к “несовершенствам” потока, включая отсутствующие и неупорядоченные данные.

4) Система должна гарантировать предсказуемые и воспроизводимые результаты.

5) Система должна иметь возможность эффективно хранить, получать доступ и изменять информацию о состоянии, а также объединять ее с потоковыми данными в реальном времени.

6) Чтобы избежать сбоев в обработке данных в реальном времени, система потоковой обработки

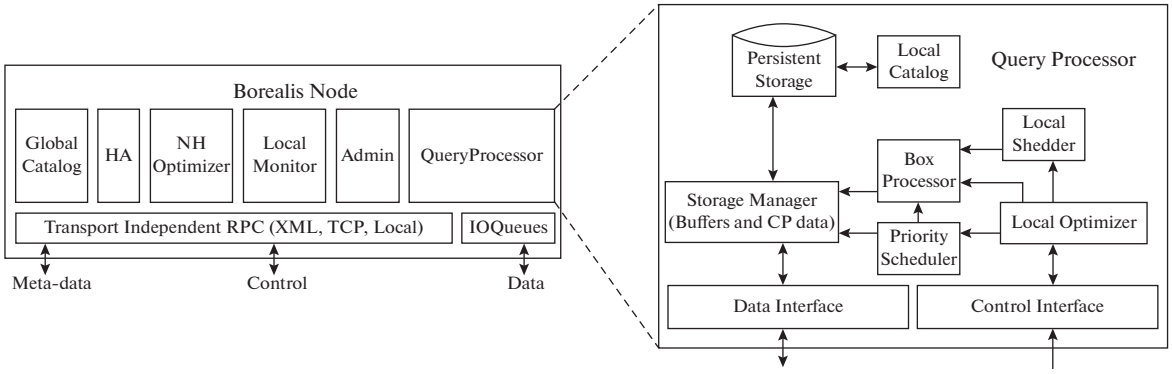

Рис. 5. Архитектура Borealis.

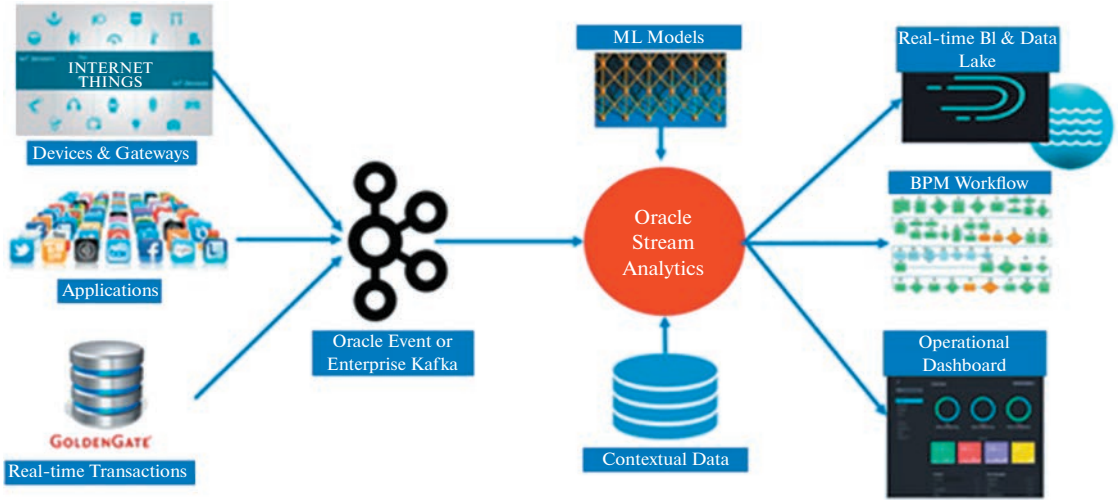

Рис. 6. Архитектура Stream Analytics.

должна использовать решение высокой доступности.

7) Система должна иметь возможность распределять обработку между несколькими процессорами и машинами для достижения дополнительной масштабируемости.

8) Система потоковой обработки должна иметь высокооптимизированный исполнительный механизм с минимальными накладными расходами, чтобы обеспечить отклик в реальном времени для приложений, работающих с большим объемом данных.

2.2. Следующее поколение и современное состояние

В то время как в первое десятилетие 2000-х годов в основном выполнялись фундаментальные исследования принципов потоковой обработки, второе десятилетие включало разработку практически полезных систем, как проприетарных, так и открытых. Общий термин *система управления потоками данных* (*Data Stream Management System, DSMS*), введенный в первое десятилетие, был принят для обозначения любой программной системы, обрабатывающей потоки данных. В то же время появились дополнительные термины *обработка событий* (*Event Processing*), *обработка сложных событий* (*Complex Event Processing, CEP*) или *обработка потока событий* (*Event Stream Processing*) для обозначения систем, специализирующихся на обработке и анализе событий для выявления событий более высокого уровня. DSMS имеют более широкую область применения, и приложения CEP могут быть реализованы с помощью DSMS [27].

Через десять лет после публикации [10] был опубликован специальный выпуск бюллетеня IEEE Data Engineering Bulletin, в котором содержались статьи, описывающие состояние дел в начале 2010-х годов. С наших позиций и в соответствии с современным состоянием технологии основной интерес представляют публикации [29–31]. Мы будем ссылаться на эти публикации в следующих подразделах.

В настоящее время рынок программного обеспечения предлагает множество решений категории DSMS, и здесь мы должны ограничиться лишь несколькими примерами. Эти примеры включают как продукты крупных поставщиков программного обеспечения, так и решения с открытым исходным кодом.

2.2.1. Oracle Stream Analytics. Компания Oracle начала заниматься обработкой потоковых данных много лет назад (см., например, публикацию [32], авторы которой описывают попытку включить обработку потоковых данных и обработку сложных событий в СУБД, первую такую попытку в коммерческих базах данных). В настоящее время основным потоковым продуктом Oracle является Oracle Stream Analytics [33]. Его архитектура изображена на рис. 6.

Oracle Stream Analytics (OSA) – это технология с хранением данных в основной памяти (*in memory*) для аналитических вычислений в реальном времени над потоковыми большими данными. OSA может автоматически обрабатывать и анализировать крупномасштабную информацию в режиме реального времени, используя сложные паттерны корреляции, обогащения данных и алгоритмы машинного обучения. Потоковые большие данные могут исходить от датчиков IoT, вебконвейеров, файлов журналов, устройств торговых точек, банкоматов, социальных сетей, транзакционных баз данных, баз данных NoSQL или любого другого источника данных. OSA работает в масштабируемой и высокодоступной кластер-

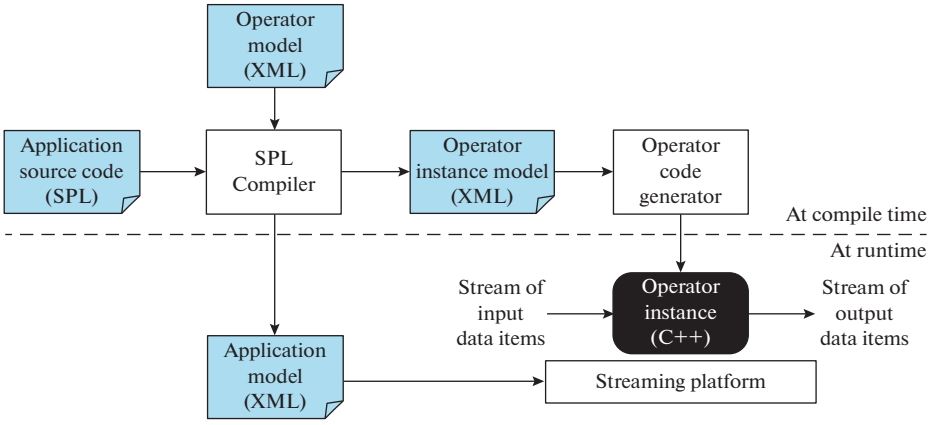

Рис. 7. Компиляция и выполнение приложения, созданного с использованием Streams Processing Language (SPL).

ной среде больших данных с использованием Apache Kafka и Apache Spark Streaming (см. последующие подразделы), интегрированных с Oracle Continuous Query Engine, и обеспечивает решение критических задач в реальном времени на современных предприятиях.

Работа Stream Analytics начинается с приема данных из Kafka с качественной поддержкой сбора данных об изменениях на основе Oracle GoldenGate. Исследование и анализ потока выполняется путем создания конвейеров данных. Конвейер данных может запрашивать данные, используя временны́е окна (с использованием CQL), искать шаблоны и применять условную логику, пока данные все еще находятся в движении. Запросы и конвейер Spark Streaming автоматически генерируются Web-ориентированным инструментальным средством. После анализа данных и обнаружения искомой ситуации конвейер может быть остановлен, чтобы запустить потоки работ BPM (Business Process Modelling) в Oracle Integration или сохранить результаты в Data Lake для обеспечения более глубокого понимания и анализа данных с помощью Oracle Analytics Cloud.

2.2.2. IBM Streams. Система IBM Streams [31, 35] (ранее называвшаяся IBM InfoSphere Streams [36, 37]) – это платформа, которая предоставляет среду программирования (на основе языка SPL – Streams Processing Language) и поддержку времени выполнения для разработки и запуска приложений реального времени над потоковыми данными разных видов (рис. 7).

Сердцем системы, очевидно, является язык обработки потоков. Центральными понятиями SPL являются потоки и операторы. Приложение SPL соответствует потоковому графу. Операторы без входных потоков являются источниками и имеют собственный поток (thread) управления. Большинство других операторов срабатывают только тогда, когда в одном из их входных потоков имеется хотя бы один элемент данных. Точнее говоря, когда в приложении на SPL срабатывает оператор, он выполняет часть кода для обработки элемента входных данных. При активации оператора потребляется один элемент входных данных и может передаваться любое количество элементов данных в выходные потоки. Поток доставляет элементы данных от одного оператора к другому в том порядке, в котором они были отправлены.

Встроенные декларативные операторы SPL поддерживают, в частности, традиционные функции управления окнами, фильтрации и агрегации потоков. Пользователи могут определять свои собственные операторы и использовать их так же, как встроенные. Основное преимущество такого подхода заключается в том, что приложения работают поблизости от поступления данных, и эти приложения относительно безопасны из-за использования одного языка со строгой системой типов.

2.2.3. Потоковая аналитика компании Microsoft. StreamInsight [38–41] – это механизм обработки сложных событий, способный обрабатывать сотни тысяч событий в секунду с чрезвычайно низкой задержкой. Он может быть размещен в любом процессе, таком как служба Windows, или встроен непосредственно в приложение. StreamInsight имеет простую модель адаптера для ввода и вывода данных, а при формулировке запросов как к данным в реальном времени, так и к историческим данным используется один и тот же синтаксис LINQ, доступный так же, как и из любого языка, который поддерживается в Microsoft .NET Framework.

Высокоуровневая архитектура StreamInsight (рис. 8) довольно проста: события собираются из множества источников через входные адаптеры. Эти события анализируются и преобразуются с помощью запросов, а результаты запросов рас-

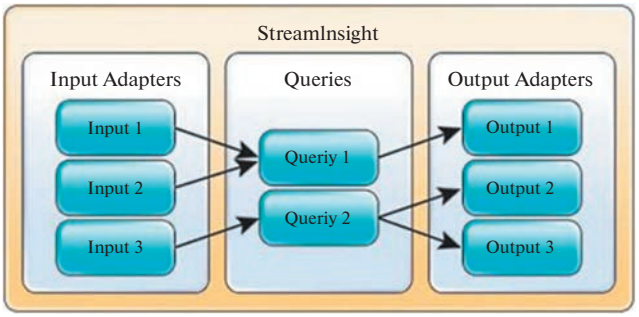

Рис. 8. Высокоуровневая архитектура Microsoft StreamInsight.

пространяются на другие системы и людей через выходные адаптеры.

В StreamInsight в качестве языка запросов используется LINQ, и такие запросы выглядят точно так же, как запросы LINQ to SQL, адресуемые к базе данных. Когда события поступают от адаптера ввода, оценивается их полезность и, если значение свойства Value больше 0.5, они передаются адаптеру вывода.

В настоящее время потоковая аналитика в Microsoft переходит в облачную среду, и наиболее продвигаемым продуктом является Azure Stream Analytics [42]. Azure Stream Analytics (рис. 9) – это механизм аналитики в реальном времени и обработки сложных событий, предназначенный для одновременного анализа и обработки больших объемов быстрых потоковых данных из нескольких источников. Паттерны и взаимосвязи могут обнаруживаться в информации, извлекаемой из ряда источников ввода, включая устройства, датчики, потоки кликов, каналы социальных сетей и приложения. Эти паттерны можно использовать для запуска действий и инициирования потоков работ, таких как создание предупреждений, передача информации в средство формирования отчетов или сохранение преобразованных данных для последующего использования. Кроме того, система Stream Analytics доступна в среде выполнения Azure IoT Edge, что позволяет обрабатывать данные на устройствах IoT.

Задание Azure Stream Analytics состоит из ввода, запроса и вывода. Stream Analytics получает данные из концентраторов событий Azure (Azure Event Hubs), включая концентраторы событий Azure из Apache Kafka, концентратора Интернета вещей Azure (Azure IoT Hub) или хранилища BLOB-объектов Azure (Azure Blob Storage). Запрос, основанный на языке запросов SQL (очевидно, с некоторыми расширениями), можно использовать для простой фильтрации, сортировки, агрегирования и соединения потоковых данных за определенный период времени. SQL может быть дополнительно расширен с помощью пользовательских функций (User-Defined Functions, UDF), написанных на JavaScript и C#. Варианты упорядочения событий и продолжительность временных окон могут быть скорректированы при выполнении операций агрегирования с помощью простых языковых конструкций и/или конфигураций.

Многие другие компании предлагают свои решения для анализа потоковых данных, например, уже упомянутая StreamBase компании TIBCO [23], Data Engineering Streaming компании Informatica [43], Event Stream Processing компании SAS [44], Apama Streaming Analytics компании Software AG и т.д. Мы не будем обсуждать здесь эти продукты и сосредоточимся далее на некоторых чрезвычайно популярных разработках с открытым исходным кодом. Все они принадлежат сообществу Apache.

2.2.4. Apache Kafka and Samza. Эти программные системы традиционно связаны, потому что обе они происходят от LinkedIn, а Samza изначально была построена над Kafka [29]. Авторы [29] называли Kafka брокером сообщений, а Samza – фреймворком для обработки сообщений. Теперь в [45] утверждается, что Kafka является рас-

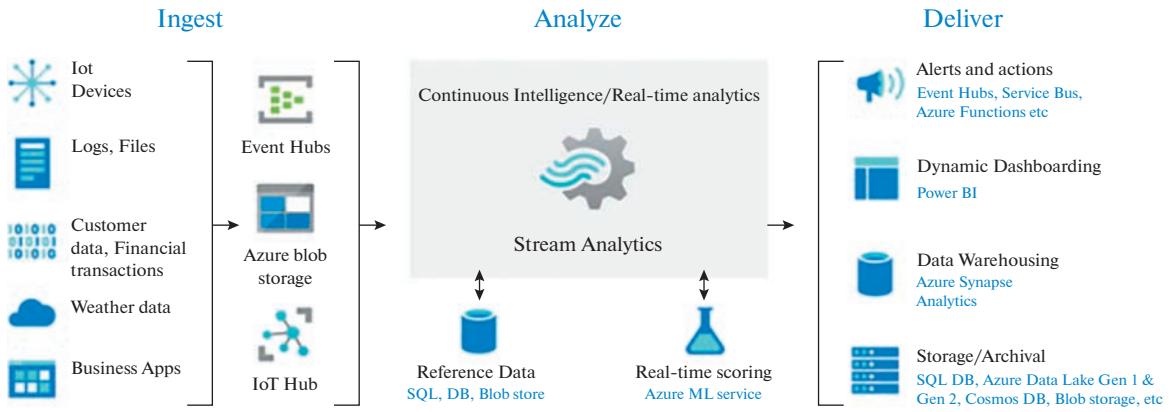

Рис. 9. Конвейер аналитической обработки в Azure Stream Analytics.

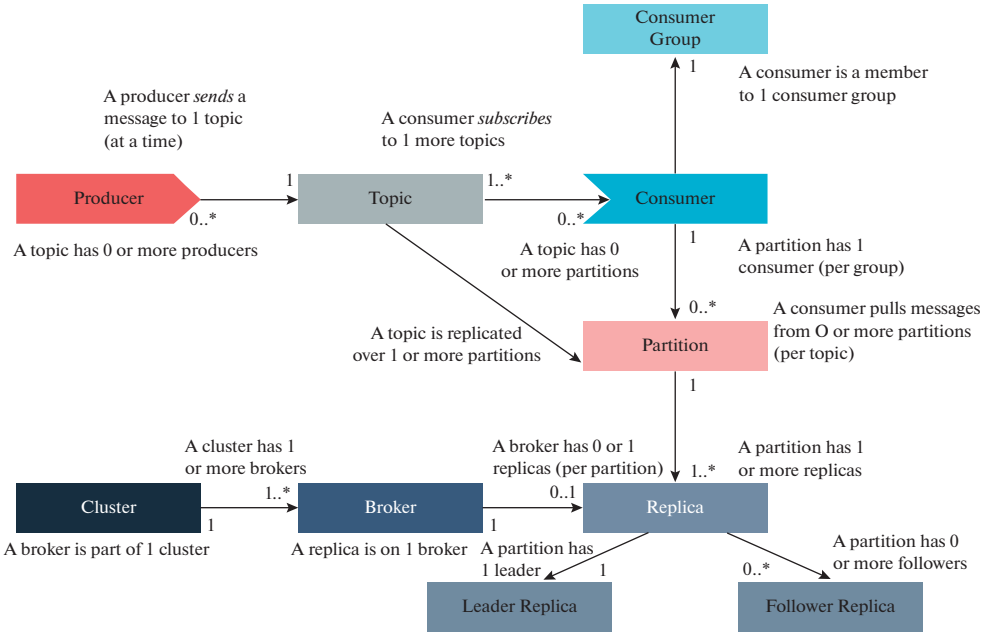

Рис. 10. Основные компоненты Kafka.

пределенной платформой потоковой передачи событий, а [46] называет Samza фреймворком распределенной потоковой обработки.

Так или иначе, Kafka активно используется в различных приложениях и системах для обработки (и анализа) потоков данных. Архитектура и основные компоненты Kafka показаны на рис. 10. Компоненты кратко описываются ниже [47].

• Брокер Kafka – это сервер, работающий в кластере Kafka (кластер Kafka состоит из нескольких брокеров). Как правило, несколько брокеров взаимодействуют, образуя кластер Kafka и добиваясь балансировки нагрузки, надежности на основе избыточности и аварийного переключения. Для управления кластером и координации совместной работы брокеры используют Apache ZooKeeper [48]. Каждый экземпляр брокера способен эффективно обрабатывать сотни тысяч сообщений в секунду суммарного терабайтного объема.

• Производитель (producer) Kafka служит источником данных, который оптимизирует, записывает и публикует сообщения для одной или нескольких тем (topic) Kafka. Производители Kafka также сериализуют, сжимают и балансируют данные между брокерами посредством партиционирования.

• Потребители (consumer) Kafka читают данные, выбирая сообщения из тем, на которые они подписаны. Каждый потребитель принадлежит к некоторой группе потребителей. Каждый потребитель в своей группе потребителей отвечает за чтение части разделов каждой темы, на которую он подписан.

• Тема Kafka определяет канал, по которому передаются данные. Производители публикуют сообщения в темах, а потребители читают сообщения из темы, на которую они подписаны. Темы организуют и структурируют сообщения, причем сообщения заданного типа публикуются в соответствующей теме. Темы в кластере Kafka идентифицируются по уникальным именам, и число создаваемых тем не ограничено.

• В кластере Kafka темы делятся на разделы (partition), и разделы реплицируются между брокерами. Из каждого раздела несколько потребителей могут читать тему параллельно. Также возможно, чтобы производители добавляли к сообщению ключ – все сообщения с одним и тем же ключом отправляются в один и тот же раздел. Сообщения добавляются в разделы последовательно, а сообщения без ключей распределяются по разделам в циклическом режиме.

Архитектура Kafka обеспечивает масштабируемость и высокую производительность, надежность и восстановление после отказов.

Фреймворк Apache Samza (высокоуровневая архитектура изображена на рис. 11) разработан специально для использования преимуществ архитектуры Kafka и гарантирует отказоустойчивость, буферизацию и хранение состояния. Для координации совместного использования ресурсов в Apache Samza используется YARN [49]. Это означает, что по умолчанию требуется кластер

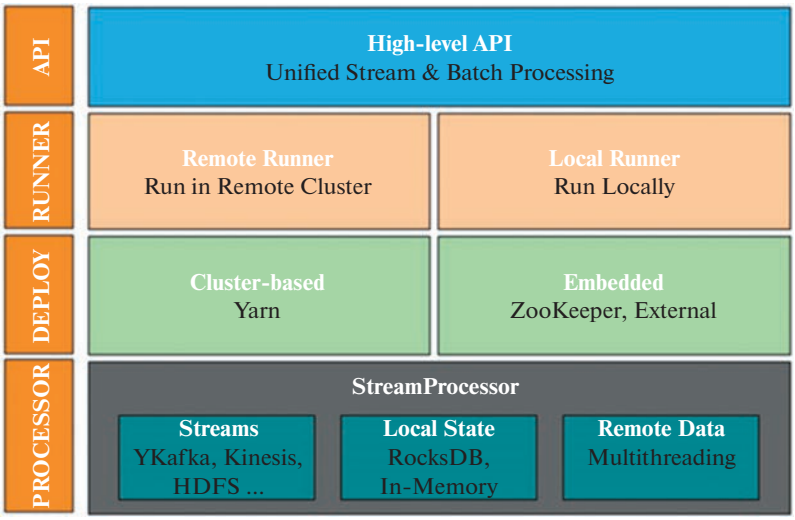

Рис. 11. Высокоуровневая архитектура Apache Samza.

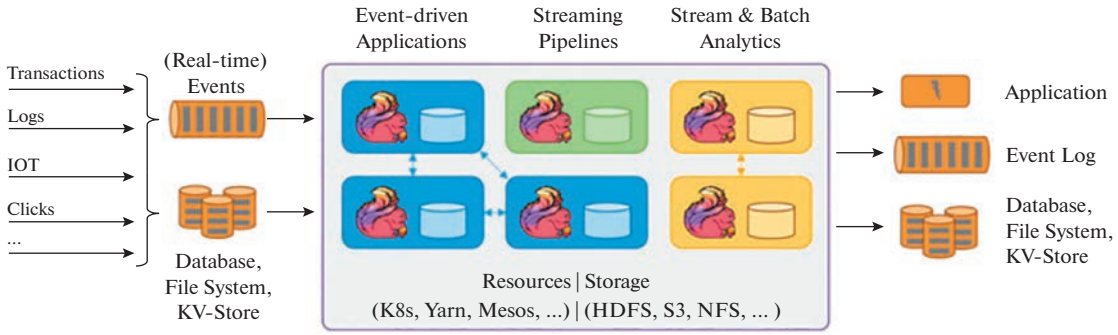

Рис. 12. Высокоуровневое представление Apache Flink.

Hadoop, а Samza полагается на развитые функции, встроенные в YARN.

2.2.5. Apache Flink. Apache Flink [30] – это фреймворк и механизм распределенной обработки для вычислений с отслеживанием состояния над *неограниченными (unbounded)* и *ограниченными (bounded)* потоками данных (см. рис. 12). Flink разработан для использования во всех распространенных кластерных средах, обеспечивая эффективное выполнения вычислений in memory в любом масштабе.

У неограниченных потоков имеется начало, но отсутствует установленный конец. Они не завершаются и предоставляют данные по мере их поступления. Неограниченные потоки должны обрабатываться непрерывно, т.е. события должны обрабатываться сразу же после их приема. У ограниченных потоков имеют установленные начало и конец. Перед выполнением каких-либо вычислений над ограниченным потоком должны быть приняты все его данные. Точный контроль времени и состояния позволяет среде поддержки времени выполнения Flink запускать любые приложения над неограниченными потоками. Ограниченные потоки обрабатываются с применением алгоритмов и структур данных, специально разработанных для наборов данных фиксированного размера, что обеспечивает высокую производительность.

2.2.6. Apache Spark Streaming. Spark Streaming [52] – это расширение базового API Spark [53], которое обеспечивает масштабируемую, высокопроизводительную и отказоустойчивую потоковую обработку потоков данных в реальном времени. Данные могут быть получены из многих источников, таких как сокеты Kafka или TCP (см. рис. 13), и могут быть обработаны с использованием сложных алгоритмов, выраженных с помощью высокоуровневых функций, таких как map, reduce, join и window. Наконец, обработанные данные можно передавать в файловые системы, базы данных и интерактивные информационные панели. К потокам данных можно применять ал-

Рис. 13. Spark Streaming.

горитмы машинного обучения и обработки графов Spark.

Spark Streaming получает потоки входных данных в реальном времени и разделяет данные на пакеты, которые затем обрабатываются механизмом Spark для создания окончательного потока результатов в пакетах. Spark Streaming предоставляет высокоуровневую абстракцию, называемую *дискретизированным потоком*, или *DStream*, которая представляет непрерывный поток данных. DStreams можно создавать либо из входных потоков данных из таких источников, как Kafka и Kinesis, либо путем применения высокоуровневых операций к другим DStreams.

Apache поддерживает ряд других проектов потоковой аналитики, но мы ограничимся четырьмя рассмотренными выше.

2.3. Преимущества, недостатки и компромиссы

Имеются многочисленные варианты использования потоковой аналитики в реальном времени, которые повторяются буквально всеми поставщиками таких решений:

• Обнаружение мошенничества в режиме реального времени на основе алгоритмов машинного обучения обнаружения аномалий.

• Формирование маркетинговых предложений в режиме реального времени на основе местоположения и лояльности клиентов.

• Улучшение обслуживания активов за счет отслеживания нормальных рабочих параметров и упреждающего планирования.

• Повышение доходности за счет постоянного отслеживания спроса и оптимизации уценок вместо случайного снижения цен.

• Корректировка цен путем постоянного отслеживания спроса, уровня запасов и мнений о продуктах в социальных сетях и т.д.

• Повышение уровня продаж продуктов и услуг за счет мгновенного определения присутствия клиента на веб-сайте компании.

• Улучшение использования активов за счет отслеживания среднего времени, необходимого для погрузки и разгрузки товаров.

• Отслеживание операционных событий авиакомпании для исключения задержки рейсов.

• Анализ потоков телеметрии с устройств Интернета Вещей.

• Геопространственная аналитика для управления автопарком и беспилотными транспортными средствами.

• Аналитика в режиме реального времени по данным точек продаж для управления запасами и обнаружения аномалий.

• И так далее.

Во всех этих случаях, когда потоки данных (или событий) генерируются в процессе работы предприятий, эти предприятия получают *реальную выгоду* от потокового анализа.

Потенциальные *недостатки* потоковой аналитики, на наш взгляд, следующие:

• Как мы упоминали во введении, у аналитики в реальном времени имеются два аспекта: она должна обеспечивать *быстрый анализ свежих данных*. Безусловно, любая DSMS *должна* поддерживать настолько быструю обработку запросов, насколько быстры соответствующие потоки данных. В этом смысле эти системы обеспечивают жесткую обработку в реальном времени. Но как насчет свежести данных? DSMS является только потребителем потоковых данных, а не их производителем. По сути, ситуация ничем не отличается от обычной загрузки данных в хранилище (см. разд. 3): непонятно, как проверить, что поступающие данные действительно свежие.

• Пятое требование Стоунбейкера и др. [26] утверждает, что DSMS должна обеспечивать унифицированный доступ на основе интегрированного языка запросов как к потоковым, так и к историческим (хранимым) данным. Это требование вполне разумно с точки зрения аналитика. Однако 5-е требование может привести к значительным проблемам с реализацией. Если объем непрерывного запроса ограничивается только потоковыми данными, то более или менее понятно, как обеспечить его быстрое выполнение: общий объем данных, к которым обращается запрос, ограничен размерами соответствующих окон, и обычно эти данные умещаются в основную память.

Но если запрос обращается как к потоковым, так и к историческим данным, то размер запрашиваемых данных практически неограничен, и единственный способ обеспечить быстрое выполнение таких запросов – ограничить их сложность. Непонятно, как определить такое ограничение, чтобы DSMS оставалась системой жесткого реального времени без чрезмерного снижения возможностей аналитиков.

Необходимые *компромиссы* очевидны: разработчики приложений и аналитики должны четко понимать потенциальные преимущества и ограничения DSMS, тщательно изучать ограничения конкретных систем и не ожидать, что возможно-

сти аналитики в реальном времени будут предоставлены им автоматически.

3. ОБЛАЧНЫЕ ХРАНИЛИЩА ДАННЫХ

Технология облачного хранилища данных относительно нова и постоянно меняется. Общепринятых определений или наборов требований пока нет. Все известные реализации, некоторые из которых кратко обсуждаются ниже, имеют очень разные архитектуры и даже разные цели проектирования, но архитектура этих систем полностью отличается от традиционной архитектуры хранилища данных Инмона (William H. Inmon) [1] и Кимбалла (Ralph Kimball) [2]. Некоторые решения перенимают существующие технологии СУБД, интеграции или преобразования данных и т.д.; другие используют новые методы и методы, но все они основаны на представлении таблиц по столбам как более эффективном подходе к поддержке аналитических запросов.

Наиболее важные общие черты современных облачных хранилищ данных выглядят следующим образом:

• Доступ к данным. Перемещение данных в облако позволяет предоставлять аналитику над данными почти реального времени, получаемыми из различных внешних источников, включая внутренние операционные данные предприятий.

• Производительность оценки запросов. Использование современных методов выполнения аналитических запросов с различными видами распараллеливания на основе передового облачного оборудования обеспечивает производительность, близкую к требованиям реального времени.

• Масштабирование. Облачные хранилища данных масштабируются быстро, а иногда и частично автоматически из-за разъединения вычислений и хранения данных.

Эти черты позволяют называть современные облачные хранилища данных облачными сервисами мягкого реального времени.

В этом разделе мы кратко обсудим архитектуру и основные функции четырех популярных современных облачных хранилищ данных:

• Google BigQuery;

• Amazon RedShift;

• Microsoft Azur Synapse Analytics (dedicated SQL pool) и

• Snowflake.

3.1. Google BigQuery

По словам Google, BigQuery – это “бессерверное, масштабируемое и экономичное мультиоблачное хранилище данных, разработанное для обеспечения адаптивности бизнеса”. [54]. Общая архитектура BigQuery показана на рис. 14.

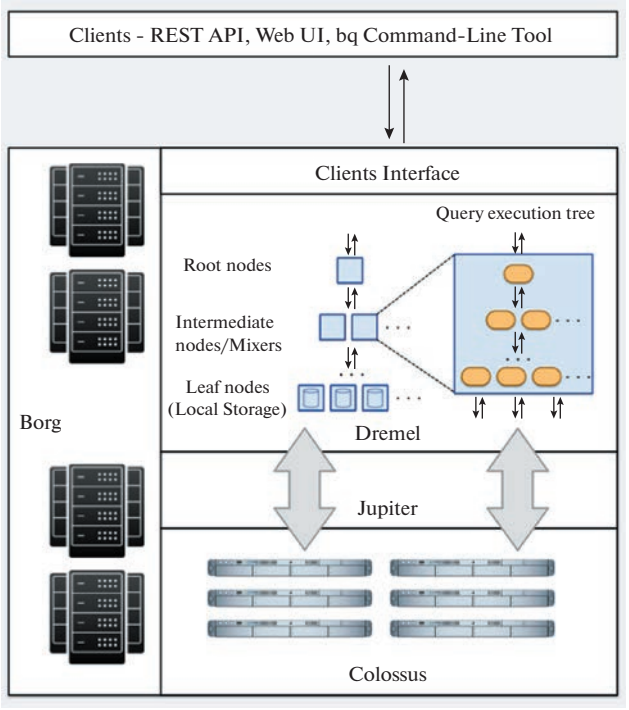

Рис. 14. Высокоуровневая архитектура службы BigQuery.

Основные компоненты BigQuery:

• Dremel – система обработки запросов;

• Borg – система управления кластерами;

• Jupiter – сетевая инфраструктура;

• Colossus – распределенная файловая система.

Dremel [56, 57] – старейший компонент BigQuery, это масштабируемая, интерактивная (быстрая) система обработки произвольных запросов для анализа неизменяемых вложенных данных. Задание Dremel

• считывает данные из файловых систем Google Colossus по сети Jupiter;

• выполняет различные операции SQL и;

• возвращает результаты клиенту.

Dremel использует оригинальное поколоночное представление таблиц Capacitor [58], которое поддерживает вложенные структуры данных и прямые операции над сжатыми данными.

Общая философия обработки запросов в Google такова.

• Никогда не известно, какие запросы потребуются в каждой возможной ситуации.

• Решение BigQuery состоит в том, чтобы увеличить скорость полного сканирования, получая доступ ко всем записям на дисках без индексации или предварительно агрегированных значений.

Поэтому используются многоуровневые деревья обслуживания и тысячи вычислительных узлов для выполнения запросов.

Корневой узел получает входящие запросы, считывает метаданные из таблиц и направляет запросы на следующий уровень дерева обслуживания. Листовые серверы взаимодействуют с уровнем хранения или обращаются к данным на локальном диске.

Схема очень хорошо подходит для простых запросов с агрегацией. При выполнении сложных запросов с агрегацией и других классов запросов используются механизмы, применяемей в параллельных СУБД и Map/Reduce. Детали не ясны, и информация о реализации недоступна.

Colossus [59] обеспечивает репликацию, восстановление и распределенное управление на уровне кластера. Для импорта данных в хранилище BigQuery можно использовать либо пакетную загрузку, либо потоковую передачу. В процессе импорта BigQuery отдельно кодирует каждый столбец в формате Capacitor.

Как только все данные столбца закодированы, они записываются обратно в Colossus. Во время кодирования собирается различная статистика о данных, которая в дальнейшем используется для планирования запросов. При записи данных в Colossus BigQuery принимает решение о первоначальной стратегии партиционирования, которая основывается на паттернов запросов и доступа к данным. После записи данных для обеспечения максимальной доступности BigQuery инициирует георепликацию данных в разных центрах обработки данных. Существующие записи не могут быть обновлены, поэтому BigQuery в основном поддерживает варианты использования только для чтения. Однако всегда можно записать обработанные данные в новые таблицы.

Борг [60] одновременно запускает тысячи заданий Dremel в одном или нескольких кластерах, состоящих из десятков тысяч машин. Кроме того, помимо выделения вычислительных мощностей для заданий Dremel, Borg обеспечивает отказоустойчивость.

Из-за разделения уровней вычислений и хранения данных для BigQuery требуется сверхбыстрая сеть, которая может за секунды доставлять терабайты данных из среды хранения данных в вычислительные узлы для выполнения заданий Dremel. Google Jupiter [61] позволяет сервису BigQuery использовать сети с пропускной способностью до 1-го петабит/с.

3.2. Amazon RedShift

Amazon RedShift основан на PostgreSQL 8.0.2, но исходный код PostgreSQL был сильно изменен [61]. Схема хранения данных и механизм выполнения запросов полностью отличаются от PostgreSQL.

Данные хранятся по столбцах с использованием специальных методов сжатия данных. Redshift автоматически сжимает все данные, которые загружаются в систему, и распаковывает их во время выполнения запроса. Поддержка вторичных индексов и эффективных операций обработки данных в построчном представлении была исключена.

Общая архитектура RedShift изображена на рис. 15 [62, 63]. Кластер – это основной структурный элемент хранилища данных Amazon Redshift, ответственный за обработку запросов. Каждый кластер Redshift состоит из двух основных компонентов.

• Вычислительный узел (Compute Node) имеет собственный центральный процессор, основную память и дисковое хранилище. Вычислительные узлы хранят данные и выполняют запросы, и в одном кластере может быть много узлов.

• Ведущий узел (Leading Node) управляет связью между вычислительными узлами и клиентскими приложениями. Ведущий узел компилирует код, распределяет скомпилированный код по вычислительным узлам и назначает часть данных каждому вычислительному узлу.

Ведущий узел Redshift и вычислительные узлы работают следующим образом. Ведущий узел получает запросы и команды от клиентских программ. Когда клиенты выполняют запрос, ведущий узел отвечает за синтаксический анализ запросов и создание оптимального плана их выполнения на вычислительных узлах над частями данных, хранящимися в каждом узле. На основе плана выполнения ведущий узел создает скомпилированный код и распределяет его по вычислительным узлам для обработки. Наконец, ведущий узел получает и агрегирует результаты и возвращает результаты клиентскому приложению.

Существует два типа узлов: *dense storage node* и *dense compute node*. Объем внешней памяти каждого узла может варьироваться от 160 ГБ до 16 ТБ – использование самого крупного варианта среды хранения данных в узле позволяет хранить в кластере данные петабайтного масштаба. Горячие данные (*hot data*) хранятся на локальных дисках, исторические данные (*historical data*) – в хранилище объектов S3. По мере роста рабочей нагрузки можно увеличивать вычислительную мощность и емкость среды хранения, добавляя узлы в кластер или изменяя тип узла.

Amazon Redshift использует сетевые подключения с высокой пропускной способностью, непосредственную физическую близость узлов и настраиваемые протоколы связи, чтобы обеспечить высокоскоростную сетевую связь между узлами кластера. Вычислительные узлы работают в отдельной изолированной сети, к которой кли-

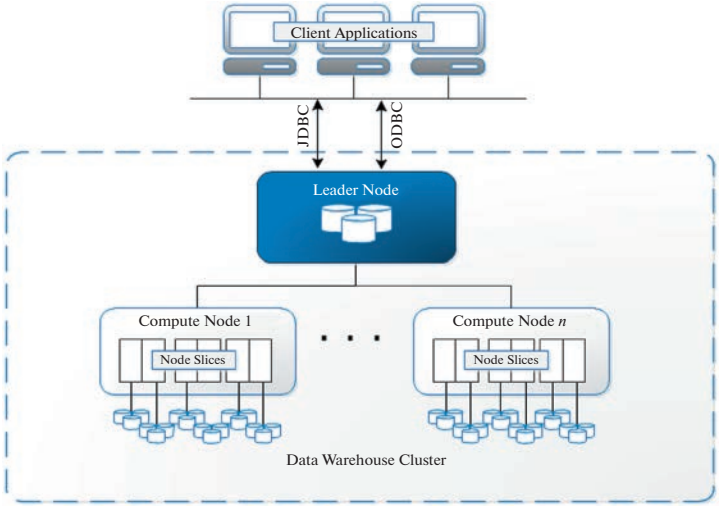

Рис. 15. Высокоуровневая архитектура Amazon Red Shift.

ентские приложения никогда не обращаются напрямую, что обеспечивает дополнительные преимущества в плане безопасности.

Каждый вычислительный узел разбивается на слайсы, и каждый слайс получает часть памяти и дискового пространства. Ведущий узел разделяет данные по слайсам и распределяет между ними части пользовательских запросов или других операций над базой данных. При выполнении операций слайсы работают параллельно. Количество слайсов на узел кластера зависит от размера этого узла кластера.

Внутри узла Redshift может автоматически решать, как распределять данные между слайсами, или можно указать один столбец в качестве ключа распределения. Выполняя запрос, оптимизатор запросов на ведущем узле по мере необходимости перераспределяет данные на вычислительных узлах, чтобы выполнять требуемые операции соединения и агрегации. Предлагаются три “стиля” выбора ключа распределения, чтобы помочь Redshift более эффективно выполнять запросы:

• по общим столбцам разделаются таблица фактов и одна таблицу измерений;

• для разделения выбирается столбец с наибольшим числом различных значений в отфильтрованном частичном результате;

• изменяются некоторые таблицы измерений, чтобы можно было использовать совместное разделение всех таблиц.

Данные, хранящиеся в Redshift, реплицируются на все узлы кластера и автоматически резервируются. Мгновенные снимки сохраняются в S3 и могут быть восстановлены. Amazon отслеживает состояние кластера Redshift, повторно реплицирует данные с отказавших дисков и при необходимости заменяет узлы. Данные могут шифроваться.

Redshift поддерживает SQL-команды UPDATE и DELETE, но не предоставляет команд MERGE или UPSERT для обновления таблицы из одного источника данных. Команда COPY является наиболее эффективным способом загрузки таблицы. Он может читать данные из нескольких файлов данных или нескольких потоков данных одновременно.

3.3. Microsoft Asur Dedicated SQL Poll

В настоящее время Microsoft позиционирует свою современную службу облачного хранилища данных [65, 66] как часть гораздо более крупной службы под названием Azure Synapse Analytics [67] (рис. 16).

Azure Synapse – это корпоративная аналитическая служба, которая ускоряет получение информации из хранилищ данных и систем больших данных. Azure Synapse объединяет технологии SQL, используемые в корпоративных хранилищах данных, технологии Spark, используемые для обработки больших данных, конвейеры для интеграции данных и ETL/ELT, а также обеспечивает глубокую интеграцию с другими службами Azure, такими как Power BI, CosmosDB и AzureML.

В данной работе мы ограничимся рассмотрением компонента Synapse SQL, обеспечивающим функции службы хранилища данных (рис. 17).

Synapse SQL использует масштабируемую архитектуру для распределения вычислительной обработки данных между несколькими узлами. Вычисления отделены от хранилища, что позволяет масштабировать вычисления независимо от данных в системе. Для *выделенного пула SQL* единицей масштабирования является абстракция вычислительной мощности, называемая *data*

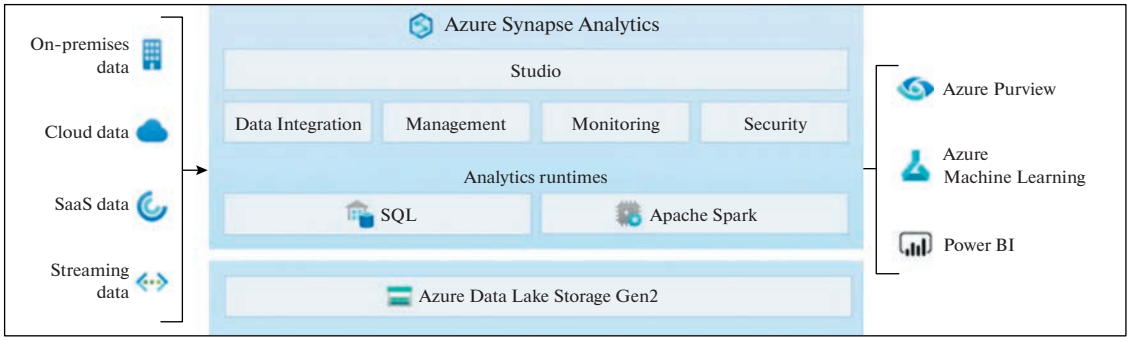

Рис. 16. Azure Synapse Analytics.

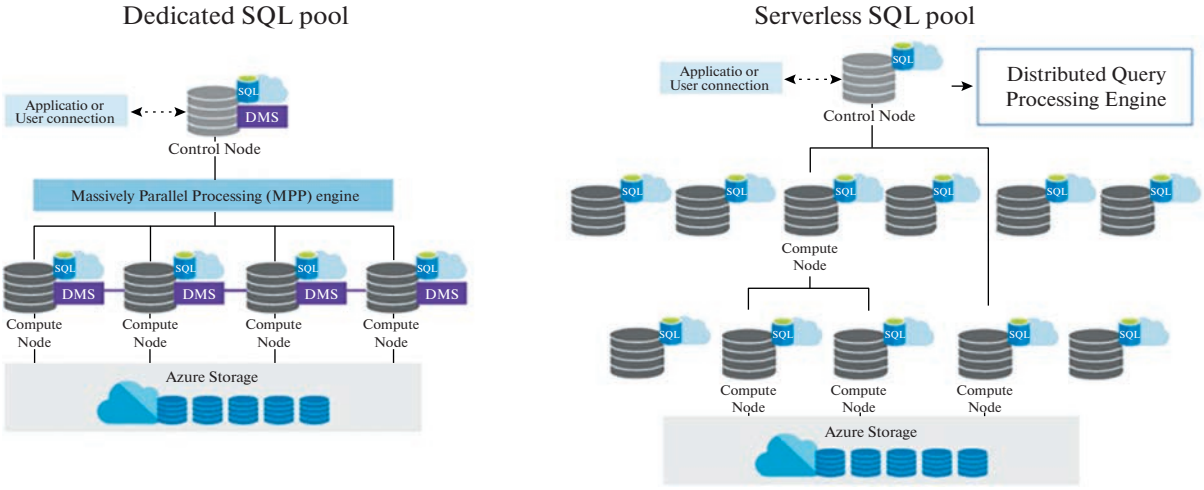

Рис. 17. Архитектура Synapse SQL.

warehouse unit. Масштабирование *бессерверного пула SQL* выполняется автоматически в соответствии с требованиями к ресурсам со стороны запросов. Поскольку топология со временем меняется путем добавления, удаления узлов или аварийного переключения, она адаптируется к изменениям и гарантирует, что запрос имеет достаточно ресурсов и успешно завершается.

Выделенный пул SQL использует архитектуру на основе узлов. Приложения подключаются к *управляющему узлу* и выдают команды T-SQL. На узле управления размещается механизм распределенной обработки запросов, который оптимизирует запросы для параллельной обработки, а затем передает операции вычислительным узлам для параллельного выполнения своей работы. Каждый из нескольких вычислительных узлов содержит экземпляр базы данных SQL и отвечает за обработку данных, хранящихся локально на его дисках. После получения промежуточных результатов вычислительные узлы возвращают эти результаты на управляющий узел для агрегирования.

Выделенный пул SQL сохраняет данные в таблицах с хранением по столбцам. По сравнению с традиционными системами баз данных аналитические запросы выполняются за секунды, а не за минуты или часы. Благодаря разделению средств хранения данных и вычислительной обработки при использовании выделенного пула SQL можно:

- масштабировать вычислительную мощность независимо от потребностей в хранении данных;
- увеличивать или уменьшать вычислительную мощность без перемещения данных;
- приостанавливать использование вычислительных ресурсов, оставляя данные сохранными и платя только за их хранение;
- возобновлять использование вычислительных ресурсов, когда это требуется.

Выделенный пул SQL использует службу хранения данных Azure (хранилище BLOB-объектов Azure) для обеспечения безопасности пользовательских данных. Данные разделяются на дистрибуции (*distribution*) для оптимизации производительности системы. При определении таблицы

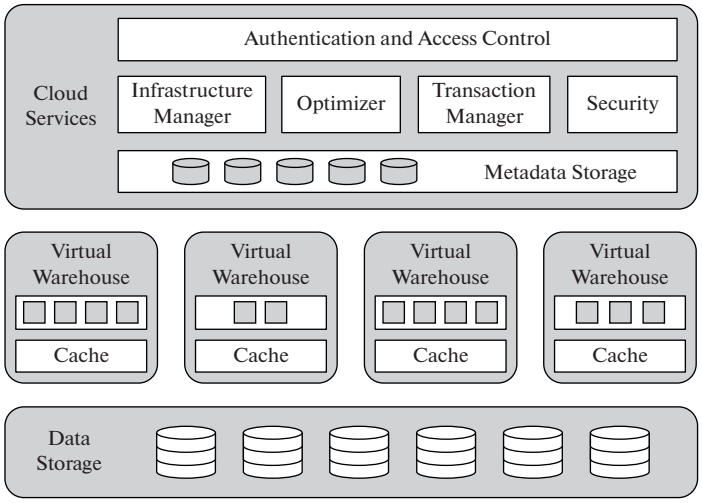

Рис. 18. Архитектура Snowflake.

можно выбирать, какой шаблон будет использоваться для разделения данных. Поддерживаются следующие шаблоны разделения:

- разделение на основе хэширования;
- циклическое разделение;
- разделение с репликацией.

Дистрибуция – это базовая единица хранения и обработки для параллельных запросов, выполняемых над распределенными данными. Когда Synapse SQL выполняет запрос, эта работа делится на 60 более мелких запросов, которые выполняются параллельно. Каждый из 60 мелких запросов выполняется над одной из дистрибуций данных. Каждый вычислительный узел управляет одной или несколькими из 60 дистрибуций. Выделенный пул SQL с максимальным количеством вычислительных ресурсов имеет одно распределение на вычислительный узел. Выделенный пул SQL с минимальными вычислительными ресурсами содержит все дистрибутивы на одном вычислительном узле.

Пул SQL поддерживает транзакции как часть рабочей нагрузки хранилища данных [68]. Однако, чтобы обеспечить поддержку пула SQL в масштабе, некоторые функции ограничены по сравнению с SQL Server:

- Отсутствие распределенных транзакций;
- Не разрешены вложенные транзакции;
- Нет разрешенных точек сохранения;
- Нет именованных транзакций;
- Нет отмеченных транзакций;
- Нет поддержки DDL, такой как CREATE TABLE внутри определяемой пользователем транзакции.

3.4. Snowflake

Snowflake Inc. – самая молодая компания на рынке современных облачных хранилищ данных. Она была основана в 2012 году и уже в 2014 году запустила свой сервис Snowflake [69, 70]. В настоящее время Snowflake является одним из основных конкурентов мировых лидеров рынка. Архитектура Snowflake показана на рис. 18.

- Уровень хранилища данных (Data Storage) использует Amazon S3 для хранения табличных данных и результатов запросов.
- Уровень виртуальных хранилищ (Virtual Warehouse) обрабатывает выполнение запросов в эластичных кластерах виртуальных машин, называемых виртуальными хранилищами.
- Уровень облачных служб (Cloud Services) – это набор служб, которые управляют виртуальными хранилищами, запросами, транзакциями и всеми сопутствующими метаданными: схемами баз данных, информацией об управлении доступом, ключами шифрования, статистикой использования и т.д.

Хранилище данных. Несмотря на то, что производительность S3 может отличаться от случая к случаю, удобство использования этой службы, ее высокую доступность и надежные гарантии долговечности хранения данных трудно превзойти. Было решено потратить основные силы на разработку методов локального кэширования данных и обеспечения устойчивости к перекосам на уровне виртуальных хранилищ. Файлы S3 могут быть (пере)записаны только полностью; невозможно даже добавить данные в конец файла. Однако S3 поддерживает запросы GET для частей файла. Эти свойства оказали сильное влияние на формат файла таблицы Snowflake и схему управления параллелизмом.

Таблицы горизонтально разделяются на большие неизменяемые файлы, которые являются эквивалентами блоков или страниц в традиционной системе баз данных. Внутри каждого файла значения каждого атрибута, или столбца сгруппированы вместе и сильно сжаты с использованием схемы PAX [71]. Каждый файл таблицы имеет заголовок, который, среди прочих метаданных, содержит смещения каждого столбца в файле. Поскольку S3 разрешает запросы GET к частям файлов, исполнителям запросов нужно загружать только заголовки файлов и те столбцы, которые их интересуют.

S3 также используется для хранения временных данных, сгенерированных исполнителями запросов, после исчерпания места на локальном диске, а также больших результатов запросов. Перенос временных данных в S3 позволяет системе выполнять произвольно сложные запросы без проявления ошибок нехватки основной или дисковой памяти.

Хранение результатов запросов в S3 позволяет использовать новые формы взаимодействия с клиентами и упрощает обработку запросов, поскольку устраняет необходимость в курсорах на стороне сервера. Метаданные – объекты каталога, статистика, блокировки и т.д. – хранятся в масштабируемом транзакционном хранилище "ключ-значение", которое является частью уровня облачных служб.

Виртуальные хранилища. Уровень виртуальных хранилищ состоит из кластеров экземпляров EC2. Каждый такой кластер предоставляется единственному пользователю через абстракцию, называемую виртуальным хранилищем (VW). Отдельные экземпляры EC2, составляющие VW, называются рабочими узлами. Пользователи не знают и не заботятся о том, какие или сколько рабочих узлов составляют VW. Вместо этого характеристики VW представлены абстрактными "размерами футболок" от X-Small до XX-Large.

VW – это чистые вычислительные ресурсы. Их можно создавать, уничтожать или изменять размер в любой момент по требованию. Каждый отдельный запрос выполняется ровно на одном VW. Рабочие узлы не используются совместно несколькими виртуальными машинами, что приводит к строгой изоляции производительности выполнения запросов. Когда поступает новый запрос, каждый рабочий узел в соответствующем VW порождает новый рабочий процесс. Каждый рабочий процесс существует только на время выполнения своего запроса. У каждого пользователя в любой момент времени может быть запущено несколько виртуальных машин, и каждая виртуальная машина, в свою очередь, может одновременно выполнять несколько запросов. Каждый VW имеет доступ к одним и тем же таблицам без необходимости физического копирования данных.

Каждый рабочий узел поддерживает кэш табличных данных на локальном диске. Кэш – это набор файлов таблиц, к которым в прошлом обращался узел, точнее, заголовки файлов и отдельные столбцы файлов, поскольку исполнители запросов загружают только те столбцы, которые им нужны. Чтобы улучшить частоту успешных обращений к кэшу и избежать избыточного кэширования отдельных файлов таблиц на рабочих узлах VW, оптимизатор запросов назначает наборы входных файлов рабочим узлам, используя консистентное хэширование (consistent hashing) [72] имен файлов таблиц.

Обработка перекосов также особенно важна в облачном хранилище данных. Некоторые узлы могут работать намного медленнее других из-за проблем с виртуализацией или конфликтов в сети. Помимо прочего, Snowflake решает эту проблему на уровне сканирования:

• всякий раз, когда рабочий процесс завершает сканирование своего набора входных файлов, он запрашивает дополнительные файлы у других узлов того же VW (перехват – stealing);

• если при поступлении такой заявки рабочий узел обнаруживает, что в его наборе файлов осталось много необработанных файлов, он отвечает на эту заявку, передавая запросившему узлу право собственности на один оставшийся файл на время выполнения текущего запроса;

• затем запросивший узел загружает файл непосредственно из S3, а не из кэша прежнего владельца файла.

Основные характеристики *исполнителя запросов SQL*: использование поколоночного представления таблиц, векторизованное исполнение и применение техники проталкивания (push). Векторизованное исполнение означает, что Snowflake избегает материализации промежуточных результатов; данные обрабатываются конвейерным способом, пакетами по несколько тысяч строк в поколоночном формате. Исполнение на основе проталкивания означает, что реляционные операции передают свои результаты нижестоящим операциям, а не ждут, пока эти операции сами извлекут данные (как в модели в стиле Volcano). Это позволяет повысить эффективность использования кэша и обрабатывать планы выполнения запросов, представляемые в виде ориентированных графов без циклов, а не просто деревьев.

В Snowflake отсутствуют многие источники накладных расходов, возникающих при традиционной обработке запросов. Нет необходимости в управлении транзакциями во время выполнения; запросы выполняются над фиксированным набором неизменяемых файлов. Нет буферного пула в

основной памяти. Большинство запросов сканируют большие объемы данных. Использование основной памяти для буферизации таблиц, а не для выполнения операций в данном случае является плохим вариантом. Однако Snowflake позволяет всем основным операциям (объединение, группировка, сортировка) переносить данные на диск и обратно при исчерпании основной памяти. Исполнитель запросов, работающий полностью а основной памяти, хотя и компактнее и, возможно, быстрее, слишком ограничен для поддержки всех интересных рабочих нагрузок.

Облачные сервисы. Слой облачных сервисов является мультитенантным. Каждая служба этого уровня – управление доступом, оптимизатор запросов, менеджер транзакций и другие – является долгоживущей и совместно используемой многими пользователями. Каждая служба реплицируется для обеспечения высокой доступности и масштабируемости. Поэтому сбой отдельных сервисных узлов не приводит к потере данных или доступности, хотя некоторые выполняемые запросы могут завершаться с ошибкой (и быть пригодными для повторного выполнения).

Управление запросами и оптимизация. Все запросы, выдаваемые пользователями, проходят через уровень облачных служб. Обрабатываются все ранние этапы жизненного цикла запроса: синтаксический анализ, контроль доступа и оптимизация плана выполнения запроса. Оптимизатор запросов следует типичному подходу в стиле Cascades [73] с нисходящей оптимизацией на основе оценки стоимости. Вся статистика, используемая для оптимизации, автоматически поддерживается при загрузке и обновлении данных. Поскольку Snowflake не использует индексы, пространство поиска плана меньше, чем в некоторых других системах. Пространство плана еще больше сокращается за счет откладывания многих решений до времени выполнения, например, определение типа распределения данных при выполнении соединений. Такой подход уменьшает количество неправильных решений, принимаемых оптимизатором, повышая надежность за счет небольшой потери пиковой производительности. После завершения работы оптимизатора результирующий план выполнения распространяется на все рабочие узлы, являющиеся частью запроса.

Параллельное управление. В аналитических рабочих нагрузках, как правило, преобладают крупные операции чтения, массовые или незначительные вставки и массовые обновления. ACID-транзакции поддерживаются с помощью изоляции мгновенных снимков (Snapshot Isolation, SI): все операции чтения внутри транзакции видят согласованный мгновенный снимок базы данных ко времени начала транзакции. SI реализуется поверх многоверсионного контроля параллелизма (multi-version concurrency control, MVCC), что означает, что копия каждого измененного объекта базы данных сохраняется в течение некоторого времени. Операции записи (вставка, обновление, удаление, слияние) в таблице создают более новую версию таблицы путем добавления и удаления целых файлов в предыдущей версии таблицы. Эти мгновенные снимки также используются для реализации переходов по времени и эффективного клонирования объектов базы данных.

Отсечение (pruning) против индексации. Поддержка индексов может быть сложным, дорогостоящим и рискованным процессом. Произвольный доступ является проблемой как из-за среды хранения данных (S3), так и из-за формата данных (сжатые файлы); поддержка индексов значительно увеличивает объем данных и время загрузки данных; пользователям необходимо явно создавать индексы, что сильно противоречит сервисному подходу. Альтернативным методом является использование отсечений на основе минимального и максимального значений, карт зон (zone map) и пропуск данных (data skipping). Система поддерживает информацию о распределении данных для заданного фрагмента данных (набор записей, файл, блок и т.д.), в частности, минимальные и максимальные значения в чанке.

3.5. Преимущества, ограничения и компромиссы

Несмотря на различные архитектурные и реализационные подходы, в рассмотренных системах имеются несколько важных общих черт, которые позволяют им обеспечивать быструю (почти в реальном времени) аналитику на достаточно свежих (почти в реальном времени) данных. К общим *выгодным* для пользователей особенностям относятся:

- поколоночный сжатый формат для хранения таблиц и работы с ними;
- массивно-параллельная обработка запросов с логически не разделенными ресурсами между узлами;
- логическое горизонтальное разделение базы данных между вычислительными узлами;
- физическое разделение вычислительных ресурсов и ресурсов хранения с физической возможностью для любого вычислительного узла получить доступ ко всем данным в общем хранилище данных;
- раздельное масштабирование вычислительной части и системы хранения;
- надежная поддержка использования локального хранилища вычислительных узлов для индивидуального кэширования физически общих данных;
- некоторый вид контроля параллелизма, который позволяет, по крайней мере, оперативно

добавлять новые данные в массовом порядке или небольшими порциями.

Однако, как и в случае DSMS, свежесть данных сильно зависит от качества источников данных, а скорость обработки запросов сильно зависит от сложности конкретного запроса. Ответ в течение одной секунды не может быть гарантирован для любого запроса. Таким образом, свойство обработки в мягком реальном времени в основном условно. Это очевидные *ограничения*.

Поэтому разработчики приложений и аналитики не должны слепо верить в то, что современные облачные хранилища данных работают в режиме мягкого реального времени. Они должны трезво оценивать свои реальные потребности и возможности конкретных облачных сервисов.

4. ГИБРИДНАЯ ТРАНЗАКЦИОННО-АНАЛИТИЧЕСКАЯ ОБРАБОТКА ДАННЫХ

Наиболее естественным источником свежих корпоративных данных являются данные, генерируемые транзакциями одного и того же предприятия. Обычно все корпоративные транзакции обрабатываются какой-либо СУБД, ориентированной на OLTP, а корпоративная аналитика поддерживается какой-либо СУБД, ориентированной на OLAP. Для обеспечения аналитики в реальном времени необходимо очень быстро передавать данные из хранилища транзакционной системы в хранилище аналитической системы. Другими словами, для обеспечения аналитики свежих транзакционных данных необходимо обеспечить очень быстрый механизм ETL. Но никто не знает, как реализовать такой быстрый ETL, и чтобы сделать возможной аналитику в реальном времени, была введена концепция гибридной транзакционно-аналитической обработки (HTAP).

“Идеалистическая” цель HTAP-подхода – одновременно обеспечить в рамках одной системы и функциональность, и производительность специализированных OLTP- и OLAP-СУБД – представляется недостижимой. Прагматическая цель этого подхода – построить систему, которая обеспечивает приемлемую пропускную способность для транзакционных рабочих нагрузок и одновременно аналитику в режиме реального времени на достаточно свежих данных – несомненно достижима.

Требования разумной пропускной способности для транзакционных рабочих нагрузок и актуальности данных для аналитических запросов явно противоречат друг другу. Для обеспечения максимальной актуальности данных необходимо сделать доступными для анализа все данные, генерируемые выполняемыми в данный момент транзакциями. Однако тогда транзакции будут конкурировать с аналитическими запросами, и обработка транзакций станет медленнее. Все известные системы HTAP используют тот или иной компромисс, чтобы разумно (в некоторой степени) удовлетворить эти противоречивые требования путем разделения аналитической и транзакционной частей базы данных и, возможно, преобразования данных из представления, хорошо подходящего для обработки транзакций (как правило, построчного формата хранения таблиц) в представление, более подходящее для аналитики (обычно поколоночный формат хранения таблиц).

Еще одним аспектом являются среда хранения данных. Большинство современных СУБД категории HTAP оптимизированы для хранения данных в основной памяти. Это означает, что все структуры данных и алгоритмы, используемые в таких системах, разработаны с учетом того, что вся обработка данных будет выполняться в основной памяти без ввода/вывода с внешними запоминающими устройствами. Вообще говоря, концепция in-memory СУБД не нова. Согласно [74], первая попытка реализовать систему баз данных, хранящую все данные в памяти, была предпринята IBM в 1976 г. (IMS Fast Path). Новая волна in-memory СУБД наблюдалась в 1990-х годах. Тогда и позже (в начале 2000-х) такие СУБД в основном специализировались на обработке транзакций. Сейчас почти все существующие HTAP-СУБД в той или иной степени относятся к классу in-memory систем. Хранение всех данных (или, по крайней мере, *горячих* данных) в основной памяти обеспечивает более высокую скорость обработки транзакций, что частично сглаживает негативные последствия параллельно выполняемых аналитических запросов.

В этом разделе мы кратко обсудим происхождение и основные архитектурные особенности некоторых систем категории HTAP. Более подробный обзор см., например, в [7].

4.1. SAP HANA

HANA, аббревиаиура от *High-Performance Analytic Appliance*, является основным продуктом управления базами данных компании SAP SE. Прежде всего, система используется в составе ERP-решений самой SAP. Однако HANA набирает все большую популярность как отдельный инструмент для разработки и поддержки приложений баз данных, которым требуется как транзакционная, так и аналитическая обработка данных. Система доступна в различных воплощениях, таких как кластерная или облачная версии. Однако здесь мы ограничиваемся основной конфигурацией HANA на одном компьютере.

HANA не разрабатывалась с нуля. На дизайн исполнительной подсистемы HANA с поколо-

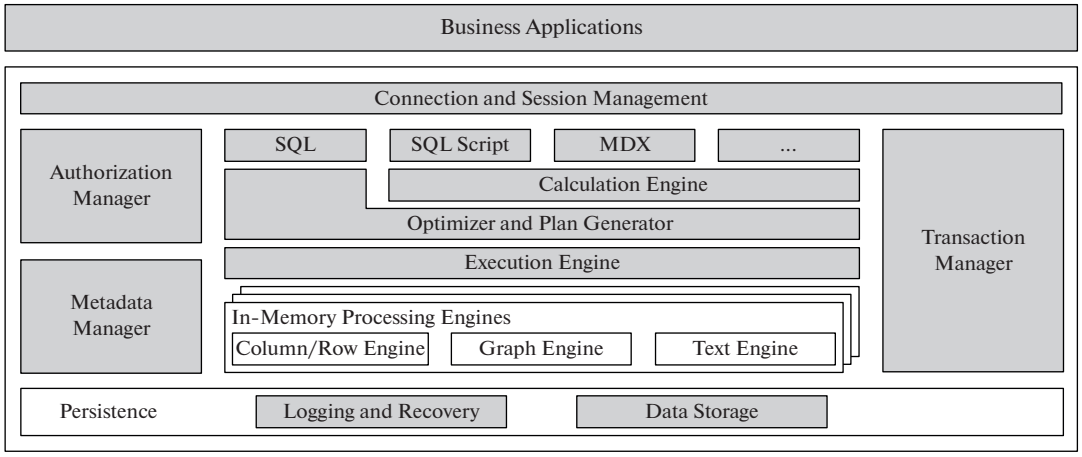

Рис. 19. Архитектура HANA.

ночным хранением таблиц в основной памяти повлияли собственные разработки SAP, система текстового поиска TREX (2001 г.) [75] и SAP Business Warehouse Accelerator (BWA, 2005 г.) [76]. Исполнительная подсистема с хранением таблиц по строкам в основной памяти основана на решении P*TIME [77], разработанном компанией Transact in Memory, Inc., которое было приобретено SAP в 2005 г. Механизм MaxDB [78], приобретенный SAP в 1997 г. у Software AG (позже называвшийся Adabas D) использовался для обеспечения уровня персистентного хранения данных. Первый вариант HANA был выпущен SAP в конце 2010 года.

Рис. 19 демонстрирует общую архитектуру HANA [79]. Мы обсуждаем только часть этой архитектуры, связанную с механизмами HTAP в основной памяти.

HANA позволяет использовать для одной и той же базы данных как хранилища таблиц по столбцам, так и хранилища по строкам. При создании новой таблицы пользователь может выбрать способ ее хранения: по строкам или по столбцам. Согласно исходным проектным решениям [16], поколоночное хранилище является предпочтительным, а HANA оптимизирована для такого типа организации таблиц. Вероятно, использование поколоночного хранилища не позволяет системе продемонстрировать максимальную производительность обработки транзакций, поскольку обновление хранилища по столбцам – это трудоемкая операция. Однако взамен колоночный способ хранения базы данных снижает затраты памяти за счет сжатия данных. Чтобы достичь более высокой скорости обработки транзакций, можно использовать дорогостоящее хранилище строк.

Как правило, каждый столбец таблицы хранится в хранилище столбцов с использованием двух структур данных – словаря и индексного вектора. Словарь содержит все различные значения, находящиеся в данный момент в столбце, а индексный вектор связывает каждую строку таблицы с соответствующей записью словаря. Для поддержки распространенных в аналитике запросов в диапазоне значений HANA сохраняет отсортированный порядок записей словаря. Однако это свойство делает очень дорогими операции обновления, часто используемые при обработке транзакций. Поэтому для каждого столбца поколоночной таблицы имеются две части хранилища: основная часть хранит словарь столбца в отсортированном порядке, а в дельта-части этот порядок не поддерживается. Все операции над таблицей используют как основные данные каждого столбца, так и его дельту. Основываясь на информации о размере дельты и текущей рабочей нагрузки, система решает, когда следует выполнить слияние основной части данных столбца с его дельтой. Во время выполнения этой операции используется новая дельта.

SAP рекомендует использовать хранилище строк для небольших таблиц, в которых строки часто обновляются или удаляются. Таблицы, хранящиеся по строкам, всегда сохраняются в основной памяти. Однако таблицы, которые хранятся по столбцам, могут быть очень большими, и они могут быть разбиты на два раздела, один из которых хранится в памяти, а другой – на дисках. SAP называет раздел таблицы в основной памяти текущим, а раздел на диске – историческим (другие поставщики вместо этого используют термины “горячий” и “холодный” соответственно). В некотором смысле текущая часть таблицы должна содержать данные, которые активно используются в данный момент. HANA может автоматически распределять данные между этими разделами в зависимости от времени жизни определенных строк таблицы или с использованием явных подсказок, предоставляемых приложениями. Теку-

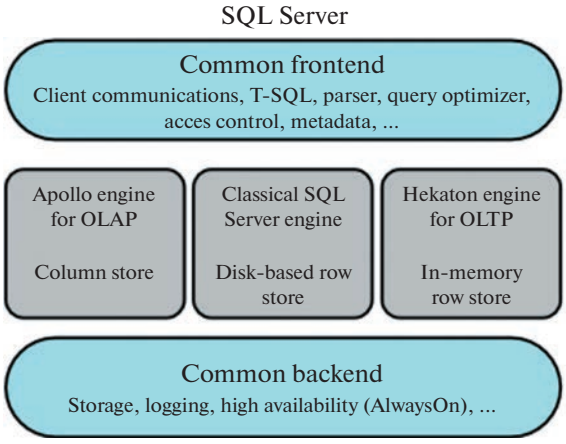

Рис. 20. Упрощенная архитектура SQL Server 2014.

щие и исторические разделы имеют разные схемы хранения, но доступ к ним можно получить с помощью одного запроса.

Имеются несколько дополнительных архитектурных решений, позволяющих одновременно и эффективно обрабатывать транзакции и аналитические запросы в одной системе. Во-первых, каждый аналитический запрос компилируется и оптимизируется с учетом самой свежей статистики базы данных. Запросы, используемые в повторяющихся (параметризованных) транзакциях, готовятся заранее; при обработке транзакций используются готовые исполняемые планы запросов. Во-вторых, аналитические запросы выполняются на высоком уровне параллелизма с использованием нескольких потоков, работающих на разных ядрах ЦП. Распараллеливание транзакционных запросов считается вредным: переключение контекста обходится слишком дорого. Поэтому каждый транзакционный (простой) запрос полностью выполняется в рамках одного потока.

Наконец, чтобы свести к минимуму негативные, потенциально замедляющие обработку транзакций эффекты конкуренции между операциями транзакций и аналитическими запросами, транзакционные операции всегда имеют более высокий приоритет, чем запросы OLAP.

4.2. Аналитика реального времени в основной памяти компании Microsoft

Похоже, компания Microsoft внедрила в SQL Server этот вид аналитики реального времени в три этапа. Во-первых, Microsoft разработала поколоночное хранилище на диске [80]. Затем была разработана и реализована система OLTP в основной памяти Hekaton [81]. И, наконец, компания разработала аналитическую систему для своего хранилища по столбцам и интегрировала все эти компоненты в SQL Server 2014 (и усовершенствовала полученный результат в SQL Server 2016 и SQL Server 2019), обеспечив тем самым аналитику в реальном времени в стиле HTAP [82].

Упрощенная архитектура SQL Server 2014 изображена на рис. 20.

Аналитику в реальном времени обеспечивают компоненты Hekaton и Apollo. Hekaton – это специализированная транзакционная подсистема, использующая хранилище таблиц по строкам в основной памяти. Apollo – это специализированная аналитическая подсистема, использующая хранилище таблиц по столбцам в дисковой памяти. Собственно, Apollo – это не название какого-либо продукта, а всего лишь условное название проекта. Оперативные данные очень быстро передаются из Hekaton в Apollo и преобразуются из представления по строкам в поколоночный формат. (Последнюю операцию можно рассматривать как разновидность облегченного ETL.)

Основные архитектурные идеи Hekaton заключаются в следующем. Индексы (хэш и своего рода B-Tree) разработаны и оптимизированы для данных, постоянно хранимых в основной памяти. Операции с индексами не журнализуются, и все индексы перестраиваются во время восстановления базы данных. Все внутренние структуры данных – описатели свободной и занятой памяти, хэш-индексы и индексы диапазонов, а также карта транзакций – полностью свободны от блокировок. Операции SQL и хранимые процедуры компилируются в настраиваемый высокоэффективный машинный код.

Система идентифицирует горячие и холодные данные и сохраняет в основной памяти только горячие данные [83]. Основная идея заключается в том, что не все данные транзакционной базы данных должны быть доступны одинаково быстро; часть из них может быть размещена во внешнем хранилище без замедления обработки транзакций. Hekaton использует LRU-подобный алгоритм, чтобы определить, какие данные активно не используются транзакциями, и перемещает соответствующие части таблиц в дисковое хранилище. Разумеется, в памяти и на дисках используются разные физические схемы расположения строк. Представление на основе столбцов для целей аналитики поддерживается для обеих частей таблиц.

4.3. Oracle Database In-Memory

In-memory решение Oracle основано на in-memory СУБД TimesTen [84]. Проект TimesTen был основан в Hewlett-Packard Laboratories Мари-Энн Неймат (Marie-Anne Neimat). В то время система называлась SmallDB [85] и предназначалась для использования во внутренних целях HP. В 1996 году Неймат основала новый стартап TimesTen Inc.

С этого времени TimesTen активно использовалась в качестве транзакционной СУБД реального времени во многих приложениях, особенно в сфере телекоммуникаций.

В 2005 году TimesTen была приобретена Oracle, интегрирована с общей инфраструктурой Oracle и начала использоваться в качестве кэша основной СУБД Oracle. Интегрированная опция Oracle Database In-Memory, поддерживающая функциональность HTAP, впервые появилась в Oracle 12c в 2013 году [86].

Система поддерживает два типа хранилищ данных: хранилище по строкам находится на дисках и хранилище по столбцам – в памяти. Можно выборочно назначить важные таблицы или их разделы резидентными в хранилище по столбцам, используя при этом буферный кеш для остальной части таблицы. Итак, есть две основные области памяти для размещения частей базы данных: традиционный кеш для блоков внешней памяти и хранилище по столбцам в основной памяти.

Части одной и той же таблицы, хранящиеся в разных представлениях (и в разных хранилищах), обновляются синхронно.

Хранилище по строкам обеспечивает быстрый транзакционный доступ к данным с помощью тщательно разработанных индексов и кэширования блоков в памяти. Хранилище по столбцам оптимизировано для обработки аналитических запросов. Свежесть данных для аналитики обеспечивается автоматической синхронизацией содержимого хранилищ по столбцам и по строкам при вставке, удалении или обновлении строк. Многоверсионность, используемая в управлении транзакциями, позволяет снизить конкуренцию за данные между транзакциями и аналитическими запросами.

Oracle Database In-Memory может быть развернута в инфраструктуре Oracle Real Application Cluster (RAC) [87]. Общая архитектура показана на рис. 21.

В этом случае доступ к блокам данных, содержащим части хранилища строк, и их изменение осуществляется через общий буферный кэш базы данных. Кроме того, каждый отдельный экземпляр базы данных можно настроить на использование частного хранилища по столбцам столбцов в основной памяти. Все таблицы горизонтально разделяются по всем узлам кластера.

Если система работает в режиме in-memory, Exadata Database Machine автоматически переформатирует все данные, к которым осуществляется доступ, в поколоночный формат в основной памяти и сохраняет их в так называемом flash-кэше (внешнее хранилище большого объема на основе flash-памяти) [88]. После этого все части запросов, выгруженные в Exadata, выполняются так же, как если бы они обрабатывались в памяти в узле RAC. В частности, Exadata использует инструкции SIMD для обработки столбцов таблицы.

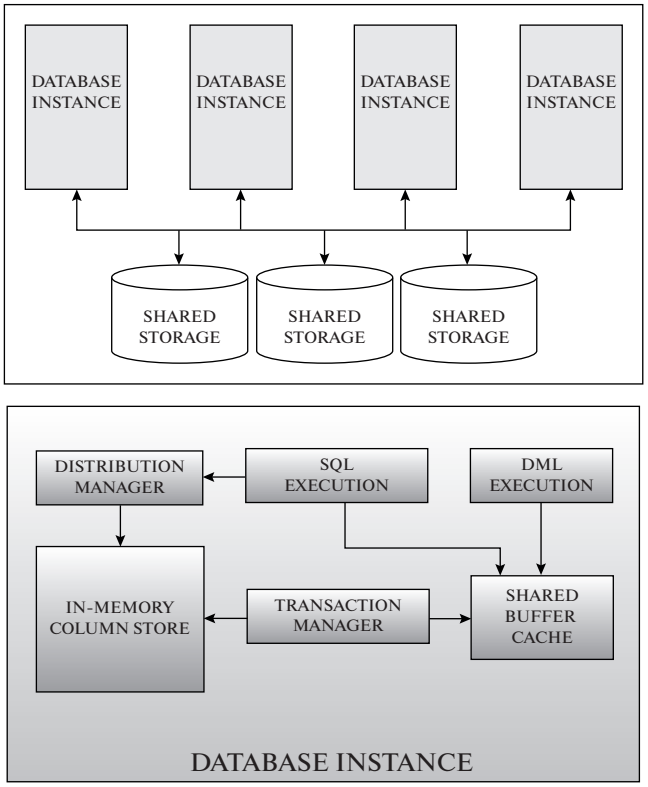

Рис. 21. Архитектура Database In-Memory в Oracle RAC.

4.4. DB2 with BLU Acceleration

В IBM DB2 ускорение аналитики осуществляется с использованием поколоночного хранилища в основной памяти, многоядерного распараллеливания и аппаратной векторной обработки. В 2006 г. началась работа IBM над собственной поколоночной подсистемой в основной памяти. Первым in-memory продуктом, разработанным командой Гая Лохмана (Guy Lohman), был Blink [89]. Это в чистом виде подсистема хранения столбцов в основной памяти. Она успешно используется и сейчас, например, в Informix Warehouse Accelerator [90]. BLU [91] – это продукт второго поколения. Общая архитектура DB2 с BLU показана на рис. 22 [92].

В базе данных, управляемой DB2 с BLU, каждая таблица может храниться либо в традиционном хранилище по строкам, либо в хранилище BLU по столбцам BLU. Важной особенностью BLU является то, что таблицы в поколоночном хранилище могут быть больше, чем размер доступной основной памяти. Фактически все таблицы хранятся в блоках внешней памяти и доступны через обычный блочный кэш.

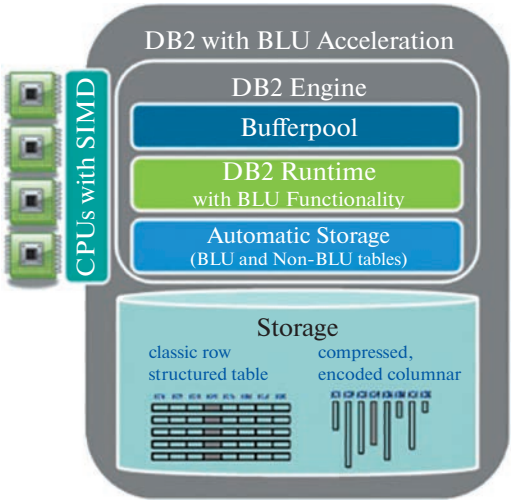

Рис. 22. Архитектура DB2 with BLU.

Однако BLU пытается полностью обработать каждый запрос в основной памяти (хотя технически в этом нет необходимости) и обеспечивает соответствующее управление рабочей нагрузкой. Чтобы обеспечить доступность всех необходимых данных в памяти, BLU использует агрессивную политику предварительной выборки.

Другие особенности DB2 с BLU кажутся традиционными для современных in-memory СУБД, хотя разработчики BLU подчеркивают очень высокую эффективность их реализации. Система поддерживает возможность сканирования и сравнения сжатых данных. Разработчики избегают блокировок, чтобы обеспечить максимальное распараллеливание. Все структуры данных спроектированы таким образом, чтобы свести к минимуму промахи кэша данных и кэша инструкций. Активно используются SIMD-инструкции.

4.4. HyPer

Проект HyPer реализуется командой базы данных Мюнхенского технического университета под руководством Альфонса Кемпера и Томаса Ноймана. Хотя это университетский проект, он не имеет открытого исходного кода. Исходный код системы никогда не публиковался. Более того, в 2015 году был создан околоуниверситетский стартап HyPer, а в 2016 году его приобрела компания Tableau Software, которая называет свою версию СУБД Hyper [93]. Мы не знаем подробностей этой сделки, но каким-то образом развитие HyPer в Мюнхенском университете продолжается. Первый известный отчет о проекте [94] был опубликован в 2010 г.

HyPer – настоящая СУБД в оперативной памяти для многоядерных компьютеров. Система поддерживает как строковые, так и поколоночные хранилища. Начальный формат таблицы может быть выбран при создании таблицы, а физическая структура этой таблицы может меняться в зависимости от рабочей нагрузки.

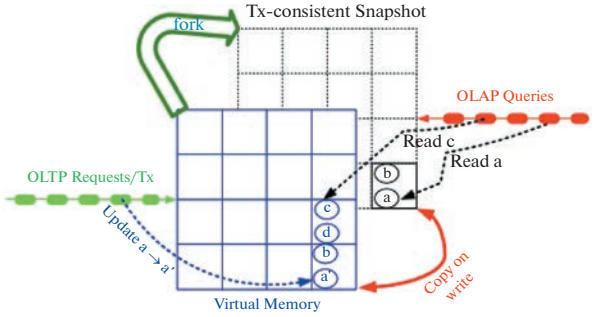

Рис. 23. Мгновенные снимки виртуальной памяти.

Первая версия архитектуры HyPer была очень простой и элегантной, хотя и имела некоторые ограничения. Основные идеи заключались в следующем [95].

Предполагалось, что большинство транзакций могут быть подготовлены заранее (pre-canned), содержать лишь несколько простых запросов и обращаться только к нескольким кортежам. Такие транзакции называются короткими. Все короткие транзакции выполняются последовательно в основном OLTP-процессе. Вся основная область памяти, в которой находится база данных, отображается в виртуальную память этого процесса.

Процесс OLTP периодически (раз в несколько секунд) разветвляет процесс OLAP после фиксации какой-либо только что завершенной транзакции. После этого процесс OLTP включает механизм копирования при записи, который предоставляет новую страницу основной памяти для любой первой операции записи в соответствующую страницу виртуальной памяти. Таким образом, вновь созданный OLAP-процесс видит в своей виртуальной памяти согласованный образ базы данных (согласованный мгновенный снимок). Процесс OLAP выполняет аналитические запросы над этим мгновенным снимком, свежесть которого определяется периодом создания новых процессов OLAP (см. рис. 23).

Транзакция, которая выполняет слишком много запросов или обращается к слишком большому количеству кортежей, считается длинной. Обнаруживаемая выполняемая длинная транзакция откатывается и перенаправляется в текущий процесс OLAP, который имитирует ее выполнение на текущем согласованном мгновенном снимке. Затем эта (частично измененная) транзакция снова перенаправляется в общую очередь транзакций как короткая транзакция. При ее выполнении система сначала проверяет все смоделиро-

ванные обновления по фактическому состоянию базы данных, а затем, если эта проверка проходит успешно, выполняет все эти обновления.

Аналитические запросы распараллеливаются по всем доступным ядрам с использованием массивно-параллельной версии известного алгоритма соединения сортировка-слияние [96].

Позже разработчики HyPer изменили эту простую архитектуру, чтобы обеспечить более высокую пропускную способность для транзакционных рабочих нагрузок. В частности, они внедрили в систему собственную версию многоверсионного контроля параллелизма [96], что значительно усложняет систему, но повышает ее транзакционную производительность.

4.5. Преимущества, недостатки и компромиссы

Как видно из приведенного выше обзора, существует множество способов реализовать функциональность HTAP и обеспечить аналитику в реальном времени. Общие принципы заключаются в следующем:

• предоставлять свежие данные для анализа, объединяя транзакционную и аналитическую обработку в одной системе баз данных;

• сделать обработку транзакций максимально быстрой, чтобы удовлетворить потребности транзакционных клиентов и повысить актуальность данных;

• максимально быстро сделать оперативные данные доступными для анализа, чтобы удовлетворить первое требование аналитики в реальном времени – свежесть данных;

• сделать выполнение аналитических запросов максимально быстрым, чтобы удовлетворить второе требование аналитики в реальном времени – быстрый анализ данных.

Обычный подход для достижения этих целей состоит в том, чтобы хранить всю базу данных в основной памяти. Этот выбор позволяет практически избежать любого ввода-вывода с внешними устройствами хранения (постоянное хранилище используется только для обеспечения долговечности результатов транзакций) и использовать прямые указатели на объекты базы данных внутри базы данных. Все внутренние структуры данных проектируются с учетом наличия аппаратного кэша. Для дальнейшего повышения производительности используются передовые методы оптимизации запросов, компиляция запросов в собственный машинный код, многоядерное распараллеливание, инструкции SIMD и аппаратные ускорители, такие как GPU и FPGA.

Общее *преимущество* подхода HTAP заключается в том, что аналитикам требуется только одна СУБД, чтобы получить возможности быстрого анализа свежих операционных данных.

Главный *недостаток* заключается в том, что практически невозможно обеспечить в одной системе наилучшие характеристики транзакционных и аналитических СУБД.

Обычно одна СУБД поддерживает два хранилища данных – по строкам и по столбцам. Этот подход представляет собой *компромисс* между возможностью быстрой обработки транзакций (хранилище по строкам) и быстрой обработкой аналитических запросов (хранилище по столбцам), с одной стороны, и необходимостью преобразования данных из построчного формата в поколоночный формат до того, как данные станут доступны для аналитики, с другой стороны.

Чтобы избежать высокого уровня конкуренции между параллельно выполняемыми транзакциями, а также между транзакциями и параллельно выполняемыми аналитическими запросами, обычно используется какой-либо многоверсионный контроль параллелизма, часто оптимистичный. Все внутренние объекты базы данных, такие как индексы, распределители памяти и т.д., обрабатываются без блокировок.

Однако хранение базы данных полностью в памяти также является *компромиссом* между высокой производительностью СУБД и ограниченным размером базы данных. Чтобы снять это ограничение, крупные поставщики любят комбинировать хранилища в памяти и на диске. Утверждают, что после этого производительность СУБД не падает (непонятно почему), но в любом случае архитектура системы значительно усложняется.

Имеются некоторые попытки предоставить функции HTAP и аналитики в реальном времени в массивно-параллельных архитектурах без общих ресурсов. Мы считаем, что такая цель является труднодостижимой. Во-первых, даже если каждый узел такой системы будет хранить базу данных целиком в памяти, неизбежные сетевые коммуникации значительно снизят производительность системы. Во-вторых, как транзакционные, так и аналитические СУБД без совместного использования полагаются на подходящее разделение данных между узлами системы. Однако цели партиционирования различны для транзакционных и аналитических СУБД. Транзакционная СУБД пытается разделить базу данных, чтобы минимизировать количество распределенных транзакций. Аналитическая СУБД при партиционировании базы данных старается минимизировать количество распределенных соединений. Сомнительно, чтобы можно было достичь обеих целей в одной системе.

Наконец, значительное число поставщиков in-memory HTAP-СУБД уже используют доступную в настоящее время энергонезависимую основную память (Non-Volatile Memory, NVM) в своих продуктах и/или планируют использовать ее в буду-

щем [98–100]. Эти продукты демонстрируют достаточно широкий спектр вариантов использования NVM от простого расширения памяти с помощью NVM до наиболее перспективных одноуровневых архитектур хранения. Однако последний вариант использования в настоящее время (частично) реализован только в исследовательских проектах [101, 102].

Причина, вероятно, в том, что в настоящее время на рынке доступен только один вид NVM – Intel Optane на базе технологии PCM. Эти модули DIMM на основе NVM имеют довольно высокую задержку и могут не заменить DRAM. Мы считаем, что широко ожидаемое новое поколение NVM (вероятно, основанное на STT) значительно ускорит внедрение NVM в HTAP и (чисто транзакционных) СУБД.

5. ЗАКЛЮЧЕНИЕ

За последние два десятилетия технологии баз данных сделали гигантский скачок вперед. Одним из наиболее заметных достижений является новая возможность анализа данных в реальном времени. Однако этот термин является относительно новым и обычно применяется к программным средам с разными целями и реальными возможностями.

В этой статье мы рассмотрели три подхода к организации управления данными – потоковая обработка, облачное хранилище данных и гибридная транзакционная/аналитическая обработка – каждый из которых претендует на обеспечение аналитики в реальном времени. Мы показали общие архитектурные принципы каждого подхода, привели примеры соответствующих программных продуктов, а также кратко описали их основные преимущества, ограничения и компромиссы.

БЛАГОДАРНОСТИ

Статья подготовлена по материалам доклада на Седьмой Международной конференции “Актуальные проблемы системной и программной инженерии” (АПСПИ 2021).

СПИСОК ЛИТЕРАТУРЫ

1. *William H. Inmon*. Building the Data Warehouse. John Wiley & Sons, 1992. 312 p.
2. *Ralph Kimball*. The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses. Wiley, 1996. 374 p.
3. Information Technology. Gartner Glossary. Real-time Analytics. Available at https://www.gartner.com/en/information-technology/glossary/real-time-analytics, accessed: 06/16/2021
4. *Arun Kejariwal, Sanjeev Kulkarni, Karthik Ramasamy*. Real Time Analytics: Algorithms and Systems. Extended version of VLDB’15 tutorial proposal. arXiv:1708.02621, 2017. 7 p.
5. *Zoran Milosevic, Weisi Chen, Andrew Berry, Fethi A. Rabhi*. Real-Time Analytics. In Big Data: Principles and Paradigms, Morgan Kaufmann, 2016. P. 39–61.
6. *Fatma Özcan, Yuanyuan Tian, Pınar Tözün*. Hybrid Transactional/Analytical Processing: A Survey. Proceedings of the 2017 ACM International Conference on Management of Data, 2017. P. 1771–1775.
7. *Кузнецов С.Д., Велихов П.Е., Фу Ц.* Аналитика в реальном времени, гибридная транзакционная/аналитическая обработка, управление данными в основной памяти и энергонезависимая память. Труды ИСП РАН. 2021. Т. 33. Вып. 3. С. 171–198. https://doi.org/10.15514/ISPRAS–2021–33(3)–13 / Sergey D. Kuznetsov, Pavel E. Velikhov, and Qiang Fu. Real-time analytics, hybrid transactional/analytical processing, in-memory data management, and non-volatile memory. Proceedings of the Ivannikov ISPRAS Open Conference, 2020. P. 78–90.
8. *Monika Rauch Henzinger, Prabhakar Raghavan, and Sridar Rajagopalan*. Computing on data streams. SRC Technical Note 1998-11, May 26, 1998. 16 p.
9. The “Stream Team” Page. Available at http://infolab.stanford.edu/sdt/, accessed 07.07.2021.
10. Special Issue on Data Stream Processing. IEEE Bulletin of the Technical Committee on Data Engineering. 2003. V. 26. № 1.
11. *Stan Zdonik, Michael Stonebraker et al.* The Aurora and Medusa Projects. IEEE Bulletin of the Technical Committee on Data Engineering. 2003 V. 26. № 1. P. 3–10.
12. *Sailesh Krishnamurthy, Sirish Chandrasekaran et al.* TelegraphCQ: An Architectural Status Report. IEEE Bulletin of the Technical Committee on Data Engineering. 2003. V. 26. № 1. P. 11–18.
13. *Arasu A., Babcock B. et al.* STREAM: The Stanford Stream Data Manager. IEEE Bulletin of the Technical Committee on Data Engineering. 2003. V. 26. № 1. P. 19–26.
14. *Douglas Terry, David Goldberg, David Nichols, Brian Oki*. Continuous queries over append-only databases. ACM SIGMOD Record. 1992. V. 21. Iss. 2. P. 321–330.
15. *Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang.* NiagaraCQ: A Scalable Continuous Query System for Internet Databases. ACM SIGMOD Record. 2000. V. 29. Iss. 2. P. 379–390.
16. *Sirish Chandrasekaran, Owen Cooper et al.* TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. Proceedings of the 2003 CIDR Conference, 2003. 12 p.
17. *Johannes Gehrke, Flip Korn, Divesh Srivastava*. On Computing Correlated Aggregates Over Continual Data Streams. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. 2001. P. 13–24.
18. *Arvind Arasu, Brian Babcock et al.* STREAM: The Stanford Data Stream Management System. Technical Report. Stanford InfoLab, 2004. 21 p. Later appeared as a chapter in Data Stream Management. Processing High-Speed Data Streams. Springer, 2016. P. 317–336.

19. *Arvind Arasu, Shivnath Babu, Jennifer Widom*. CQL: A Language for Continuous Queries over Streams and Relations. Lecture Notes in Computer Science. 2003. V. 2921. P. 1–19.
20. *Daniel J. Abadi*. Don Carney, et al. Aurora: a new model and architecture for data stream management. The International Journal on Very Large Data Bases. 2003. V. 12. Iss. 2. P. 120–139.
21. *Uğur Çetintemel, Daniel Abadi*. The Aurora and Borealis Stream Processing Engines. A chapter in Data Stream Management. Processing High-Speed Data Streams. Springer, 2016. P. 337–359.
22. *Daniel J. Abadi, Yanif Ahmad et al.* The Design of the Borealis Stream Processing Engine. Proceedings of the 2005 CIDR Conference, 2005. P. 277–289.
23. TIBCO StreamBase. Available at https://www.tibco.com/sites/tibco/files/resources/DS-TIBCO-StreamBase-final.pdf, accessed 07/14/2021.
24. StreamSQL Guide. Available at https://docs.tibco.com/pub/sb-lv/2.1.8/doc/html/streamsql/index.html, accessed 07/14/2021.
25. *Namit Jain, Shailendra Mishra et al.* Towards a Streaming SQL Standard. Proceedings of the VLDB Endowment. 2008. V. 1. Iss. 2. P. 1379–1390.
26. *Michael Stonebraker, Uğur Çetintemel, Stan Zdonik*. The 8 requirements of real-time stream processing. ACM SIGMOD Record. 2005. V. 34. Iss. 4. P. 42–47.
27. *Sandra Geisler*. Data Stream Management Systems. In Data Exchange, Integration, and Streams. Dagstuhl Follow-Ups. 2013. V. 5. P. 275–304.
28. Special Issue on Next-Generation Stream Processing. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4.
29. *Martin Kleppmann, Jay Kreps*. Kafka, Samza and the Unix Philosophy of Distributed Data. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4. P. 4–14.
30. *Paris Carbone, Stephan Ewen*. Apache Flink: Stream and Batch Processing in a Single Engine. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4. P. 28–38.
31. *Scott Schneider, Buğra Gedik, Martin Hirzel*. Language Runtime and Optimizations in IBM Streams. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4. P. 61–72.
32. *Andrew Witkowski, Srikanth Bellamkonda et al.* Continuous Queries in Oracle. Proceedings of the 33rd International Conference on Very Large Data Bases, 2007. P. 1173–1184.
33. Oracle Fusion Middleware Understanding Stream Analytics. Available at https://docs.oracle.com/en/middleware/fusion-middleware/osa/18.1/understanding-stream-analytics/understanding-oracle-stream-analytics.pdf, accessed 07/16/2021.
34. *Thomas Vengal*. What is Oracle Stream Analytics? Available at https://blogs.oracle.com/dataintegration/what-is-oracle-stream-analytics, accessed 07/16/2021.
35. IBM Streams. Available at https://www.ibm.com/cloud/streaming-analytics, accessed 07/16/2021.
36. *Alain Biem, Eric Bouillet et al.* IBM InfoSphere Streams for Scalable, Real-Time, Intelligent Transportation Services. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, 2010. P. 1093–1104.
37. *Hirzel M., Andrade H. et al.* IBM Streams Processing Language: Analyzing BigData in motion. IBM Journal of Research and Development. 2013. V. 57. № 3/4. 11 p.
38. *Mohamed Ali, Badrish Chandramouli et al.* Spatio-Temporal Stream Processing in Microsoft StreamInsight. IEEE Bulletin of the Technical Committee on Data Engineering. 2010. V. 33. № 2. P. 69–74.
39. M*ohamed Ali, Badrish Chandramouli* et al. The Extensibility Framework in Microsoft StreamInsight. Proceedings of the IEEE 27th International Conference on Data Engineering, 2011. P. 1242–1253.
40. *Rob Pierry*. StreamInsight – Master Large Data Streams with Microsoft StreamInsight. MSDN Magazine. 2011. V. 26. № 06.
41. What is Microsoft StreamInsight? Available at https://azurecloudai.blog/2013/01/30/what-is-microsoft-streaminsight/, accessed 07/16/2021.
42. Welcome to Azure Stream Analytics. Available at https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction, accessed 07/16/2021.
43. Data Engineering Streaming. Available at https://www.informatica.com/products/big-data/big-data-streaming.html, accessed 07/16/2021.
44. SAS's Event Stream Processing. Available at https://www.sas.com/en_us/software/event-stream-processing.html, accessed 07/16/2021.
45. Apache Kafka. Available at https://kafka.apache.org/, accessed 07/16/2021.
46. Apache Samza. Available at http://samza.apache.org/, accessed 07/16/2021.
47. Apache Kafka Architecture – Kafka Component Overview. Available at https://www.instaclustr.com/apache-kafka-architecture/#, accessed 07/16/2021.
48. Apache ZooKeeper. Available at https://zookeeper.apache.org/, accessed 07/16/2021.
49. Apache Hadoop YARN. Available at https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, accessed 07/16/2021.
50. *Rahul Anand*. What is Apache Samza? Available at https://www.quora.com/What-is-Apache-Samza-1, accessed 07/16/2021.
51. What is Apache Flink? – Architecture. Available at https://flink.apache.org/flink-architecture.html, accessed 07/16/2021.
52. Spark Streaming Programming Guide. Available at https://spark.apache.org/docs/latest/streaming-programming-guide.html, accessed 07/16/2021.
53. Spark API Documentation. Available at https://spark.apache.org/docs/2.4.0/api.html, accessed 07/16/2021.
54. BigQuery. Available at https://cloud.google.com/bigquery, accessed 07/17/2021.
55. A Deep Dive into Google BigQuery Architecture. Available at

https://panoply.io/data-warehouse-guide/bigquery-architecture/, accessed 07/17/2021.

56. *Sergey Melnik, Andrey Gubarev et al.* Dremel: Interactive Analysis of Web-Scale Datasets. Proceedings of the VLDB Endowment. 2010. V. 3. № 1. P. 330–339.
57. *Foto N. Afrati, Dan Delorey et al.* Storing and Querying Tree Structured Records in Dremel. Proceedings of the VLDB Endowment. 2014. V. 7. № 11. P. 1131–1142.
58. *Mosha Pasumansky*. Inside Capacitor, BigQuery's next-generation columnar storage format. Available at https://cloud.google.com/blog/products/bigquery/inside-capacitor-bigquerys-next-generation-columnar-storage-format, accessed 07/17/2021.
59. *Dean Hildebrand, Denis Serenyi*. Colossus under the hood: a peek into Google's scalable storage system. Available at https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system, accessed 07/17/2021.
60. *Abhishek Verma, Luis Pedrosa et al*. Large-scale cluster management at Google with Borg. Proceedings of the Tenth European Conference on Computer Systems, 2015. P. 1–17.
61. *Arjun Singh, Joon Ong, et al.* Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. ACM SIGCOMM Computer Communication Review, 2015. P. 183–197.
62. Amazon Redshift and PostgreSQL. Available at https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-and-postgres-sql.html, accessed 07/17/2021.
63. Data warehouse system architecture. Available at https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html, accessed 07/17/2021.
64. *Anurag Gupta, Deepak Agarwal et al*. Amazon Redshift and the Case for Simpler Data Warehouses. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015. P. 1917–1923.
65. The Microsoft Modern Data Warehouse. White paper, 2016. Available at http://download.microsoft.com/download/C/2/D/C2D2D5FA-768A-49AD-8957-1A434C6C8126/Microsoft_Modern_Data_Warehouse_white_paper.pdf, accessed 07/18/2021.
66. Azure Synapse SQL architecture. Available at https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/overview-architecture, accessed 07/18/2021.
67. What is Azure Synapse Analytics? Available at https://docs.microsoft.com/en-us/azure/synapse-analytics/overview-what-is, accessed 07/18/2021.
68. Use transactions in a SQL pool in Azure Synapse. Available at https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/synapse-analytics/sql-data-warehouse/sql-data-warehouse-develop-transactions.md, accessed 07/18/2021.
69. Ashish Motivala, Jiaqi Yan. The Snowflake Elastic Data Warehouse, SIGMOD 2016 and beyond. Available at https://15721.courses.cs.cmu.edu/spring2018/slides/25-snowflake.pdf, accessed 07/18/2021.
70. *Benoit Dageville, Thierry Cruanes et al*. The Snowflake Elastic Data Warehouse. Proceedings of the 2016 International Conference on Management of Data, 2016. P. 215–226.
71. *Anastassia Ailamaki, David J. DeWitt et al*. Weaving Relations for Cache Performance. Proceedings of the 27th International Conference on Very Large Data Bases, September 2001. P. 169–180.
72. *David Karger, Eric Lehman et al*. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997. P. 654–663.
73. *Goetz Graefe*. The Cascades Framework for Query Optimization. IEEE Bulletin of the Technical Committee on Data Engineering. V. 18. № 3, 1995. P. 19–29.
74. *Franz Faerber, Alfons Kemper et al.* Main Memory Database Systems. Foundations and Trends in Databases. 2016. V. 8. № 1–2. P. 1–130.
75. *Frederik Transier, Peter Sanders*. Engineering basic algorithms of an in-memory text search engine. ACM Transactions on Information Systems, 2010, Article No. 2.
76. *J. Andrew Ross*. SAP NetWeaver BI Accelerator. SAP PRESS, 2008. 260 p.
77. *Sang K. Cha and Changbin Song*. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004. P. 1033–1044.
78. *André Bögelsack, Stephan Gradl, Manuel Mayer, Helmut Krcmar*. SAP MaxDB Administration. SAP PRESS, 2009. 326 p.
79. *Franz Faerber, Norman May et al.* The SAP HANA Database – An Architecture Overview. IEEE Bulletin of the Technical Committee on Data Engineering. 2012. V. 35. № 1. P. 28–33.
80. *Per-Åke Larson, Cipri Clinciu et al*. SQL Server Column Store Indexes. Proceedings of the ACM SIGMOD International Conference on Management of data, 2011. P. 1177–1184.
81. *Per-Åke Larson, Mike Zwilling, Kevin Farlee*. The Hekaton Memory-Optimized OLTP Engine. Bulletin of the Technical Committee on Data Engineering. 2013. V. 36. № 2. P. 34–40.
82. *Per-Åke Larson, Adrian Birka et al.* Real-Time Analytical Processing with SQL Server. Proceedings of the VLDB Endowment. 2015. V. 8. № 12. P. 1740–1751.
83. *Ahmed Eldawy, Justin Levandoski, Per-Åke Larson*. Trekking Through Siberia: Managing Cold Data in a Memory-Optimized Database. Proceedings of the VLDB Endowment. 2014. V. 7. № 11. P. 931–942.
84. *Tirthankar Lahiri, Marie-Anne Neimat, Steve Folkman*. Oracle TimesTen: An In-Memory Database for Enterprise Applications. Bulletin of the Technical Committee on Data Engineering. 2013. V. 36. № 2. P. 6–13.
85. *Sherry Listgarten and Marie-Anne Neimat*. Modelling Costs for a MM-DBMS. In Proceedings of the International Workshop on Real-Time Databases, Issues and Applications (RTDB), 1996. P. 72–78.
86. *Tirthankar Lahiri, Shasank Chavan et al*. Oracle Database In-Memory: A dual format in-memory database. 2015 IEEE 31st International Conference on Data Engineering, Seoul, 2015. P. 1253–1258.

87. *Niloy Mukherjee, Shasank Chavan et al.* Distributed Architecture of Oracle Database In-memory. Proceedings of the VLDB Endowment. 2015. V. 8. № 12. P. 1630–1641.
88. *Shasank Chavan, Gurmeet Goindi*. Oracle Database In-Memory on Exadata: A Potent Combination. Oracle OpenWorld 2018. Available at https://www.oracle.com/technetwork/database/exadata/pro4016-exadataandinmemory-5187037.pdf, accessed 07/18/2021.
89. *Ronald Barber, Peter Bendel et al.* Business Analytics in (a) Blink. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 2012. V. 35. № 1. P. 9–14.
90. IBM Informix Warehouse Accelerator. Technical white paper. URL: https://www.iiug.org/library/ids_12/IWA%20White%20Paper-2013-03-21.pdf, accessed 07/18/2021.
91. *Vijayshankar Raman, Gopi Attaluri et al.* DB2 with BLU Acceleration: So Much More than Just a Column Store. Proceedings of the VLDB Endowment. 2013. V. 6. № 11. P. 1080–1091.
92. *Whei-Jen Chen, Brigitte Bläser et al.* Architecting and Deploying DB2 with BLU Acceleration. IBM Redbooks, 2014. 420 p.
93. Faster analytics with Hyper. Available at https://www.tableau.com/products/new-features/hyper, accessed 07/18/2021.
94. *Alfons Kemper and Thomas Neumann*. HyPer – Hybrid OLTP&OLAP High Performance Database System. Technical Report, TUM-I1010, Munich Technical University, 2010. 29 p.
95. *Alfons Kemper, Thomas Neumann et al.* Transaction Processing in the Hybrid OLTP&OLAP Main-Memory Database System HyPer. Bulletin of the Technical Committee on Data Engineering. 2013. V. 36. № 2. P. 41–47.
96. *Martina-Cezara Albutiu, Alfons Kemper, Thomas Neumann*. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. Proceedings of the VLDB Endowment. 2012. V. 5. № 10. P. 1064–1075.
97. *Thomas Neumann, Tobias Mühlbauer, Alfons Kemper*. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. Proceedings of the ACM SIGMOD International Conference on Management of data, 2015. P. 677–689.
98. *Mihnea Andrei, Christian Lemke, et al.* SAP HANA Adoption of Non-Volatile Memory. Proceedings of the VLDB Endowment. 2017. V. 10. № 12. P. 1754–1765.
99. *Bob Dorr*. How It Works (It Just Runs Faster): Non-Volatile Memory SQL Server Tail of Log Caching on NVDIMM. Available at https://docs.microsoft.com/ru-ru/archive/blogs/bobsql/how-it-works-it-just-runs-faster-non-volatile-memory-sql-server-tail-of-log-caching-on-nvdimm, accessed 07/18/2021.
100. Oracle Database 20c. Database Administrator's Guide. Using Persistent Memory Database. Available at https://docs.oracle.com/en/database/oracle/oracle-database/20/admin/index.html, accessed 07/18/2021.
101. *Joy Arulraj, Andrew Pavlo*. Non-Volatile Memory Database Management Systems. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019. 192 p.
102. *Ismail Oukid*. Architectural Principles for Database Systems on Storage-Class Memory. Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2019. P. 477–486.

ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 32–37

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 681.3.06, 004.94, 519.63

ИССЛЕДОВАНИЯ РАЗНОСТНЫХ СХЕМ ДЛЯ ДВУМЕРНЫХ УРАВНЕНИЙ НАВЬЕ–СТОКСА АЛГОРИТМАМИ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

© 2023 г. Ю. А. Блинков[a,b,c,*] (ORCID: 0000-0001-7340-0919), А. Ю. Ребрина[d,]**

[a] *Саратовский национальный исследовательский государственный университет имени Н.Г. Чернышевского, 410012 Саратов, ул. Астраханская, д. 83, Россия*

[b] *Российский университет дружбы народов (РУДН), 117198 Москва, ул. Миклухо-Маклая, д. 6, Россия*

[c] *Объединенный институт ядерных исследований, 141980 Дубна, ул. Жолио-Кюри, д. 6, Россия*

[d] *Саратовский государственный технический университет имени Гагарина Ю.А., 410054 Саратов, ул. Политехническая, д. 77, Россия*

**E-mail: blinkovua@info.sgu.ru*

***E-mail: anrebrina@yandex.ru*

Поступила в редакцию 01.07.2022 г.
После доработки 23.08.2022 г.
Принята к публикации 07.09.2022 г.

На основе алгоритма построения базисов Грёбнера рассмотрен класс совместных разностных схем для уравнений Навье–Стокса несжимаемой жидкости в физических переменных и их дифференциальные приближения. Представлены результаты исследования первых дифференциальных приближений этих схем, выполненные авторскими программами, реализованными в системе компьютерной алгебре SymPy. Для рассмотренных разностных схем показана квадратичная зависимость погрешности рассмотренных разностных схем для больших чисел Рейнольдса и обратная пропорциональная для ползущих течений.

DOI: 10.31857/S0132347423010028, **EDN:** GPETXS

1. ВВЕДЕНИЕ

Найти точные решения систем уравнений в частных производных удается лишь в весьма редких случаях, поэтому обычно такие системы в механике и физике приходится решать численно, то есть путем их замены дискретными аналогами. Наиболее известными методами дискретизации и численного решения являются методы конечных элементов, конечных объемов и конечных разностей. Последний метод исторически является первым [1] и основан на замене дифференциальных уравнений разностными, определенными на выбранной сетке решений. Чтобы построить численное решение, разработанное конечно-разностное приближение уравнений в частных производных дополняется соответствующей дискретизацией начальных или/и граничных условий. При этом близость и сходство численного и точного решений системы уравнений в частных производных в решающей степени зависит от качества разностной схемы.

В работах [2, 3] разработан и реализован средствами компьютерной алгебры Maple алгоритм построения консервативных разностных схем. После задания искомой системы в виде условий связи разностных функций и их разностных производных в виде интегральных соотношений и выбора соответствующего упорядочения путем вычисления разностного базиса Грёбнера, который адаптирует к разностному случаю известный алгоритм для исследования полиномиальных систем [4]. В результате применения алгоритма построения базиса Грёбнера разностного идеала для дифференциальных уравнений с производными выше первого порядка строится разностная схема в виде соотношения только на сами функции. При это проходит проверка на совместность полученной разностной схемы и определяется ее размерность на пространство решений.

Одним из алгоритмических подходов для исследования разностных схем является построение первого дифференциального приближения [5]. На его основе и с использованием базисов Грёбнера построены дифференциальные приближения для уравнений эволюционного типа [6] и для систем уравнений [7]. В работе [8] рассмотрены вопросы вычисления первого дифференциального приближения в системах компьютерной

алгебры. Если разностная схема не совместна или имеет меньшее пространство решений по сравнению с исходной системой дифференциальных уравнений, то при построении первого дифференциального приближения будут получены *лишние уравнения* и вычисления можно остановить. Эта проверка значительно проще, чем проверка совместности в разностном случае как по требуемому времени, так и по памяти и легко может быть реализована обычными средствами систем компьютерной алгебры.

В данной работе предлагается в рамках универсального алгоритма для коммутативной алгебры средствами компьютерной алгебры проверить совместность исходной системы дифференциальных уравнений и аппроксимирующей ее разностной схемой, прямо и в рамках первого дифференциального приближения. Хотя алгоритм построения базисов Грёбнера и встроен в большинство систем компьютерной алгебры, он как правило имеет ограничения на работу с дифференциальными уравнениями, разностными схемами и *формально бесконечными* рядами Тейлора при построении первого дифференциального приближения [9–11].

В качестве модельной системы для реализованных в системе компьютерной алгебры SymPy будет использована система двухмерных уравнений Навье–Стокса несжимаемой жидкости. В работах [12–17] были построены и численно исследованы разностные схемы на некоторых точных решениях.

2. НЕПРЕРЫВНЫЙ СЛУЧАЙ

Запишем систему двухмерных уравнений Навье–Стокса несжимаемой жидкости в декартовой системе координат

$$\begin{aligned} F^1 &: u_x + v_y = 0,\\ F^2 &: u_t + uu_x + vu_y + p_x - \frac{1}{\mathrm{Re}}\Delta u = 0,\\ F^3 &: v_t + uv_x + vv_y + p_y - \frac{1}{\mathrm{Re}}\Delta v = 0. \end{aligned} \tag{2.1}$$

Пусть $p \succ u \succ v$ и $t \succ x \succ y$. Введем допустимое упорядочение по *POT* (позиция старше терма) таким способом, что любая производная по t старше любой другой производной.

Тогда лидирующими членами для системы (2.1) будут соответственно u_x, u_t, v_t и можно будет построить следующий S-полином:

$$\begin{aligned} &(F^2_x - F^1_t) + F^3_y + \frac{1}{\mathrm{Re}}\Delta F^1 =\\ &= \left(uu_x + vu_y\right)_x + \left(uv_x + vv_y\right)_y + \Delta p. \end{aligned} \tag{2.2}$$

В результате получим *инволютивную систему*:

$$\begin{aligned} F^1 &: u_x + v_y = 0,\\ F^2 &: u_t + uu_x + vu_y + p_x - \frac{1}{\mathrm{Re}}\Delta u = 0,\\ F^3 &: v_t + uv_x + vv_y + p_y - \frac{1}{\mathrm{Re}}\Delta v = 0,\\ F^4 &: \left(uu_x + vu_y\right)_x + \left(uv_x + vv_y\right)_y + \Delta p = 0, \end{aligned} \tag{2.3}$$

где лидирующими членами будут соответственно u_x, u_t, v_t, p_{xx}. Эту систему можно переписать в виде законов сохранения:

$$\begin{aligned} F^1 &: \mathrm{div}(u,v) = 0,\\ F^2 &: u_t + \mathrm{div}\left(u^2 + p - \frac{1}{\mathrm{Re}}u_x, vu - \frac{1}{\mathrm{Re}}u_y\right) = 0,\\ F^3 &: v_t + \mathrm{div}\left(uv - \frac{1}{\mathrm{Re}}v_x, v^2 + p - \frac{1}{\mathrm{Re}}v_y\right) = 0,\\ F^4 &: \mathrm{div}\left(uu_x + vu_y + p_x, uv_x + vv_y + p_y\right) = 0. \end{aligned} \tag{2.4}$$

Понятие *инволютивности* было введено около ста лет тому назад Картаном (Cartan) [18] при исследовании уравнений типа Пфаффа в полных дифференциалах. Инволютивная система содержит в себе все условия интегрируемости и дифференциальные продолжения системы не дают новых условий совместности. Пополнение системы ее условиями интегрируемости называется *замыканием*. Подход Картана был обобщен Келером (Kähler) [19] к произвольным системам внешних дифференциальных уравнений. Рикье (Riquier) [20], для исследования решений дифференциальных уравнений в виде формальных рядов, предложил полное упорядочение для частных производных. Используя упорядочение он выделил часть производных, называемых *главными*, относительно которых можно разрешить систему ДУЧП. Оставшиеся производные, называемые *параметрическими*, задают произвол в решении и влияют на постановку начальных условий. В результате Рикье была построена теория содержащая теорему Коши–Ковалевской как частный случай.

3. КОНЕЧНО-РАЗНОСТНАЯ АППРОКСИМАЦИЯ

При переходе от непрерывного случая к его конечно разностной аппроксимации на равномерных сетках возникают три проблемы:

1. Для дискретной производной D не выполняется правило Лейбница: $\mathrm{D}(ab) \neq \mathrm{D}(a)b + a\mathrm{D}(b)$.

2. Если в непрерывном случае допустимое упорядочение было связанно с производными, то в дискретном случае оно привязано к дискретным координатам n, j, k (им соответствуют t, x, y) самой функции. Другими словами, в непрерывном

случае упорядочение допустимо относительно производных, а в дискретном допустимо относительно сдвигов.

3. Сдвиги могут быть вперед и могут быть назад, а базисы Грёбнера могут работать только с положительными, поэтому желательно сдвинуть первоначальные разностные соотношения вперед для упрощения итогового базиса Грёбнера.

В статье [12], чтобы обойти пункт 2 для получения конечно разностной аппроксимации системы (2.1) использовали дискретные операторы, которые обладали коммутативностью и позволяли применить теорию базисов Грёбнера, а в статье [13] для вычисления разностных базисов Грёбнера использовали их специализированную реализацию средствами компьютерной алгебры.

3.1. MAC схема

Метод маркеров и частиц (MAC) [21] представляет очень известный вариант явной разностной схемы на разнесенной сетке с использованием осреднений для компонент скорости:

$$\frac{u^n_{j+1/2,k}-u^n_{j-1/2,k}}{h}+\frac{v^n_{j,k+1/2}-v^n_{j,k-1/2}}{h}=0,$$
$$u^n_{j+1/2,k}=F^n_{j+1/2,k}-\frac{\tau}{h}(p^{n+1}_{j+1,k}-p^{n+1}_{j,k}),$$

$$v^n_{j,k+1/2}=G^n_{j,k+1/2}-\frac{\tau}{h}(p^{n+1}_{j,k+1}-p^{n+1}_{j,k}),$$
$$\frac{p^{n+1}_{j+1,k}-2p^{n+1}_{j,k}+p^{n+1}_{j-1,k}}{h^2}+\frac{p^{n+1}_{j,k+1}-2p^{n+1}_{j,k}+p^{n+1}_{j,k-1}}{h^2}= \quad (3.1)$$
$$=\frac{1}{\tau}\left(\frac{F^n_{j+1/2,k}-F^n_{j-1/2,k}}{h}+\frac{G^n_{j,k+1/2}-G^n_{j,k-1/2}}{h}\right),$$

$$F^n_{j+1/2,k}=\ldots,$$
$$G^n_{j,k+1/2}=\ldots.$$

Уравнение Пуассона решается на каждом шаге по времени и после того, как определено значение $p^{n+1}_{j,k}$, подстановка этого значения в остальные уравнения системы (3.1) позволяет определить компоненты скорости на временном слое $n+1$.

Также стоит отметить, что фактически в статье [21] осуществлена проверка совместности системы (3.1) путем построения s-полинома. Как дословно пишут авторы, что дискретная производная по времени от уравнения неразрывности равна нулю. Удивительно, что одновременно, в том же году, была построена теория базисов Грёбнера [22] Бруно Бухбергером.

3.2. Обычная разностная схема

Поскольку при построении базиса Грёбнера нельзя работать с отрицательными сдвигами, рассмотрим разностные соотношения в точке $j+l$, $k+l$, где $l>0$ достаточно большое число, чтобы избежать отрицательных сдвигов. Запишем соотношения для уравнения неразрывности, например, в точке $j+l+m$, $k+l$ и $m>0$, чтобы получить более компактный базис Грёбнера. В результате при любом выборе разностных производных D_1, D_2, D_3, Δ и осреднения по времени I_t (для построения явной, полунеявной или чисто неявной схемы) следующая система будет представлять собой разностный базис Грёбнера

$$F^1: D_{1x}u^n_{j+l+m,k+l}+D_{1y}v^n_{j+l+m,k+l}=0,$$
$$F^2: \frac{u^{n+1}_{j+l,k+l}-u^n_{j+l,k+l}}{\tau}+I_t\Big(u^n_{j+l,k+l}D_{2x}u^n_{j+l,k+l}+$$
$$+v^n_{j+l,k+l}D_{2y}u^n_{j+l,k+l}+D_{3x}p^n_{j+l,k+l}-\frac{1}{\mathrm{Re}}\Delta u^n_{j+l,k+l}\Big)=0,$$

$$F^3: \frac{v^{n+1}_{j+l,k+l}-v^n_{j+l,k+l}}{\tau}+I_t\Big(u^n_{j+l,k+l}D_{2x}v^n_{j+l,k+l}+$$
$$+v^n_{j+l,k+l}D_{2y}v^n_{j+l,k+l}+ \quad (3.2)$$
$$+D_{3y}p^n_{j+l,k+l}-\frac{1}{\mathrm{Re}}\Delta v^n_{j+l,k+l}\Big)=0,$$

$$F^4: D_{1x}(u^n_{j+l,k+l}D_{2x}u^n_{j+l,k+l}+v^n_{j+l,k+l}D_{2y}u^n_{j+l,k+l})+$$
$$+D_{1y}(u^n_{j+l,k+l}D_{2x}v^n_{j+l,k+l}+v^n_{j+l,k+l}D_{2y}v^n_{j+l,k+l})+$$
$$+(D_{1x}D_{3x}p^n_{j+l,k+l}+D_{1y}D_{3y}p^n_{j+l,k+l})=0.$$

Для проверки совместности системы (3.2) построим следующий S-полином, аналогичный (2.2) для непрерывного случая:

$$D_{1x}F^2+D_{1y}F^3-\frac{(D_{1x}u^{n+1}_{j+l,k+l}+D_{1y}v^{n+1}_{j+l,k+l})}{\tau}- \quad (3.3)$$
$$-\frac{(D_{1x}u^n_{j+l,k+l}+D_{1y}v^n_{j+l,k+l})}{\tau}+I_t\left(\frac{1}{\mathrm{Re}}\Delta F^1\right)-I_t(F^4).$$

Система (3.2) совместна, если этот многочлен тождественно равен нулю. В отличии от схемы (3.1), схема (3.2) используя уравнения F^2, F^3 строит значение для скоростей на следующем временном слое. Уравнение неразрывности F^1 выполнено автоматически. Уравнения F^4 служит только для определения давления для следующего временного слоя.

3.3. Разностная схема в дивергентном виде

Аналогично, при любом выборе разностных производных D_1, D_2, D_3, Δ и осреднения по време-

ни I_t следующая система будет представлять собой разностный базис Грёбнера

$$F^1 : D_{1x}u^n_{j+l+m,k+l} + D_{1y}v^n_{j+l+m,k+l} = 0,$$

$$F^2 : \frac{u^{n+1}_{j+l,k+l} - u^n_{j+l,k+l}}{\tau} + I_t\left(D_{2x}(u^n_{j+l,k+l}{}^2) + D_{2y}(v^n_{j+l,k+l}u^n_{j+l,k+l}) + D_{3x}p^n_{j+l,k+l} - \frac{1}{\mathrm{Re}}\Delta u^n_{j+l,k+l}\right) = 0,$$

$$F^3 : \frac{v^{n+1}_{j+l,k+l} - v^n_{j+l,k+l}}{\tau} + I_t\left(D_{2x}(u^n_{j+l,k+l}v^n_{j+l,k+l}) + D_{2y}(v^n_{j+l,k+l}{}^2) + D_{3y}p^n_{j+l,k+l} - \frac{1}{\mathrm{Re}}\Delta v^n_{j+l,k+l}\right) = 0, \qquad (3.4)$$

$$F^4 : D_{1x}(D_{2x}(u^n_{j+l,k+l}{}^2) + D_{2y}(v^n_{j+l,k+l}u^n_{j+l,k+l})) + D_{1y}(D_{2x}(u^n_{j+l,k+l}v^n_{j+l,k+l}) + D_{2y}(v^n_{j+l,k+l}{}^2)) + (D_{1x}D_{3x}p^n_{j+l,k+l} + D_{1y}D_{3y}p^n_{j+l,k+l}) = 0.$$

Для проверки совместности системы (3.4) можно построить S-полином, в точности совпадающий с (3.3).

Аналогично схеме (3.2) для схемы (3.4) используя уравнения F^2, F^3 строится значение для скоростей на следующем временном слое. Уравнение неразрывности F^1 выполнено автоматически, а уравнения F^4 служит только для определения давления для слоя $n+1$.

4. ДИФФЕРЕНЦИАЛЬНЫЕ ПРИБЛИЖЕНИЯ

Построение первого дифференциального приближения было сделано с оригинальными программами авторов статьи в системе алгебры SymPy и может быть скачана `https://github.com/blinkovua/sharing-blinkov/blob/master/NavierStokes2D.ipynb`.

Рассмотрим сначала аппроксимацию уравнений (3.1) до первых ненулевых членов:

$$F^1 : u_x + v_y + \tau(u_{tx}/2 + \ldots) + h^2(u_{xxx}/24 + \ldots),$$

$$F^2 : p_x + 2uu_x + uv_y + u_t + u_yv + (-u_{xx} - u_{yy})/\mathrm{Re} + \tau(p_{tx} + \ldots) + h^2((p_{xxx}/6 + \ldots) - (5u_{xxxx}/24 + \ldots)/\mathrm{Re}),$$

$$F^3 : p_y + uv_x + u_xv + 2vv_y + v_t + (-v_{xx} - v_{yy})/\mathrm{Re} + \tau(p_{ty} + \ldots) + h^2((p_{yyy}/6 + \ldots) - (v_{xxxx}/12 + \ldots)/\mathrm{Re}),$$

$$F^4 : p_{xx} + p_{yy} + 2uu_{xx} + 2uv_{xy} + 2u_{xy}v + 2u_x^2 + 2u_xv_y + 2u_yv_x + 2vv_{yy} + 2v_y^2 + (-u_{xxx} - u_{xyy} - v_{xxy} - v_{yyy})/\mathrm{Re} + \tau(p_{txx} + \ldots) + h^2(p_{xxxx}/12 + \ldots).$$

Некоторая несимметрия в разложениях для F^2, F^3 связана с порядком переменных. Преобразовав эти уравнения с учетом исходных уравнений (2.1), получим первое дифференциальное приближение для (2.3):

$$F^1 : u_x + v_y + h^2(\mathrm{Re}\,(-p_{yy}/24 + \ldots)) + v_{yyy}/12),$$

$$F^2 : p_x + 2uu_x + uv_y + u_t + u_yv + (-u_{xx} - u_{yy})/\mathrm{Re} + \tau(p_{tx} + \ldots) + h^2(\mathrm{Re}^2v^2(p_x/12 + \ldots) + \mathrm{Re}(-p_{tx}/12 + \ldots) + (p_{xyy}/12 + \ldots) - v_{xyyy}/(6\mathrm{Re})),$$

$$F^3 : p_y + uv_x + u_xv + 2vv_y + v_t + (-v_{xx} - v_{yy})/\mathrm{Re} + \tau((p_{ty} + \ldots)) + h^2(\mathrm{Re}^2u^2(p_y/12 + \ldots) + \mathrm{Re}(p_{ty}/12 + \ldots) + (5p_{yyy}/24 + \ldots) - v_{yyyy}/(6\mathrm{Re}))$$

$$F^4 : p_{xx} + p_{yy} + 2uu_{xx} + 2uv_{xy} + 2u_{xy}v + 2u_x^2 + 2u_xv_y + 2u_yv_x + 2vv_{yy} + 2v_y^2 + (-u_{xxx} - u_{xyy} - v_{xxy} - v_{yyy})/\mathrm{Re} + \tau(-2u_{ty}v_x + \ldots) + h^2(\mathrm{Re}^2(19p_xvv_x/24 + \ldots) - \mathrm{Re}(-7p_{xyy}u/24 + \ldots)(p_{yyyy}/6\ldots))$$

Можно сделать следующие выводы.

• Для F^1 нет зависимости от τ, а вот зависимость от h^2 имеет зависимость $h^2(\mathrm{Re}(\ldots) + (\ldots))$. Это означает, что для достижения той же точности для F^1 при увеличении числа $\mathrm{Re} \gg 1$ необходимо в квадратичный корень раз уменьшить шаг h.

• Многочлены F^2, F^3, F^4 зависят от τ, но не зависит от Re. Для F^2, F^3 зависимость от h^2 имеет вид $h^2(\mathrm{Re}^2(\ldots) + \mathrm{Re}(\ldots) + (\ldots)) + (\ldots))/\mathrm{Re}$. Для F^4 зависимость от h^2 имеет зависимость $h^2(\mathrm{Re}^2(\ldots) + \mathrm{Re}(\ldots) + (\ldots))$. Поэтому при $\mathrm{Re} \gg 1$ для достижения той же точности для F^2, F^3, F^4 при увеличении числа Re необходимо линейно уменьшить шаг h. В случае же ползущих течений, когда $\mathrm{Re} \ll 1$, необходимо для достижения той же точности для F^4 в квадратичный корень раз уменьшить шаг h.

Для класса разностных схем (3.2), (3.4) рассмотрим только разностные операторы второго порядка точности по пространственным переменным заданные в целых узлах:

$$D_x g = \frac{g^n_{j+1,k} - g^n_{j-1,k}}{2h}, \quad D_y g = \frac{g^n_{j,k+1} - g^n_{j,k-1}}{2h},$$
$$\Delta_1 = \frac{g^n_{j+1,k+1} + g^n_{j+1,k} + g^n_{j,k-1} + g^n_{j-1,k-1} - 4g^n_{j,k}}{h^2} \quad (4.1)$$
$$\Delta_2 = \frac{g^n_{j+2,k+2} + g^n_{j+2,k} + g^n_{j,k-2} + g^n_{j-2,k-2} - 4g^n_{j,k}}{h^2}$$

Последние соотношение в (4.1) на Δ_2 использовано, поскольку шаблон для F^4 получается минимум размером 3×3.

Для получения явных, полунеявных и неявных схем можно использовать следующие операторы

$$It_0 g = g^n_{j,k}, \quad It_1 g = \frac{g^{n+1}_{j,k} + g^n_{j,k}}{2}, \quad It_2 g = g^{n+1}_{j,k}. \quad (4.2)$$

Этим путем из (3.2), (3.4) получены 12 различных разностных схем, для них выполнена проверка равенства нулю S-полинома. Выводы, сделанные из первого дифференциального приближения для (3.1), справедливы и по схемам (3.2), (3.4), за одним исключением: для всех 12 схем для F^4 не зависит от τ. Вероятно, это связано с тем, что для этих схем в отличии от (3.1) для следующего временного слоя сначала вычисляются компоненты скорости u, v, а уже потом определяется давление p.

Несмотря на очень громоздкий вид первые дифференциальные приближения могут быть эффективно проверены на точных решениях.

5. ОЦЕНКА КАЧЕСТВА РАЗНОСТНЫХ СХЕМ НА ТОЧНЫХ РЕШЕНИЯХ

Первое дифференциальное приближение при подстановке в него точного решения позволяет оценить саму схему без ее программирования и проведение вычисленных экспериментов для ее проверки. В двумерном случае известно несколько точных решений системы Навье–Стокса (2.1), описанные в [23] и [24]. Приведем здесь первое из них:

$$u = -e^{-2t/\mathrm{Re}} \cos(x)\sin(y),$$
$$v = e^{-2t/\mathrm{Re}} \sin(x)\cos(y), \quad (5.1)$$
$$p = -e^{-4t/\mathrm{Re}}(\cos(2x) + \cos(2y))/4.$$

Подстановка этого решения в первое дифференциальное приближение схемы МАС (3.1), приводит к следующему. Первое дифференциальное приближение аппроксимации уравнения неразрывности F^1 обращается тождественно в нуль. Первые дифференциальные приближения F^2, F^3 имеет вид

$$\tau(\ldots)/\mathrm{Re}^2 + h^2((\ldots) + (\ldots)/\mathrm{Re}).$$

Подстановка решения (5.1) в первое дифференциальное приближение полученных выше 12 разностных схем дает различные выражения для F^2, F^3, вид которых зависит от выбора осреднения по времени (4.2). Для схемы (3.2):

$$\begin{aligned} It_0 &: \tau(\ldots)/\mathrm{Re}^2 + h^2\left((\ldots) + (\ldots)/\mathrm{Re}\right), \\ It_1 &: h^2\left((\ldots) + (\ldots)/\mathrm{Re}\right), \\ It_2 &: \tau(\ldots)/\mathrm{Re}^2 + h^2((\ldots) + (\ldots)/\mathrm{Re} + \\ &\quad + (\ldots)/\mathrm{Re}^2 + (\ldots)/\mathrm{Re}^3), \end{aligned} \quad (5.2)$$

а для схемы (3.4):

$$\begin{aligned} It_0 &: \tau(\ldots)/\mathrm{Re}^2 + h^2\left((\ldots)/\mathrm{Re}\right), \\ It_1 &: h^2\left(\ldots\right)/\mathrm{Re}, \\ It_2 &: \tau(\ldots)/\mathrm{Re}^2 + h^2((\ldots)/\mathrm{Re} + (\ldots)/\mathrm{Re}^3). \end{aligned} \quad (5.3)$$

Таким образом, лучшее поведение демонстрируют осреднение для полунеявной схемы.

Особо отметим поведение первого дифференциального приближения для F^4 на решении (5.1). Для схемы МАС (3.1):

$$\begin{aligned} &-\tau(4\left(\cos\left(2x\right) + \cos\left(2y\right)\right)e^{-\frac{4t}{\mathrm{Re}}})/\mathrm{Re} + \\ &+ h^2((12\sin^2(x)\sin^2(y) - 8\sin^2(x) - \\ &- 8\sin^2(y) + 5)/4 - \\ &- 16(\cos\left(2x\right) + \cos\left(2y\right))e^{-\frac{4t}{\mathrm{Re}}}/\mathrm{Re}^2). \end{aligned}$$

Для схемы (3.2) при I_t, равным It_0 или It_1, первое дифференциальное приближение для F^4 равно:

$$h^2((\sin^2\left(x\right) + \sin^2\left(y\right) - 1)e^{-\frac{4t}{\mathrm{Re}}}).$$

Для схемы (3.2) с вариантами с It_2 и для всех вариантов (3.4) первое дифференциальное приближение для F^4 равно нулю.

6. ЗАКЛЮЧЕНИЕ

Применение алгоритма базисов Грёбнера позволяет проверить на совместность исходную систему, равно как и аппроксимирующую ее разностную схему в первом дифференциальном приближении, что особенно важно, если задача является вычислительно трудной, как например, схема МАС (3.1). Первое дифференциальное приближение позволяет получить важную информацию о разностной схеме. В частности, для рассмотренных в статье разностных схем, аппроксимирующих плоские уравнения Навье–Стокса, это позволило определить их погрешность в зависимости от числа Рейнольдса Re.

ИСТОЧНИК ФИНАНСИРОВАНИЯ

Работа выполнена при финансовой поддержке РНФ (проект № 20-11-20257).

СПИСОК ЛИТЕРАТУРЫ

1. *Самарский А.А.* Теория разностных схем. 3-е изд., испр. – М. Наука. 1989. 616 с.
2. *Блинков Ю.А., Мозжилкин В.В.* Генерация разностных схем для уравнения Бюргерса построением базисов Грёбнера // Программирование. 2006. № 2. С. 71–74.
3. *Gerdt V.P., Blinkov Yu.A., Mozzhilkin V.V.* Gröbner Bases and Generation of Difference Schemes for Partial Differential Equations // SIGMA. 2006. № 2(051). 26 p.
4. *Buchberger B.* Gröbner bases: an Buchberger algorithmic method in polynomial ideal theory // Recent Trends in Multidimensional System Theory / Ed. by *N.K. Bose*. V. 6. Reidel, Dordrecht, 1985. P. 184–232.
5. *Шокин Ю.И., Яненко Н.Н.* Метод дифференциального приближения. Применение к газовой динамике. Новосибирск: Наука: Сиб. отд-ние, 1985. 364 с.
6. *Блинков Ю.А., Гердт В.П., Маринов К.Б.* Дискретизация квазилинейных эволюционных уравнений методами компьютерной алгебры // Программирование. 2017. № 2. С. 28–34.
7. *Zhang X., Gerdt V.P., Blinkov Y.A.* Algebraic Construction of a Strongly Consistent, Permutationally Symmetric and Conservative Difference Scheme for 3D Steady Stokes Flow // Symmetry. 2019. № 11(269). 15 p.
8. *Блинков А.Ю., Малых М.Д., Севастьянов Л.А.* О дифференциальных приближениях разностных схем // Изв. Сарат. ун-та. Нов. сер. Сер. Математика. Механика. Информатика. 2021. № С. 472–488.
9. *Gerdt V.P., Robertz D.* Consistency of Finite Difference Approximations for Linear PDE Systems and its Algorithmic Verification / In: S. Watt (ed.). Proceedings of ISSAC 2010 P. 53–59.
10. *Gerdt V.P.* Consistency Analysis of Finite Difference Approximations to PDE Systems / Mathematical Modelling in Computational Physics 2011. LNCS. V. 7125. 2012. P. 28–42.
11. *Scala R.L.* Gröbner bases and gradings for partial difference ideals // Math. Comput. 2015. № 84. P. 959–985.
12. *Gerdt V.P., Blinkov Yu.A.* Involution and Difference Schemes for the Navier–Stokes Equations / In: *Gerdt V.P., Mayr E.W., Vorozhtsov E.V.* (eds.) CASC 2009. LNCS. 2009. V. 5743. P. 94–105.
13. *Amodio P., Blinkov Yu.A., Gerdt V.P., La Scala R.* On Consistency of Finite Difference Approximations to the Navier-Stokes Equations / In: *Gerdt V.P., Koepf W., Mayr E.W., Vorozhtsov E.V.* (eds.) CASC 2013. LNCS. 2013. V. 8136. P. 46–60.
14. *Amodio P., Blinkov Yu.A., Gerdt V.P., Scala R.La.* Algebraic construction and numerical behavior of a new s-consistent difference scheme for the 2D Navier–Stokes equations // Appl. Math. and Comput. 2017. № 314. P. 408–421.
15. *Blinkov Yu.A., Gerdt V.P., Lyakhov D.A., Michels D.L.* A Strongly Consistent Finite Difference Scheme for Steady Stokes Flow and its Modified Equations / In: *Gerdt V.P., Koepf W., Mayr E.W., Vorozhtsov E.V.* (eds.) CASC 2018. LNCS. 2018. V. 11077. P. 67–81.
16. *Michels D.L., Gerdt V.P., Blinkov Y.A., Lyakhov D.A.* On the Consistency Analysis of Finite Difference Approximations // Journal of Mathematical Sciences (United States). 2019. № 5. P. 665–677.
17. *Gerdt V.P., Robertz D., Blinkov Yu.A.* Strong Consistency and Thomas Decomposition of Finite Difference Approximations to Systems of Partial Differential Equations / eprint arXiv:2009.01731. 2019. 48 p.
18. *Cartan E.* Sur certaines expressions différentielles et le problème de Pfaff // Annales scientifiques de l'École Normale Supérieure Sér. 3. 1899. № 16. P. 239–332.
19. *Kähler E.* Einführung in die Theorie der Systeme von Differentialgleichungen. Hamburger mathematische Einzelschriften 16. Leipzig: B.G. Teubner. 1934. Vol IV. 80 p.
20. *Riquier C.* Les Systèmes d'Equations aux Dérivées Partielles. Mémorial Sci. Math. XXXII, Gauthier-Villars, Paris. 1910.
21. *Harlow F.H., Welch J.E.* Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free // Phys. Fluids. 1965. № 8. P. 2182–2189.
22. *Buchberger B.* Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal: Ph.D. thesis / Universiät Innsbruck. 1965.
23. *Taylor G.I.* On the decay of vortices in a viscous fluid // Philosophical Magazine. 1923. V. 46. P. 671–674.
24. *Kovasznay L.I.G.* Laminar flow behind a two-dimensional grid // Mathematical Proceedings of the Cambridge Philosophical Society. 1948. V. 44. № 1. P. 58–62.

ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 38–47

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 517.36

ВЫЧИСЛЕНИЕ УНИМОДУЛЯРНЫХ МАТРИЦ СТЕПЕННЫХ ПРЕОБРАЗОВАНИЙ

© 2023 г. А. Д. Брюно[a,*] (ORCID: 0000-0002-7465-1258),
А. А. Азимов[b,] (ORCID: 0000-0002-7799-2525)**

[a]*Институт прикладной математики им. М.В. Келдыша РАН,*
125047 Москва, Миусская пл., д. 4, Россия

[b]*Самаркандский государственный университет,*
140104 Самарканд, Университетский бульвар, д. 15, Узбекистан

**E-mail: abruno@keldysh.ru*

***E-mail: Azimov_Alijon_Akhmadovich@mail.ru*

Поступила в редакцию 01.08.2022 г.
После доработки 13.08.2022 г.
Принята к публикации 07.09.2022 г.

Здесь указан алгоритм решения следующей задачи. Пусть в n-мерном вещественном пространстве задано $m < n$ целочисленных векторов. Их линейная оболочка образует линейное подпространство L в $\mathbb{R}^n$. Требуется вычислить такую унимодулярную матрицу, что линейное преобразование с ней переводит подпространство L в координатное. Также приведены программы, реализующие эти алгоритмы, и степенные преобразования, для которых они предназначены.

DOI: 10.31857/S013234742301003X, **EDN:** GRRDJF

1. ВВЕДЕНИЕ

Напомним, что матрица называется унимодулярной, если она квадратная, все ее элементы – целые числа и ее определитель равен ± 1. Ее обратная матрица также унимодулярна.

Будем векторы обозначать как векторы-строки: $X = (x_1,\ldots,x_n)$. $[x]$ – это целая часть вещественного числа x.

Задача 1. Пусть в n-мерном вещественном пространстве $\mathbb{R}^n$ задано m, $(m < n)$ целочисленных векторов $A_1,\ldots,A_m$. Их линейная оболочка

$$L = \left\{ X = \sum_{j=1}^{m} \lambda_j A_j, \lambda_j \in \mathbb{R}, j = 1,\ldots,m \right\} \qquad (1.1)$$

образует линейное подпространство в $\mathbb{R}^n$. Требуется вычислить такую унимодулярную матрицу α, что преобразование

$$X\alpha = Y$$

переводит подпространство L в координатное подпространство

$$M = \{Y : y_{n-l+1} = \ldots = y_n = 0\},$$

где $l = \dim L$.

Здесь указан алгоритм решения этой задачи, и составлена программа его реализующая. Это некоторое обобщение алгоритма цепной дроби [1], который напоминается в разделе 2. В разделе 3 описан алгоритм Эйлера [2], который обобщает алгоритм Евклида (т.е. цепной дроби) на n-мерный целочисленный вектор. В разделе 4 дается решение задачи 1, а в разделах 5, 6 – программы, соответствующие разделам 2, 3, 4. В разделе 7 рассмотрены степенные преобразования. Для вычисления их унимодулярных матриц развиты все эти алгоритмы и программы.

2. АЛГОРИТМ ЕВКЛИДА И ЦЕПНАЯ ДРОБЬ

Задача 2. Пусть заданы 2 целых положительных числа a_1 и a_2. Надо найти их наибольший общий делитель (НОД).

Для этого используется **алгоритм Евклида**. Пусть $a_1 \geq a_2$; делим a_1 на a_2 с остатком:

$$a_1 = b_1 \cdot a_2 + a_3, \qquad (2.1)$$

где $b_1 = [a_1/a_2]$ и a_3 – целые числа, $0 \leq a_3 < a_2$. Если $a_3 = 0$, то НОД равен a_2. Если $a_3 \neq 0$, то делим с остатком a_2 на a_3:

$$a_2 = b_2 \cdot a_3 + a_4, \qquad (2.2)$$

где $b_2 = [a_2/a_3]$ и где $0 \le a_4 < a_3$. Если $a_4 = 0$, то НОД равен a_3. Если $a_4 \ne 0$, то продолжаем процедуру, пока не получим нулевой остаток $a_{k+1} = 0$.

Тогда НОД – это a_k. Эту процедуру можно записать в виде цепной дроби

$$\frac{a_1}{a_2} = b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \cfrac{1}{\ldots + \cfrac{1}{b_{k-1}}}}}. \qquad (2.3)$$

Она применима к любому вещественному числу β и дает, вообще говоря, бесконечное разложение. Только для рациональных чисел $\beta = a_1/a_2$ оно конечно. Для квадратичных иррациональностей β оно периодично [1]. Если в цепной дроби (2.3) отбросить окончание, начинающееся с b_{l+1}, и свернуть полученную цепную дробь в рациональное число, то оно называется *подходящей дробью*.

Задача 3. Пусть заданы 2 целых положительных числа a_1 и a_2. Надо вычислить такую унимодулярную матрицу α, что $(a_1, a_2)\alpha = (a_k, 0)$ или $(0, a_k)$, где целое число $a_k > 0$.

Деление с остатком (2.1) можно записать в виде умножения на матрицу $(a_1, a_2)\begin{pmatrix} 1 & 0 \\ -b_1 & 1 \end{pmatrix} = (a_3, a_2)$ или $(a_1, a_2)\beta_1 = (a_3, a_2)$, где $\beta_1 = \begin{pmatrix} 1 & 0 \\ -b_1 & 1 \end{pmatrix}$, а деление с остатком (2.2) есть $(a_3, a_2)\begin{pmatrix} 1 & -b_2 \\ 0 & 1 \end{pmatrix} = (a_3, a_4)$ или $(a_3, a_2)\beta_2 = (a_3, a_4)$, где $\beta_2 = \begin{pmatrix} 1 & -b_2 \\ 0 & 1 \end{pmatrix}$. Последний шаг алгоритма Евклида есть либо

$$(a_k, a_{k-1})\begin{pmatrix} 1 & -b_{k-1} \\ 0 & 1 \end{pmatrix} = (a_k, 0),$$

либо

$$(a_{k-1}, a_k)\begin{pmatrix} 1 & 0 \\ -b_{k-1} & 1 \end{pmatrix} = (0, a_k).$$

Поэтому искомая матрица

$$\alpha = \beta_1\beta_2\cdots\beta_{k-1},$$

где

$$\beta_j = \begin{pmatrix} 1 & 0 \\ -b_j & 1 \end{pmatrix}, \qquad (2.4)$$

если j нечетно,

$$\beta_j = \begin{pmatrix} 1 & -b_j \\ 0 & 1 \end{pmatrix}, \qquad (2.5)$$

если j четно.

Поскольку все матрицы β_j унимодулярны, то их произведение α также унимодулярная матрица. Она дает решение задачи 3.

Отметим, что

$$\alpha^{-1} = \beta_{k-1}^{-1}\beta_{k-2}^{-1}\cdots\beta_2^{-1}\beta_1^{-1}$$

и согласно (2.4) и (2.5) $\beta_j^{-1} = \begin{pmatrix} 1 & 0 \\ b_j & 1 \end{pmatrix}$ или $\begin{pmatrix} 1 & b_j \\ 0 & 1 \end{pmatrix}$, т.е. содержит неотрицательные элементы. Поэтому в матрице α^{-1} все элементы неотрицательны.

Пример 1. Пусть $a_1 = 17$, $a_2 = 5$. Тогда $b_1 = [17/5] = 3$, $a_3 = 2$, $\alpha_1 = \begin{pmatrix} 1 & 0 \\ -3 & 1 \end{pmatrix}$,

$$b_2 = \left[\frac{5}{2}\right] = 2, \quad a_4 = 1, \quad \alpha_2 = \begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix},$$

$$b_3 = \left[\frac{2}{1}\right] = 2, \quad a_5 = 0, \quad \alpha_3 = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix}.$$

Матрица $\alpha = \alpha_1\alpha_2\alpha_3 = \begin{pmatrix} 1 & 0 \\ -3 & 1 \end{pmatrix}\begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix} = \begin{pmatrix} 5 & -2 \\ -17 & 7 \end{pmatrix}$.

Раскладывая в цепную дробь

$$\frac{17}{5} = 3 + \frac{1}{2 + \frac{1}{2}}, \qquad (2.6)$$

получим, что последняя подходящая дробь – это $3 + 1/2 = 7/2$. Поэтому в матрице α второй столбец состоит из $-2, 7$.

Здесь было предложено решение задачи 3 для $a_1, a_2 > 0$. Если $a_1, a_2 < 0$, то надо взять матрицу α для вектора $(-a_1, -a_2)$. Если $a_1 \cdot a_2 < 0$, то надо взять матрицу $\alpha = \begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix}$ для вектора $(|a_1|, |a_2|)$. Тогда матрица

$$\alpha = \begin{pmatrix} \alpha_{11}\operatorname{sign} a_1 & \alpha_{12}\operatorname{sign} a_2 \\ \alpha_{21}\operatorname{sign} a_1 & \alpha_{22}\operatorname{sign} a_2 \end{pmatrix}$$

является унимодулярной и аннулирует одну из координат вектора (a_1, a_2).

Здесь предполагалось, что $|a_1| \ge |a_2|$. Если это не так, то надо предварительно сделать перестановку координат

$$(a_1, a_2)\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = (a_2, a_1).$$

Похожее изложение см. в п. 1.9, § 1, гл. I [3].

3. АЛГОРИТМ ЭЙЛЕРА И ОБОБЩЕНИЕ ЦЕПНОЙ ДРОБИ

Задача 4. Пусть задан n-мерный целочисленный вектор $A = (a_1, a_2, \ldots, a_n)$. Надо найти такую n-мерную унимодулярную матрицу α, что вектор $A\alpha = C = (c_1, \ldots, c_n)$ содержит только одну координату c_n, отличную от нуля.

Для ее решения Эйлер [2] предложил следующий алгоритм. Пусть для начала все координаты вектора A неотрицательны. С помощью перестановки $A\alpha_0 = (\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n)$ упорядочим координаты

$$\tilde{a}_j \le \tilde{a}_{j+1}, \quad j = 1, \ldots, n-1.$$

Здесь α_0 – унимодулярная матрица перестановки. Пусть $\tilde{a}_k$ – наименьшее из чисел $\tilde{a}_j$, отличное от нуля.

Пусть

$$b_j = \left[\frac{\tilde{a}_j}{\tilde{a}_k}\right], \quad j = 1, \ldots, n.$$

При этом $b_1 = \cdots = b_{k-1} = 0$, $b_k = 1$. Делаем преобразование

$$d_j = \tilde{a}_j - b_j\tilde{a}_k, \quad 1 \le j \le n, \quad j \ne k, \quad d_k = \tilde{a}_k. \quad (3.1)$$

Ему соответствует унимодулярная матрица α_1, у которой на диагонали стоят единицы, в k-й строке стоят

$$0, 0, \ldots, 0, 1, -b_{k+1}, \ldots, -b_n,$$

т.е.

$$\tilde{A}\alpha_1 = D = (d_1, \ldots, d_n).$$

Теперь упорядочиваем компоненты вектора D с помощью унимодулярной матрицы-перестановки β_0 так, что $D\beta_0 = \tilde{D} = (0, \ldots, 0, \tilde{d}_k, \ldots, \tilde{d}_n)$, где $\tilde{d}_j \le \tilde{d}_{j+1}$.

Пусть $\tilde{d}_l$ – наименьшее из $\tilde{d}_j$, отличное от нуля, и $e_j = [\tilde{d}_j/\tilde{d}_l]$, $j = 1, \ldots, l$. Делаем преобразование

$$f_j = \tilde{d}_j - e_j\tilde{d}_l, \quad 1 \le j \le n, \quad j \ne l, \quad f_l = \tilde{d}_l,$$

и так далее. На каждом шаге максимум координат вектора убывает и является n-й координатой. Поэтому через конечное число шагов получаем вектор с одной ненулевой координатой – последней. Ее величина – это НОД всех исходных координат $a_1, \ldots, a_n$. Каждый шаг состоит из матрицы-перестановки и треугольной матрицы с единичной диагональю:

$$A\alpha_0\alpha_1\beta_0\beta_1\gamma_0\gamma_1 \ldots \omega_0\omega_1 = A\alpha = C = (0, \ldots, 0, c_n).$$

Матрица

$$\alpha = \alpha_0\alpha_1\beta_0\beta_1\gamma_0\gamma_1 \cdots \omega_0\omega_1 \quad (3.2)$$

является решением задачи 4.

Если не все координаты a_j исходного вектора A одного знака, то сначала упорядочиваем их по модулю

$$|\tilde{a}_j| \le |\tilde{a}_{j+1}|$$

и полагаем

$$b_j = [|\tilde{a}_j| / |\tilde{a}_k|] \operatorname{sign} \tilde{a}_j \operatorname{sign} \widetilde{a_k}.$$

Замечание 1. Умножая справа матрицу α на унимодулярную матрицу-перестановку, можно из вектора C получить вектор, у которого все координаты кроме одной равны нулю, а единственная ненулевая координата расположена на любом месте.

Пример 2. Пусть $n = 4$ и $A = (5, 2, 4, 3)$. Упорядочиваем координаты вектора A: $A \cdot \begin{pmatrix} 0&0&0&1 \\ 1&0&0&0 \\ 0&0&1&0 \\ 0&1&0&0 \end{pmatrix} = \tilde{A} = (2, 3, 4, 5)$, где $\begin{pmatrix} 0&0&0&1 \\ 1&0&0&0 \\ 0&0&1&0 \\ 0&1&0&0 \end{pmatrix} = \alpha_0$, имеем $k = 1$. Получаем

$$b_2 = \left[\frac{3}{2}\right] = 1, \quad b_3 = \left[\frac{4}{2}\right] = 2, \quad b_4 = \left[\frac{5}{2}\right] = 2.$$

Поэтому

$$\alpha_1 = \begin{pmatrix} 1&-1&-2&-2 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix},$$

$$\tilde{A}\alpha_1 = (2, 1, 0, 1) = D.$$

Упорядочиваем координаты вектора D:

$$D\begin{pmatrix} 0&0&0&1 \\ 0&1&0&0 \\ 1&0&0&0 \\ 0&0&1&0 \end{pmatrix} = (0, 1, 1, 2) = \tilde{D}, \text{ где } \begin{pmatrix} 0&0&0&1 \\ 0&1&0&0 \\ 1&0&0&0 \\ 0&0&1&0 \end{pmatrix} = \beta_0.$$

Здесь $l = 2$. Получаем $e_1 = 0$, $e_2 = 1$, $e_3 = 1$, $e_4 = 2$ и

$$\beta_1 = \begin{pmatrix} 1&0&0&0 \\ 0&1&-1&-2 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}.$$

$\tilde{D}\beta_1 = (0,1,0,0)$. Наконец, $\gamma_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ и

$\tilde{D}\beta_1\gamma_0 = (0,0,0,1) = C$.

Здесь

$$\alpha = \alpha_0\alpha_1\beta_0\beta_1\gamma_0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -2 & -1 & 3 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & -2 & 1 \end{pmatrix}. \qquad (3.3)$$

Пусть заданный вектор A является нормалью к некоторому линейному многообразию. Тогда после преобразования с помощью матрицы α получим, что этот вектор содержит первые $n-1$ нулевые координаты. Следовательно, все векторы исходного многообразия после преобразования будут иметь последней координатой ноль.

Алгоритм Эйлера обобщает алгоритм цепной дроби только для целочисленных векторов. Для произвольных вещественных векторов это обобщение искали все крупные математики XIX века. Но безуспешно. В [4] предложено такое обобщение цепной дроби для n-мерного вектора, которое дает последовательность наилучших приближений и периодично, если все координаты исходного вектора являются корнями многочлена степени n с целыми коэффициентами.

4. РЕШЕНИЕ ЗАДАЧИ 1

Пусть заданы целочисленные векторы

$$\begin{aligned} A_1 &= (a_{11}, a_{12}, \ldots, a_{1n}), \\ A_2 &= (a_{21}, a_{22}, \ldots, a_{2n}), \\ &\ldots \\ A_m &= (a_{m1}, a_{m2}, \ldots, a_{mn}) \end{aligned} \qquad (4.1)$$

($m < n$) и линейное пространство (1.1).

Первым делом проверяем, что среди них нет одинаковых. Если есть, то оставляем из них только один, а остальные отбрасываем. Поэтому будем считать, что все векторы (4.1) разные. Теперь применяем алгоритм Эйлера к вектору A_1, т.е. вычисляем матрицу α_0, такую, что $A_1\alpha_0 = C_1 = c_nE_n$, где c_n – целое число и E_k – это k-й единичный вектор.

Пусть $A_j\alpha_0 = C_j = (c_{j1}, \ldots, c_{jn})$, $j = 2, \ldots, m$. Полагаем $A_j^1 = (c_{j1}, \ldots, c_{jn-1})$, $j = 2, \ldots, m$. Применяем алгоритм Эйлера к $(n-1)$-мерному вектору A_2^1. Получаем $A_2^1\alpha_1 = C_2^1 = (0,0,\ldots,c_{n-1}^1)$, где α_1 – это $(n-1)$-мерная квадратная матрица. Пусть $A_j^1\alpha_1 = C_j^1 = (c_{j1}^1, \ldots, c_{jn-1}^1)$, $j = 3, \ldots, m$. Применяем алгоритм Эйлера к $(n-2)$-мерному вектору C_3^1 и т. д. В итоге получаем последовательность матриц $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ убывающих размерностей n, $n-1$, ..., $n-m+1$. Образуем блочные квадратные матрицы $\beta_j = \begin{pmatrix} \alpha_j & 0 \\ 0 & I_{j+1} \end{pmatrix}$, $j = 0, \ldots, n-m$, размерности n, где I_{j+1} – это единичные матрицы размерности $j+1$. Положим

$$\gamma = \beta_0\beta_1\cdots\beta_{m-1}.$$

Тогда $A_j\gamma = (0,0,\ldots,0,w_{j,n-j+1},\ldots,w_{j,n}) = W_j$, $j = 1, \ldots, m$. Матрица γ дает решение задачи 1.

Замечание 2. Умножая справа матрицу γ на матрицу-перестановку $\begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix}$, можно из набора векторов $W_1, \ldots, W_m$ получить треугольную матрицу $U = (u_{jk})$, $j = 1, \ldots, m$, $k = 1, \ldots, n$, у которой $u_{jk} = 0$, если $k > j$.

Пример 3 (продолжение примера 2). Пусть $n = 4$, $m = 2$, заданы векторы $A_1 = (5,2,4,3)$, $A_2 = (7,8,9,3)$. Согласно примеру 2 матрица α из (3.3) приводит вектор A_1 к виду (0, 0, 0, 1). Имеем

$$A_2 \cdot \alpha = (7,8,9,3)\begin{pmatrix} 0 & 1 & 0 & 0 \\ -2 & -1 & 3 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & -2 & 1 \end{pmatrix} = (-7,-4,18,-5) = C_2,$$

образуем вектор $A_2^1 = (-7,-4,18)$. К этому трехмерному вектору применяем алгоритм Эйлера. Упорядочиваем его координаты по модулю

$$A_2^1\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (-4,-7,18).$$

Имеем

$$d_2 = \left[\frac{7}{4}\right] = 1, \quad d_3 = -\left[\frac{18}{4}\right] = -4.$$

Получаем

$$(-4,-7,18)\begin{pmatrix} 1 & -1 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (-4,-3,2).$$

Упорядочиваем координаты по модулю

$$(-4,-3,2)\begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix}=(2,-3,-4).$$

Имеем

$$d_2=-\left[\frac{3}{2}\right]=-1,\quad d_3=-\left[\frac{4}{2}\right]=-2.$$

Получаем

$$(2,-3,-4)\begin{pmatrix}1&1&2\\0&1&0\\0&0&1\end{pmatrix}=(2,-1,0).$$

Упорядочиваем

$$(2,-1,0)\begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix}=(0,-1,2).$$

Имеем

$$d_3=-\left[\frac{2}{1}\right]=-2,$$

получаем

$$(0,-1,2)\begin{pmatrix}1&0&0\\0&1&2\\0&0&1\end{pmatrix}=(0,-1,0).$$

Упорядочиваем

$$(0,-1,0)\begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix}=(0,0,-1).$$

Теперь

$$\alpha_1=\begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix}\cdot\begin{pmatrix}1&-1&4\\0&1&0\\0&0&1\end{pmatrix}\cdot\begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix}\begin{pmatrix}1&1&2\\0&1&0\\0&0&1\end{pmatrix}\times$$
$$\times\begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix}\cdot\begin{pmatrix}1&0&0\\0&1&2\\0&0&1\end{pmatrix}\cdot\begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix}=\begin{pmatrix}0&2&1\\9&10&3\\2&3&1\end{pmatrix}. \qquad (4.2)$$

Полагаем

$$\beta_1=\begin{pmatrix}\alpha_1&0\\0&1\end{pmatrix}=\begin{pmatrix}0&2&1&0\\9&10&3&0\\2&3&1&0\\0&0&0&1\end{pmatrix}.$$

Теперь

$$\gamma=\alpha\beta_1=\begin{pmatrix}0&1&0&0\\-2&-1&3&-1\\1&0&0&0\\0&-1&-2&1\end{pmatrix}\cdot\begin{pmatrix}0&2&1&0\\9&10&3&0\\2&3&1&0\\0&0&0&1\end{pmatrix}=$$
$$=\begin{pmatrix}9&10&3&0\\-3&-5&-2&-1\\0&2&1&0\\-13&-16&-5&1\end{pmatrix}.$$

Проверяем $A_1\gamma=(0,0,0,1)$, $A_2\gamma=(0,0,-1,-5)$. Итак,

$$\begin{pmatrix}A_1\\A_2\end{pmatrix}\gamma=\begin{pmatrix}0,0,0,1\\0,0,-1,-5\end{pmatrix}.$$

5. ПРОГРАММЫ ВЫЧИСЛЕНИЯ ЦЕПНОЙ ДРОБИ

Алгоритмы вычисления цепных дробей реализованы в различных системах. Здесь опишем основные процедуры в двух системах компьютерной алгебры (СКА): в проприетарной `Maple` и в открытой `sympy`. Пакет `NumberTheory` в СКА Maple [5] позволяет получать разложение в цепную дробь рациональных, алгебраических и трансцендентных чисел, а также полиномов или элементарных функций от одной переменной. В пакете sympy [7] такой функционал реализован только для рациональных чисел или квадратичных иррациональностей. Если требуется работа с цепными дробями для других иррациональностей или трансцендентных чисел, то следует использовать открытую СКА `Sage` [6].

Для работы с рациональным числом в виде цепной дроби достаточно трех основных процедур: [1)]

1) преобразование в цепную дробь;

2) получение элементов цепной дроби;

3) получение рациональных приближений.

В СКА `Maple` соответствующие действия обеспечивают процедуры `ContinuedFraction`, `Term` и `Convergent`, а в sympy функционал пунктов 1 и 2 обеспечивает процедура `continued_fraction`, а пункта 3 – процедура *continued_fraction_convergents*.

Ниже приведем примеры реализации вычисления унимодулярных 2 × 2 матриц в соответствии с алгоритмами п. 1.9 книги [3]. Эти алгоритмы используют цепные дроби.

Первый алгоритм (см. [3], стр. 28–30) использует слагаемые разложения рационального числа p/q, а итоговая унимодулярная матрица получается путем последовательного умножения верхне- или нижнетреугольных матриц, которые составляются на основе элементов разложения цепной

дроби. Его реализация для Maple и для sympy приведена на листингах 1 и 2 соответственно.

Листинг 1

```
UniMod1:=proc (p:: integer, q:: integer)
description "Compute unimodular 2 × 2-
    matrix with the help of continued
    fraction";
uses NumberTheory, LinearAlgebra,
    ListTools;
local pabs:=abs (p), qabs:=abs (q), gcdpq
    :=igcd (pabs, qabs), cf_terms,
k,M, alpha:=DiagonalMatrix ([1, 1]);
if gcdpq!=1 then pabs:=pabs/gcdpq;
    qabs:=qabs/gcdpq; end if;
cf_terms:= Term(ContinuedFraction (
    pabs/qabs), all);
for k in [seq] ([i, cf_terms [i]], i =1..
    numelems (cf_terms)) do
M:=DiagonalMatrix ([1, 1]);
if type (k [1], even) then M [2, 1]:= -k
    [-1] else M [1, 2]:= -k[-1] end if;
alpha:=M. alpha;
end do;
return Matrix ([sign (p) *Column(alpha
    , 1), sign (q) *Column(alpha, 2)]);
[-1] else M [1, 2]:= =k[=1] end if;
alpha:=M. alpha;
end proc;
```

Листинг 2

```
import sympy as sym
from sympy import Rational, eye, Matrix
from sympy.ntheory.continued_fraction\
import continued_fraction_convergents,\
continued_fraction_iterator,\
continued_fraction

def UniMod1(p, q):
"""
Construct 2 × 2 unimodular matrix
for fraction p/q.
First variant
"""
r - Rational (p, q)
cfr - continued_fraction (r)
Mlst - [eye (2.1) for k in range (len (cfr))]
for k, m in enumerate(cfr):
if k%2=-1: Mlst [k] [1,0]=-m
else: Mlst [k] [0,1]= -m
alpha = eye (2.1)
for M in Mlst [:: -1]: alpha*-M
return alpha
```

Пример работы процедуры UniMod1 для пары чисел 5, 17 примера 1 приведен на листинге 3.

Листинг 3

```
>r := [5, 17]: UniMod2(op(r1)); %.Vector (r1);
```

$$\begin{bmatrix} 7 & -2 \\ -17 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Второй алгоритм (см. [3], стр. 30–31) использует только само рациональное число p/q и его последнюю подходящую дробь p_{n-1}/q_{n-1}. Его реализация приведена на листингах 4 и 5 соответственно.

Листинг 4

```
UniMod2:=proc (p:: integer, q:: integer)
description "Compute unimodular 2 × 2=
    matrix with the help of continued
    fraction. Second variant"; uses
    NumberTheory;
local pabs:=abs (p), qabs:=abs (q), gcdpq
    :=igcd (pabs, qabs), cf, cf_conv,
gamma, rho, Sigma, alpha;
if gcdpq!=1 then pabs:=pabs/gcdpq;
    qabs:=qabs /gcdpq; end if;
cf:= ContinuedFraction (pabs/qabs);
cf_conv:=Convergent (cf, all);
gamma:=cf_conv [=1];
rho:=cf_conv [=2];
Sigma:=numer (gamma=rho);
alpha:=Matrix ([[s i gn (p) *Sigma*denom(
    rho),=sign (q) *Sigma*numer (rho)],
[-sign (p) *denom(gamma), sign (q) *numer (
    gamma)]]);
return alpha;
end proc;
```

Листинг 5

```
import sympy as sym
```

```
from sympy import Rational, eye, Matrix
from sympy.ntheory.continued_fraction\
import continued_fraction_convergents,\
continued_fraction_iterator,\
continued_fraction

def UniMod2(p, q):
"""
Construct 2 × 2 unimodular matrix
for fraction p/q.
Second variant
"""
r = Rational (p, q)
cfr = continued_fraction (r)
cfr_conv = \
list (continued_fraction_convergents (cfr))
gamma = cfr1_conv [=1]
rho = cfr1_conv [=2]
sigma=(gamma=rho). numerator
alpha = Matrix ([[sigma* rho. denomina-
tor, \
=sigma * rho.numerator], \
[=gamma.denominator, gamma. numerator]])
return alpha
```

6. РЕАЛИЗАЦИЯ АЛГОРИТМА ЭЙЛЕРА И РЕШЕНИЯ ЗАДАЧИ 1

Для имплементации алгоритма Эйлера, описанного в разделе 3, был реализован набор процедур в СКА Maple, листинги которых представлены ниже вместе с кратким их описанием. Отметим, что целочисленный вектор в СКА Maple может быть представлен в двух различных формах: в виде списка (перечня чисел в квадратных скобках) или в виде вектора-строки (вектора-столбца) пакета LinearAlgebra (Vector[row] или Vector[column] соответственно). Если имя процедуры содержит цифру 2, то это означает, что входной целочисленный вектор может быть задан в одном из двух указанных видов.

Процедура MakePermute2 представлена на листинге 6. Она строит перестановочную матрицу по заданному вектору A. Результат работы процедуры – упорядоченный вектор и перестановочная матрица α_0. Порядок сортировки элементов вектора задается параметром sorting, но по умолчанию элементы упорядочиваются по возрастанию. В начале работы процедура проверяет, что вектор A не является нульмерным (строка 4 листинга 6).

Листинг 6

```
1  MakePermute2:= proc (A:: {Vector, list
      }, sorting:='<')
   uses LinearAlgebra;
   local Asort, Aind, Aper, i, nA:=
      numelems (A);
   if nA=0 then error ("Zero dimensional
      vector!"); end if;
   Aind:= sort (abs~(A), sorting, output
      = [permutat ion]);
6  Asort:= A[Aind];
   Aper:= Matrix (nA, nA, fill = 0);
   for i to numelems (Aind) do
   Aper [i, Aind [i]]:= 1;
   end do;
11 if type (A, Vector [row]) then
   return Asort, Transpose (Aper);
   else
   return Asort, Aper;
   end if;
16 end proc;
```

Вторая процедура MakeUnimod2 (листинг 7) для упорядоченного по возрастанию вектора A строит унимодулярную матрицу α_1, реализующую один шаг (3.1) алгоритма Эйлера.

Листинг 7

```
1  MakeUnimod2:= proc (As:: {Vector, list
     })
   uses ListTools, LinearAlgebra;
   local M, i, Amin, nminpos, ncol, absAs
     :=abs~(As),nA;
4  nA:=numelems (As);
   if nA=0 then error ("Zero dimensional
     vector!"); end if;
   M:= DiagonalMatrix ([seq] (1, k = 1..
     nA));
   Amin, nminpos:= FindMinimalElement (
     convert (absAs, list), position);
   Amin:= As [nminpos];
9  if Amin = 0 then
   ncol:= [Sear chAl l] (0, convert (absAs,
      list)) [=1] + 1;
   else
   ncol:= nminpos;
   end if;
14 for i from ncol+1 to nA do
   M[i, ncol]:= =t runc (As [i] /As [ncol]);
   end do;
   if type (As, Vector [row]) then
```

```
   return LinearAlgebra:=Transpose (M);
19 else
   return M;
   end if;
   end proc;
```

Рекурсивная процедура Unimodr2 (листинг 8) вычисляет по исходному вектору A унимодулярную матрицу α, которая преобразует A в вектор C с единственной последней ненулевой координатой. При первом ее вызове второй параметр Uni не указывается. В этом случае он полагается равным единичной матрице (строка 7 листинга). При последующих вызовах в процедуру передается унимодулярная матрица, вычисленная на предыдущих вызовах. Для своей работы процедура Unimod2 использует описанные выше процедуры MakePermute2 и MakeUnimod2. Условие окончания рекурсии – это получение вектора, у которого все элементы, кроме одного, равны нулю (строка 16 листинга).

Листинг 8

```
1  Unimodr2:= proc (A:: {Vector, list},
      Uni:: Matrix)
   uses LinearAlgebra, ListTools;
3  local Alen, Avec, Alst, As, Aper, M,
      Mperm, Auni, res, _;
   Alen:= numelems (A)
   if Alen=0 then error ("Zero
      dimensional vector!"); end if;
   if nargs = 1 then
   res:= DiagonalMatrix ([seq] (1, k = 1
      .. Alen));
8  else
   res:= Uni;
   end if;
   if type (A, list) then
   Alst:=A; Avec:=convert (A, Vector [
      column]);
13 elif type (A, Vector) then
   Alst:=convert (A, list); Avec:=convert (A
      , Vector [column]);
   end if;
   if Occurrences (0, Alst) = Alen = 1
      then
   _, Mperm:= MakePermute2 (A);
18 if type (A, list) then
   return convert (Mperm. Vector (A), list),
      Mperm. res;
   elif type (A, Vector [row]) then
   return A.Mperm, res.Mperm;
   else
23 return Mperm. Avec, Mperm. res;
   end if;
   end if;
   As, Aper:= MakePermute2 (A);
   M:= MakeUnimod2 (As);
28 if type (A, list) then
   Auni:= M. Aper;
   return Unimodr2 (convert (Auni. (Vector (
      A)), list), Auni. res);
   elif type (A, Vector [row]) then
   Auni:= Aper.M;
33 return Unimodr2 (A. Auni, res. Auni);
   el se
   Auni:= M. Aper;
   return Unimodr2 (Auni.A, Auni. res);
   end if;
38 end proc;
```

Ниже приведен листинг 9 решения примера 2 для вектора $A = (5,2,4,3)$.

Листинг 9

```
>A:= [5, 2, 4, 3]: Arow:=Vector [row] (A):
>Unimodr (Arow);
```

$$[0\ 0\ 0\ 1],\begin{bmatrix} 0 & 1 & 0 & 0 \\ -2 & -1 & 3 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & -2 & 1 \end{bmatrix}$$

Решение задачи 1 в соответствии с алгоритмом раздела 4 реализовано в рекурсивной процедуре UniSys, приведенной на листинге 10. Процедура получает исходный набор целочисленных векторов A_j, $j = 1,\ldots,m$ в виде списка Vlst. Если исходный список пуст (строка 5), либо число векторов превышает их размерность (строка 8), либо векторы являются линейно зависимыми (строки 9–15), либо векторы из набора имеют разный тип (строки 16–24) или разную размерность (строки 25–28), то процедура UniSys завершает свою работу. Если список состоит из одного вектора, то вызывается процедура Unimodr2, вычисляется матрица α – и на этом работа процедуры окончена. В противном случае осуществляется повторный вызов процедуры UniSys для набора векторов A_j^1, $j = 2,\ldots,m$, из которого исключен первый вектор, к оставшимся векторам применена унимодулярная матрица α. При этом размерность векторов A_j^1 уменьшена на единицу, а матрица α передается в виде параметра Uni для повторного вызова процедуры. При успешном окончании ра-

боты процедура UniSys возвращает итоговую матрицу γ, решающую задачу 1.

Листинг 10

```
UniSys:=proc (Vlst:: list, Uni:: Matrix)
uses LinearAlgebra;
local res, dlst,UM,_,Vdim, nVlst, Vtcol
   :=t rue,Mtmp;
nVl st:=numelems (Vl s t);
if nVl st=0 then error ("Initial list
   is empty!"); end if;
Vdim:=Dimension (Vlst [1]);
if type (Vlst [1], Vector [row]) then
   Vtcol:= false; end if;
if nVlst>Vdim then error ("Too many
   vector sofdimens ion ",Vdim); end
   if;
if Vtcol then
Mtmp:=Matrix (Vlst);
if Rank(Mtmp)<nVl st then error ("
   Vectors are not independent ! ");
   end if;
else
Mtmp:=<op (Vlst)>;
if Rank(Mtmp)<nVl st then error ("
   Vectors are not independent!");
   end if;
end if;
if Vtcol then
if not evalb ('and' (op ([seq] (type (V,
   Vector [column]),V in Vl s t)))) then
error ("All elements should be Vectors
   ");
end if;
else
if not evalb ('and' (op ([seq] (type (V,
   Vector [row]),V in Vl s t)))) then
error ("All elements should be Vectors
   ");
end if;
end if;
dlst:=[seq] (Dimension (V),V in Vl st);
if ListTools [Occur rences] (dlst [1],
   dlst) <> nVlst then
error ("All vectors should be o f the
   same size");
end if;
if nargs = 1 then
res:= DiagonalMatrix ([seq] (1, k = 1..
   Vdim));
else
res:= Uni;
end if;
_,UM:=Unimodr2 (Vlst [1]);
if nVl st=1 then
if Vtcol then return UM. res; else
   return res.UM; end if;
end if;
if Vtcol then
return DiagonalMatrix ([UniSys2 ([seq] (
   SubVector (UM.V, [1.. Vdim=1]), V in
   Vl s t [2.. =1])), 1]).UM;
else
return UM. DiagonalMatrix ([UniSys2 ([
   seq] (SubVector (V.UM, [1.. Vdim=1]),
   V in Vlst [2.. =1])), 1]);
end if;
end proc;
```

Ниже приведен листинг 11 решения примера 3 для векторов $A = (5,2,4,3)$ и $B = (7,8,9,3)$.

Листинг 11

```
>A:= [5, 2, 4, 3]: Arow:=Vector [row] (A):
>B:= [7–9, 3]: Brow:=Vector [row] (B):
>UniSys ([Arow,Brow]);
```

$$[0\ 0\ 0\ 1], \begin{bmatrix} 9 & 10 & 3 & 0 \\ -3 & -5 & -2 & -1 \\ 0 & 2 & 1 & 0 \\ -13 & -16 & -5 & 1 \end{bmatrix}$$

7. СТЕПЕННЫЕ ПРЕОБРАЗОВАНИЯ

Пусть задан многочлен

$$f(X) = \sum f_Q X^Q, \quad Q \in \mathbf{S}, \tag{7.1}$$

где $X = (x_1,\ldots,x_n) \in \mathbb{R}^n$ или $\mathbb{C}^n$, $Q = (q_1,\ldots,q_n) \in \mathbb{Z}^n$, $Q \geq 0$, f_Q – постоянные коэффициенты из $\mathbb{R}$ или $\mathbb{C}$, $\mathbf{S} = \mathbf{S}(f)$ – носитель многочлена f. Пусть $\mathcal{F}$ – алгебраическое многообразие $f(X) = 0$ и точка $X = X^0 \in \mathcal{F}$.

Если X^0 – простая точка, т.е. хотя бы одна из производных $\partial f/\partial x_j$ в этой точке X^0 отлична от нуля, то, по теореме о неявной функции, вблизи точки X^0 многообразие $\mathcal{F}$ описывается уравнением

$$\Delta x_j = \varphi(\Delta x_1,\ldots,\Delta x_{j-1},\Delta x_{j+1},\ldots,\Delta x_n), \tag{7.2}$$

где $\Delta x_k = x_k - x_k^0$, φ – сходящийся степенной ряд от своих аргументов.

Если точка X^0 – не простая, то согласно [8, 9] можно искать ветви многообразия $\mathcal{F}$, проходящие через точку X^0, в виде параметрических разложений

$$\Delta x_j = \varphi_j(\xi_1,\ldots,\xi_{n-1}), \quad i = 1,\ldots,n, \tag{7.3}$$

где ξ_k – малые параметры, а φ_j – сходящиеся степенные ряды. Для этого строится выпуклая оболочка Γ носителя $\mathbf{S}$ в пространстве $\mathbb{R}^n$. Тогда Γ – это многогранник, граница которого $\partial\Gamma$ состоит из (обобщенных) граней $\Gamma_j^{(d)}$ размерностей d, $0 \le d < n$. Здесь j – это номер грани. Поскольку все вершины $\Gamma_j^{(0)}$ многогранника Γ целочисленны, то каждая грань $\Gamma_j^{(d)}$ имеет $n - d$ целочисленных линейно независимых нормалей $N_{j1}^{(d)},\ldots,N_{jn-d}^{(d)} \in \mathbb{R}_*^n$, т.е. лежащих в пространстве $\mathbb{R}_*^n$, двойственном (сопряженном) пространству $\mathbb{R}^n$.

Кроме того, каждой грани $\Gamma_j^{(d)}$ соответствуют граничное множество

$$D_j^{(d)} = \left\{Q \in S \cap \Gamma_j^{(d)}\right\}$$

и укороченная сумма

$$\hat{f}_j^{(d)}(X) = \sum f_Q X^Q \quad \text{по} \quad Q \in D_j^{(d)}. \tag{7.4}$$

Теорема ([8, следствие § 3, гл. II] и теорема 3.1 [9]). *Для грани $\Gamma_j^{(d)}$ существует степенное преобразование*

$$\ln Y = \ln X \cdot \alpha, \tag{7.5}$$

где $\ln Y = (\ln y_1,\ldots,\ln y_n)$, $\ln X = (\ln x_1,\ldots,\ln x_n)$ с унимодулярной матрицей α, которое переводит укороченную сумму (7.4) *в многочлен g от d координат, т.е.*

$$\hat{f}_j^{(d)}(X) = Y^T g(y_1,\ldots,y_d), \tag{7.6}$$

где $T = (t_1,\ldots,t_n) \in \mathbb{Z}^n$.

Но в [8, 9] не было указано, как вычислять унимодулярную матрицу α. Это сделано в настоящей работе. А именно: если $n = 2$, то с помощью цепной дроби раздела 2, если $d = n - 1$, то с помощью алгоритма Эйлера, если $n > 2$ и $d < n - 1$, то с помощью алгоритма раздела 4, решающего задачу 1.

СПИСОК ЛИТЕРАТУРЫ

1. *Хинчин А.Я.* Цепные дроби. М.: Физматгиз, 1961.
2. *Euler L.* De relatione inter ternas pluresve quantitates instituenda // 1785, All Works 591.
3. *Брюно А.Д.* Локальный метод нелинейного анализа дифференциальных уравнений. М.: Наука, 1979. 252 с.
4. *Брюно А.Д.* Вычисление основных единиц числовых колец с помощью обобщенной цепной дроби // Программирование. 2019. № 2. С. 17–31. https://doi.org/10.1134/S0132347419020055
5. *Thompson I.* Understanding Maple. Cambridge University Press, 2016. 228 p.
6. *The Sage Developers.* SageMath, the Sage Mathematics Software System (Version 9.1.1). 2020. https://doi.org/10.5281/zenodo. 4066866. https://www.sagemath.org.
7. *Meurer A., Smith C.P.,* [et al.]. SymPy: symbolic computing in Python // PeerJ Computer Science. 2017. V. 3. e103. ISSN 2376–5992. DOI: . URL: https://doi.org/10.7717/ peerj-cs.103.
8. *Брюно А.Д.* Степенная геометрия в алгебраических и дифференциальных уравнениях. М.: Физматлит, 1998. 288 с.
9. *Брюно А.Д., Батхин А.Б.* Разрешение алгебраической сингулярности алгоритмами степенной геометрии // Программирование. 2012. № 2. С. 11–28.

ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 48–55

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.422.8

РЕАЛИЗАЦИЯ ГЕОМЕТРИЧЕСКОЙ АЛГЕБРЫ В СИСТЕМАХ СИМВОЛЬНЫХ ВЫЧИСЛЕНИЙ

© 2023 г. М. Н. Геворкян[a,*], А. В. Королькова[a,], Д. С. Кулябов[a,b,***], А. В. Демидова[a,****], Т. Р. Велиева[a,*****]**

[a] *Российский университет дружбы народов, 117198, Москва, ул. Миклухо-Маклая, д. 6, Россия*

[b] *Объединенный институт ядерных исследований, 141980, Дубна, Московская область, ул. Жолио-Кюри 6, Россия*

**E-mail: gevorkyan-mn@rudn.ru*

***E-mail: korolkova-av@rudn.ru*

****E-mail: kulyabov-ds@rudn.ru*

*****E-mail: demidova-av@rudn.ru*

******E-mail: velieva-tr@rudn.ru*

Поступила в редакцию 01.09.2022 г.
После доработки 02.09.2022 г.
Принята к публикации 08.09.2022 г.

Для описания специализированных математических структур предпочтительнее использовать более специальный формализм вместо более общего. Однако, зачастую в этом вопросе превалирует традиция. Например, для описания вращений в трехмерном пространстве, или например, для описания движения в пространствах Гилилея или Минковского обычно используют векторный (или тензорный) формализм взамен более специализированных формализмов представлений алгебры Клиффорда. Этот подход является исторически обусловленным. Применение специализированных формализмов (таких как спиноры или кватернионы) не стало научным мейнстримом, однако заняло свое место при решении практических и инженерных задач. Следует также отметить, что все операции в теоретических задачах проводятся именно с формульными данными. А манипуляции с многомерными геометрическими объектами подразумевают большое количество операций с одинаковыми объектами. И именно в таких задачах сильна компьютерная алгебра. В данной работе авторы хотят обратить внимание на один из таких специализированных формализмов, формализм геометрической алгебры. А именно, предлагается рассмотреть варианты реализации геометрической алгебры в рамках парадигмы символьных вычислений.

DOI: 10.31857/S0132347423010041, **EDN:** GRTBSQ

1. ВВЕДЕНИЕ

Геометрическая алгебра основывается на работах Г.Г. Грассмана [1], У.Р. Гамильтона [2] и У.К. Клиффорда [3]. Клиффорд в своих работах обобщил алгебру Грассмана и алгебру кватернионов Гамильтона. В геометрической алгебре рассматривается конкретная реализация алгебры Клиффорда, основанная на мультивекторах, которая включает в себя реализацию алгебры Грассмана в виде внешней алгебры p-векторов (контравариантных антисимметричных тензоров с операцией внешнего умножения). Алгебра кватернионов также является частным случаем алгебры мультивекторов.

Долгое время работы Клиффорда не привлекали особого внимание физиков и математиков. Алгебра Клиффорда была известна [4], но математический аппарат не был разработан. Д. Хинстейн [5] судя по всему является первым исследователем, который формализовал разработки Клиффорда в виде современного математического аппарата. Основная волна исследований на эту тему началась уже в 21 веке.

Первоначально алгебра мультивекторов формулировалась с прицелом на использование в физике [6, 7], однако довольно быстро она обрела большую популярность у специалистов по компьютерной графике [8, 9]. Объясняется это тем, что алгебра мультивекторов включает в себя в виде частных случаев алгебру комплексных чисел, кватернионов и бикватернионов, которые используются в компьютерной графике для описания вращений в двумерном, трехмерном и однородном пространствах.

В настоящее время сведения из геометрической алгебры входят во многие учебники по компьютерной графике [10, 11], а методы описания вращений с помощью бивекторов могут потенциально вытеснить кватернионы и бикватернионы [12].

В целом следует отметить прикладной характер исследований по геометрической алгебре [13, 14]. Это объясняется сравнительной простотой математического аппарата, для освоения которого достаточно стандартного курса линейной алгебры и некоторые сведения из общей алгебры [15]. Следует также отметить, что изучаемые объекты допускают довольно наглядное визуальное представление, что делает их изучение проще.

Исследования по использованию геометрической алгебры в различных областях физики также продолжаются, см. например, статью [16].

В данной статье дается обзор библиотеки GAlgebra [17] для языка Python. Данная библиотека реализует основные операции и объекты геометрической алгебры. Авторы уже делали обзор на библиотеку Grassmann.jl в статье [18] и приводили необходимые математические сведения из геометрической алгебры. Однако символьные алгебраические вычисления в Grassmann.jl крайне ограничены, а некоторые функции и вовсе не работают. Поэтому возникла задача найти более функциональную альтернативу, какой и стал пакет символьных вычислений для задач геометрической алгебры GAlgebra.

1.1. Структура статьи

Раздел 2 дает краткое описание основных понятий и операций геометрической алгебры. В разделе 3 статьи дается краткий обзор библиотек для разных языков программирования, реализующих операции геометрической алгебры. Оставшаяся часть статьи посвящена библиотеке GAlgebra и ее использованию. Изложение ведется на основе примеров двумерного (раздел 4) и трехмерного (раздел 5) евклидова пространства, а также более специфического двумерного пространства Минковского (раздел 6). Для данного пространства также указывается связь геометрической алгебры с гиперболическими числами и с преобразованиями Лоренца (раздел 7), что иллюстрирует применение геометрической алгебры в физике.

Все примеры кода, представленные в статье, выполнялись в оболочке Jupyter Notebook с интерпретатором языка Python 3.9.7 и библиотекой SymPy [19] версии 1.9. Примеры кода приводятся вместе с возвращаемым результатом. Для лучшей читабельности мы делаем некоторые косметические улучшения формул в результатах, например, выделяем векторы полужирным шрифтом.

2. ОСНОВНЫЕ ПОНЯТИЯ И ОПЕРАЦИИ ГЕОМЕТРИЧЕСКОЙ АЛГЕБРЫ

В этом разделе напомним основные понятия операции, связанные с геометрическим произведением и мультивекторами. В статье [18] нами уже кратко была изложена основа внешней алгебры p-векторов, которая входит в расширенные курсы алгебры, например [20], обозначений которой мы будем придерживаться.

Будем рассматривать евклидово или псевдоевклидово пространство L с ортонормированным базисом $\langle \mathbf{e}_1,\ldots,\mathbf{e}_n\rangle$, где буквой n всегда будем обозначать размерность пространства. Контравариантные кососимметричные тензоры валентности $(0, p)$ называются *p-векторами* или реже поливекторами. Пространство p-векторов и операция внешнего произведения $\wedge$ образуют *внешнюю алгебру* т.е. ассоциативную алгебра над полем $\mathbb{R}$ – реализацию абстрактной алгебры Грассмана.

Объекты, состоящие из формальной суммы всех возможных p-векторов, называются *мультивекторами* и образуют градуированную алгебру, называемую *геометрической алгеброй*, которая является реализацией алгебры Клиффорда относительно операции *геометрического произведения*.

Геометрическим произведением двух векторов $\mathbf{u}$ и $\mathbf{v}$ назовем отображение $L\times L\to L$, которое имеет следующие свойства.

• Геометрическое умножение двух скаляров сводится к обычному умножению, определенному в поле этих скаляров.

• Геометрическое умножение скаляра на вектор из L сводится к обычному умножению на скаляр, определенному в L.

• Геометрическое произведение вектора $\mathbf{u}$ самого на себя равно скаляру, значения которого равно норме вектора (норме):

$$\mathbf{u}^2 \equiv \mathbf{u}\mathbf{u} = \|\mathbf{u}\|^2.$$

• Дистрибутивность: $\mathbf{u}(\mathbf{v}+\mathbf{w}) = \mathbf{u}\mathbf{v}+\mathbf{u}\mathbf{w}$ и $\mathbf{u}(\mathbf{v}+\mathbf{w}) = \mathbf{u}\mathbf{v}+\mathbf{u}\mathbf{w}$. В силу того, что $\mathbf{u}$ может быть скаляром, из дистрибутивности следует линейность.

• Ассоциативность: $\mathbf{v}(\mathbf{u}\mathbf{w}) = (\mathbf{v}\mathbf{u})\mathbf{w}$.

Коммутативность или антикоммутативность явным образом не требуется. По соглашению геометрическое произведение никаким специальным символом не обозначается. Заметим, что геометрическое произведение действует на все градуировки рассматриваемой градуированной алгебры.

Более полезно конструктивное определение геометрического произведения через скалярное и внешнее произведения:

$$\mathbf{u}\mathbf{v} = (\mathbf{u},\mathbf{v}) + \mathbf{u}\wedge\mathbf{v}. \qquad (1)$$

Можно определить обратный вектор $\mathbf{u}^{-1} = \dfrac{\mathbf{u}}{\|\mathbf{u}\|^2}$ относительно геометрического произведения:

$$\mathbf{u}\frac{\mathbf{u}}{\mathbf{u}^2} = \frac{\mathbf{uu}}{\|\mathbf{u}\|^2} = \frac{\|\mathbf{u}\|^2}{\|\mathbf{u}\|^2} = 1,$$

Наличие ортонормированного базиса в пространстве L позволяет используя лишь формулу (1) вычислять геометрические произведения произвольных мультивекторов, разложенных по базисным p-векторам. Вычисления основываются на следующих двух формулах:

$$\mathbf{e}_i\mathbf{e}_j = -\mathbf{e}_j\mathbf{e}_i, \quad i \neq j \quad \text{и} \quad \mathbf{e}_i\mathbf{e}_i = 1. \qquad (2)$$

Оба соотношения можно получить, если рассмотреть геометрическое произведение двух ортонормированных базисных векторов: $\mathbf{e}_i\mathbf{e}_j = (\mathbf{e}_i, \mathbf{e}_j) + \mathbf{e}_i \wedge \mathbf{e}_j = \delta_{ij} + \mathbf{e}_i \wedge \mathbf{e}_j$.

Кроме того, если $i \neq j$, то $\mathbf{e}_i\mathbf{e}_j = \mathbf{e}_i \wedge \mathbf{e}_j$, то есть базис бивекторов можно выразить через геометрическое произведение. В общем случае для базиса p-вектора:

$$\mathbf{e}_{i_1 i_2 \ldots i_p} = \mathbf{e}_{i_1}\mathbf{e}_{i_2}\ldots\mathbf{e}_{i_p} = \mathbf{e}_{i_1} \wedge \mathbf{e}_{i_2} \wedge \ldots \wedge \mathbf{e}_{i_p}.$$

Используя (2) можно находить геометрические произведения любых мультивекторов, разложенных по базисным p-векторам. Например:

$$(3 + 5\mathbf{e}_1\mathbf{e}_2 - \mathbf{e}_1\mathbf{e}_3)(4\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3) = 12\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 + 20\mathbf{e}_1\mathbf{e}_2\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 -$$
$$- 4\mathbf{e}_1\mathbf{e}_3\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 = 12\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 - 20\mathbf{e}_3 - 4\mathbf{e}_2.$$

При раскрытии скобок использовалась ассоциативность и дистрибутивность, затем используя правила (2) упрощаем выражение $\mathbf{e}_1\mathbf{e}_2\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 = -\mathbf{e}_1\underbrace{\mathbf{e}_2\mathbf{e}_2}_{1}\mathbf{e}_1\mathbf{e}_3 = -\underbrace{\mathbf{e}_1\mathbf{e}_1}_{=1}\mathbf{e}_3 = -\mathbf{e}_3$. Аналогично поступаем и с $\mathbf{e}_1\mathbf{e}_3\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$. В результате получаем мультивектор, разложенный по базисным p-векторам.

Таким образом, вводить формальное определение геометрического произведения произвольных мультивекторов при работе в пространстве L с ортонормированным базисом не требуется – оно вводится конструктивно из определения геометрического произведения для векторов.

3. КРАТКИЙ ОБЗОР СУЩЕСТВУЮЩИХ БИБЛИОТЕК ПО ГЕОМЕТРИЧЕСКОЙ АЛГЕБРЕ

Математический аппарат геометрической алгебры получил широкую популярность среди специалистов по компьютерной графике. Вследствие этого довольно быстро появилось большое количество библиотек для разных языков программирования, которые реализуют различные возможности геометрической алгебры.

Основным источником сведений о существующих библиотеках с открытым исходным кодом, является сайт bivector.net [21], на котором в разделе lib собрано большое количество ссылок на актуальные библиотеки разного плана. Перечислим здесь некоторые из них.

Большинство библиотек ориентированы на численные расчеты.

• `Clifford` [22] – библиотека для языка Python, реализующая все основные операции геометрической алгебры в численном виде. Библиотека довольно стабильна, последняя версии 1.4.0 вышла в июле 2020 года.

• `Ganja.js` [23] – библиотека для языка JavaScript. Имеет встроенные возможности визуализации и интерактивного взаимодействия геометрических объектов. Существуют библиотеки-обертки и для других языков: `Python`, `C++`, `C#` и `Rust`.

• `Grassmann.jl` [24] – библиотека для языка Julia, последняя версии 0.7.7 вышла в ноябре 2021 года. Поддерживает как численные, так и символьные вычисления. Однако библиотека довольно нестабильна и ряд заявленных функций работает с ошибками.

• `Garamon` [25], `Klein` [26], `Versor` [27] – библиотеки для языка C++. Большой выбор библиотек именно для этого языка объясняется его доминированием в области компьютерной графики.

Все вышеперечисленные библиотеки ориентированы на численные расчеты, а не на символьные вычисления. Исключение составляет только `Grassmann.jl` так как частично поддерживает символьные вычисления с помощью системы компьютерной алгебры Reduce. Мы уже делали обзор [18] `Grassmann.jl` и указывали, что поддержка символьных вычислений не доработана и крайне ограничена. Впрочем, учитывая перспективность языка Julia [28, 29], стоит в будущем еще вернуться к решениям на его основе.

Судя по всему, единственной библиотекой с открытым исходным кодом с поддержкой символьных вычислений для геометрической алгебры является библиотека `GAlgebra` [17].

4. ДВУМЕРНОЕ ЕВКЛИДОВО ПРОСТРАНСТВО

Рассмотрим случай двумерного евклидова пространства, для которого существуют четыре внешние алгебры, перечисленные в таблице 1.

Размерность внешней алгебры $\dim \Lambda^p(L)$ зависит от *ранга p* (*градуировки*, grade) внешней алгебры и от размерности пространства L. При этом $\dim\Lambda^p(L) = C_n^p$ (см. рис. 1).

Таблица 1. Базисы для двумерного случая

p	$\Lambda^p(L)$	Базис	Элемент	$\dim \Lambda^p(L)$
0	$\mathbb{R}$	1	$a = a \cdot 1$	1
1	L	$\mathbf{e}_1, \mathbf{e}_2$	$\mathbf{a} = a^1\mathbf{e}_1 + a^2\mathbf{e}_2$	2
2	$\Lambda^2(L)$	$\mathbf{e}_1\mathbf{e}_2$	$\mathbf{a} = a^{12}\mathbf{e}_1\mathbf{e}_2$	1
3	$\Lambda^3(L)$	0	$\mathbf{a} = 0$	0

Приведем примеры с помощью GAlgebra. Для начала подключим библиотеку и настроим механизм отображение формул.

```
import sympy as sp
from galgebra.ga import Ga
sp.init_printing(latex_printer=latex,
↪ use_latex='mathjax' )
```

Зададим двумерное пространство, указав символы для базисных элементов, для индексов, а также единичную диагональную метрику g, которая определяет ортонормированность пространства:

```
xy = sp.symbols('1 2' , real=True)
o2d = Ga('e' , g= [1, 1], coords=xy)
```

Следует отметить, что GAlgebra позволяет задавать метрики только диагонального вида, но не обязательно единичные.

После определения пространства L можно задавать p-векторы произвольного вида, например, скаляр, вектор и бивектор:

```
a_scalar = o2d.mv('a' , 0)
a_vect = o2d.mv('a' , 1)
a_bivect = o2d.mv('a' , 2)
a_scalar, a_vect, a_bivect
```

$$(a, a^1\mathbf{e}_1 + a^2\mathbf{e}_2, a^{12}\mathbf{e}_1 \wedge \mathbf{e}_2)$$

Также можно записать базисные векторы в отдельные переменные и использовать их для формирования произвольных мультивекторов, например:

```
e_1, e_2 = o2d.mv()
 4*e_1 + 3*e_1*e_2
```

$$4\mathbf{e}_1 + 3\mathbf{e}_1 \wedge \mathbf{e}_2$$

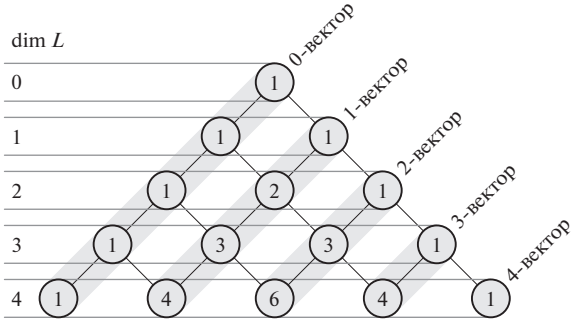

Рис. 1. Зависимость размерности p-вектора от градуировки внешней алгебры и размерности пространства L.

В GAlgebra все базисные элементы записываются через внешнее произведение $\wedge$, хотя при их задании мы использовали геометрическое умножение *.

Внешнее произведение двух векторов имеет простой геометрический смысл параллелограмма, построенного на этих векторах, а единственная компонента равна ориентированной площади этого параллелограмма.

```
a,b = o2d.mv('a' , 1), o2d.mv('b' , 2)
    a ^ b # внешнее произведение
```

$$(a^1b^2 - a^2b^1)\mathbf{e}_1 \wedge \mathbf{e}_2$$

Мультивекторы со скалярной и бивекторной частями $a + b\mathbf{e}_1\mathbf{e}_2$ изоморфны комплексным числам. Это легко проверить, показав, что базисный бивектор $\mathbf{e}_1\mathbf{e}_2$ обладает свойствами эллиптической мнимой единицы:

```
i = e_1*e_2
i*i == -1 # True
```

Создав два мультивектора указанного типа, можно с помощью GAlgebra проверить, что операции сложения, умножения и деления дают результат, полностью аналогичный комплексным числам.

```
u = o2d.mv('u' , 0) + o2d.mv('u' , 2)
v = o2d.mv('v' , 0) + o2d.mv('v' , 2)
u, v
```

$$(u + u^{12}\mathbf{e}_1 \wedge \mathbf{e}_2, v + v^{12}\mathbf{e}_1 \wedge \mathbf{e}_2),$$

```
u + v, u * v
```

$$(u + v) + (u^{12} + v^{12})\mathbf{e}_1 \wedge \mathbf{e}_2,$$

$$(uv - u^{12}v^{12}) + (uv^{12} + u^{12}v)\mathbf{e}_1 \wedge \mathbf{e}_2,$$

```
u / v
```

$$\frac{uv + u^{12}v^{12}}{v^2 + (v^{12})^2} + \frac{-uv^{12} + u^{12}v}{v^2 + (v^{12})^2}\mathbf{e}_1 \wedge \mathbf{e}_2.$$

Часто используют обозначение $\mathbf{i} = \mathbf{e}_1\mathbf{e}_2$, что делает алгебру мультивекторов $a + b\mathbf{i}$ практически неотличимой от алгебры комплексных чисел (эллиптического типа) [30] даже на уровне обозначений. Так в GAlgebra для бивекторов определена функция экспоненты:

```
θ = sp.Symbol(' θ ' , real=True)
(θ*i).exp()
```

В результате возвращается мультивектор вида:

$$\cos\theta + \sin\theta\mathbf{e}_1 \wedge \mathbf{e}_2.$$

Отметим, что в GAlgebra для мультивекторов определены операции сложения +, разности –, скалярного произведения |, внешнего произведения ^. Использование знака * для геометрического произведения не вносит никаких противоречий, так как для скаляров геометрическое умножения эквивалентно обычному умножению чисел.

5. ТРЕХМЕРНОЕ ЕВКЛИДОВО ПРОСТРАНСТВО

Увеличим теперь размерность L до трех и составим таблицу 2 всех возможных нетривиальных внешних алгебр.

Зададим трехмерное евклидово пространство с ортонормированным базисом и три произвольных вектора:

```
o3d = Ga('e' , g= [1, 1, 1], coords=xyz)
a, b, c = o3d.mv('a' , 1), o3d.mv('b' ,
↪ 1), o3d.mv('c' , 1)
```

В трехмерном пространстве внешнее произведение двух векторов дает бивектор с тремя компонентами, которые совпадают с компонентами псевдовектора векторного произведения $\mathbf{a} \times \mathbf{b}$:

```
a ^ b
```

$$(a^1b^2 - a^2b^1)\mathbf{e}_1 \wedge \mathbf{e}_2 + (a^1b^3 - a^3b^1)\mathbf{e}_1 \wedge \mathbf{e}_3 + \\ + (a^2b^3 - a^3b^2)\mathbf{e}_2 \wedge \mathbf{e}_3.$$

Заметим, что для получения точного соответствия векторному произведению, необходимо воспользоваться операцией правого или левого дополнения, которая в GAlgebra не реализована (собственно поэтому мы не упоминали ее в теоретическом введении). Этот недостаток для трехмерного пространства легко преодолим, так как операция дополнения эквивалентна умножению на обратный базисный тривектор $\mathbf{I} = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$ или, иначе, делению на $\mathbf{I}$.

```
I = e_1*e_2*e_3
(a^b) / I
```

$$(a^2b^3 - a^3b^2)\mathbf{e}_1 + (-a^1b^3 + a^3b^1)\mathbf{e}_2 + (a^1b^2 - a^2b^1)\mathbf{e}_3$$

В случае внешнего произведения трех векторов мы получим тривектор, единственная компонента которого совпадает со смешанным произведением трех векторов. Деление на $\mathbf{I}$ вновь дает точное значение, равное смешанному произведению.

```
a ^ b ^ c
(a ^ b ^ c) / I
```

$$a^1b^2c^3 - a^1b^3c^2 - a^2b^1c^3 + a^2b^3c^1 + a^3b^1c^2 - a^3b^2c^1.$$

Обозначение $\mathbf{I}$ для базисного тривектора выбрано из-за свойства эллиптической мнимой единицы $\mathbf{II} = -1$. Кроме того, базисные бивекторы $\mathbf{e}_1\mathbf{e}_2$, $\mathbf{e}_2\mathbf{e}_3$ и $\mathbf{e}_1\mathbf{e}_3$ проявляют свойства мнимых единиц кватернионов:

```
i, j, k = e_1*e_2, e_2*e_3, e_1*e_3
i*i == j*j == k*k == i*j*k == -1 # True
```

Подалгебра мультивекторов со скалярной и бивекторной частями изоморфна алгебре кватернионов. Заметим, что мультивекторы точно такого же вида в двумерном пространстве L были изоморфны алгебре комплексных чисел.

6. ДВУМЕРНОЕ ПСЕВДОЕВКЛИДОВО ПРОСТРАНСТВО

Рассмотрим подробно случай пространства $\Lambda(L)$, когда пространство $\mathbf{I}$ является двумерным пространством Минковского (двумерным псевдоевклидовым пространством $E^2_{1,1}$) с базисом $\langle \mathbf{e}_0, \mathbf{e}_1 \rangle$. В этом случае $\Lambda(L) = \Lambda^0(L) \oplus \Lambda^1(L) \oplus \Lambda^2(L)$ и общий вид мультивектора:

$$\mathbf{U} = u^0 + \underbrace{u^0\mathbf{e}_0 + u^1\mathbf{e}_1}_{\text{вектор}} + \underbrace{u^{01}\mathbf{e}_0\mathbf{e}_1}_{\text{бивектор}}.$$

Задать такое пространство в GAlgebra можно следующим образом:

```
tx = (t, x) = sp.symbols('0 1' ,
↪ real=True)
m2d = Ga('e' , g=[1, -1], coords=tx)
e0, e1 = m2d.mv()
```

Базисный бивектор $\mathbf{e}_0\mathbf{e}_1$ обозначим буквой $\mathbf{j}$, так как он обладает свойствами гиперболической мнимой единицы [30] относительно геометрического умножения:

```
j = e0*e1
j*j == 1 # return True
```

Таблица 2. Базисы для трехмерного случая

p	$\Lambda^p(L)$	Базис	Элемент	$\dim \Lambda^p(L)$
0	$\mathbb{R}$	1	$a = a \cdot 1$	1
1	L	$\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$	$\mathbf{a} = a^1\mathbf{e}_1 + a^2\mathbf{e}_2 + a^3\mathbf{e}_3$	3
2	$\Lambda^2(L)$	$\mathbf{e}_1\mathbf{e}_2, \mathbf{e}_1\mathbf{e}_3, \mathbf{e}_2\mathbf{e}_3$	$\mathbf{a} = a^{12}\mathbf{e}_1\mathbf{e}_2 + a^{23}\mathbf{e}_2\mathbf{e}_3 + a^{13}\mathbf{e}_1\mathbf{e}_3$	3
3	$\Lambda^3(L)$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$\mathbf{a} = a^{123}\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	1

Мультивектор, содержащий лишь скалярную и бивекторную части, можно записать в виде $\mathbf{u} = a + b\mathbf{e}_0\mathbf{e}_1 = a + b\mathbf{j}$. Множество мультивекторов такого вида, изоморфно относительно геометрического произведения множеству комплексных чисел гиперболического типа [30], для которых определены те же алгебраические операции, что и для стандартных эллиптических комплексных чисел[1]. Покажем, как эти операции реализованы в GAlgebra. Зададим вначале два мультивектора нужного нам вида. Обратите внимание, что здесь мы используем альтернативный способ задания, без функции mv.

```
a, b, c, d = sp.symbols('a, b, c, d' ,
↪ real=True)
u = a + b*j
v = c + d*j
```

Теперь испробуем все стандартные алгебраические операции. Начнем со сложения и умножения.

```
u + v, u * v
```

$$((a+c)+(b+d)\mathbf{e}_0\wedge\mathbf{e}_1,(ac+bd)+(ad+bc)\mathbf{e}_0\wedge\mathbf{e}_1).$$

Также определены сопряжение и норма (модуль)

```
~u, u.norm()
```

$$(a-b\mathbf{e}_0\wedge\mathbf{e}_1,\sqrt{|a^2-b^2|}).$$

Вместо операции деления покажем, как найти обратный элемент:

```
u.inv()
```

$$\frac{a}{a^2-b^2}-\frac{b}{a^2-b^2}\mathbf{e}_0\wedge\mathbf{e}_1.$$

В частности базисный бивектор **j** является обратным сам для себя так как $\mathbf{jj} = 1$, следовательно $\mathbf{j}^{-1} = \mathbf{j}$.

```
j = e0*e1
j.inv() == j # True
```

Для гиперболических чисел справедлив аналог формулы Эйлера $e^{\mathbf{j}\theta} = \theta + \mathbf{j}\theta, \theta \in \mathbb{R}$. В GAlgebra для бивектора определена операция exp:

```
θ= sp.Symbol(' θ ' , real=True)
(θ*j).exp();
```

которая как раз возвращает результат в виде аналога формулы Эйлера:

$$\mathrm{ch}\theta + \mathrm{sh}\theta\mathbf{e}_0\mathbf{e}_1 = \mathrm{ch}\theta + \mathrm{sh}\theta\mathbf{j}.$$

В евклидовом пространстве мультивекторы состоящие из скалярной и бивекторной частей изоморфны комплексным числам (двумерный случай) и кватернионам (трехмерный случай) и с помощью них задаются вращения на плоскости и в трехмерном пространстве. В случае псевдоевклидового пространства аналогичный мультивектор задает преобразования Лоренца и в двумерном случае изоморфен гиперболическим комплексным числам.

7. ПРЕОБРАЗОВАНИЯ ЛОРЕНЦА И РЕЛЯТИВИСТСКОЕ СЛОЖЕНИЕ СКОРОСТЕЙ

Релятивистские операции можно рассматривать как в представлении гиперболических комплексных чисел [30, 32], так и в представлении геометрической алгебры. Рассмотрим мультивектор $\mathbf{u} = \exp\{\theta\mathbf{j}\}$ и умножим его на произвольный вектор $\mathbf{x} = x^0\mathbf{e}_0 + x^1\mathbf{e}_1$ пользуясь геометрическим умножением:

$$\mathbf{ux} = (\mathrm{ch}\theta + \mathbf{j}\mathrm{sh}\theta)(x^0\mathbf{e}_0 + x^1\mathbf{e}_1) = \mathrm{ch}\theta x^0\mathbf{e}_0 +$$
$$+ \mathrm{ch}\theta x^1\mathbf{e}_1 + \mathrm{sh}\theta x^0\mathbf{j}\mathbf{e}_0 + x^1\mathrm{sh}\theta\mathbf{j}\mathbf{e}_1 = \mathrm{ch}\theta x^0\mathbf{e}_0 -$$
$$- \mathrm{sh}\theta x^1\mathbf{e}_0 + \mathrm{ch}\theta x^1\mathbf{e}_1 - \mathrm{sh}\theta x^0\mathbf{e}_1 =$$
$$= (\mathrm{ch}\theta x^0 - \mathrm{sh}\theta x^1)\mathbf{e}_0 + (-\mathrm{sh}\theta x^0 + \mathrm{ch}\theta x^1)\mathbf{e}_1.$$

Мы получили преобразование Лоренца:

$$\begin{bmatrix} \mathrm{ch}\theta & -\mathrm{sh}\theta \\ -\mathrm{sh}\theta & \mathrm{ch}\theta \end{bmatrix}\begin{bmatrix} x^0 \\ x^1 \end{bmatrix},$$

где $\mathrm{ch}\theta$ равен коэффициенту Лоренца γ:

$$\mathrm{ch}\theta = \gamma = \frac{1}{\sqrt{1-\frac{v^2}{c^2}}}, \quad \mathrm{th}\theta = \frac{v}{c}.$$

В GAlgebra данные операции выполняются следующим образом:

```
θ1 = sp.Symbol(' θ1 ' , real=True)
θ2 = sp.Symbol(' θ2 ' , real=True)
u = (θ*j).exp()
x = m2d.mv(' x ' , 1)
u*x
```

$(x^0\mathrm{ch}\theta - x^1\mathrm{sh}\theta)\mathbf{e}_0 + (-x^0\mathrm{sh}\theta + x^1\mathrm{ch}\theta)\mathbf{e}_1$

Релятивистскую формулу сложения скоростей можно получить выполнив преобразования Лоренца два раза подряд:

$$\mathbf{x}' = \exp(\theta_2\mathbf{j})\exp(\theta_1\mathbf{j})\mathbf{x} =$$
$$= \exp\big((\theta_1+\theta_2)\mathbf{j}\big)\mathbf{x} \Rightarrow \theta_3 = \theta_1 + \theta_2.$$

Так как $\mathrm{th}\theta = \frac{v}{c}$, то $\mathrm{th}\theta_3 = \mathrm{th}(\theta_1 + \theta_2)$ и по формуле тангенса суммы:

$$\mathrm{th}(\theta_3) = \frac{\mathrm{th}\theta_1 + \mathrm{th}\theta_2}{1 - \mathrm{th}\theta_1\mathrm{th}\theta_2} = \frac{v_1 + v_2}{1 - \frac{v_1v_2}{c^2}}.$$

[1] Фактически, здесь использован виковский поворот [31].

Сложение скоростей свелось к сложению гиперболических углов. Естественно, что в двумерном случае это сложение коммутативно и ассоциативно (в отличии от случая полного пространства Минковского).

То же самое легко повторить с помощью GAlgebra:

```
u1 = ('θ1 ' *j).exp()
u2 = ('θ2 ' *j).exp()
u1*u2*x
```

$$(x^0\mathrm{ch}(\theta_1+\theta_2) - x^1\mathrm{sh}(\theta_1+\theta_2))\mathbf{e}_0 +$$
$$+ (-x^0\mathrm{sh}(\theta_1+\theta_2) + x^1\mathrm{ch}(\theta_1+\theta_2))\mathbf{e}_1.$$

Фактически, применяя специализированный формализм геометрической алгебры мы смогли вместо громоздких (векторных) операций с леренцовыми коэффициентами применить элегантные операции поворотов в соответствующем пространстве.

8. ЗАКЛЮЧЕНИЕ

Мы рассмотрели практически весь основной функционал модуля `GAlgebra` для языка `Python` и пакета компьютерной алгебры `SymPy`. За рамками статьи остались лишь примеры использования дифференциального исчисления ввиду их объемности. В рамках геометрической алгебры вводится оператор градиента, который обобщает стандартные операторы векторного анализа. Примером его использования может служить запись уравнений Максвелла в виде единственного уравнения и крайне простая процедура получения волнового уравнения (подробнее см. [7]).

Если ограничиться рассмотрением лишь свободного программного обеспечения, то модуль GAlgebra является безальтернативным. Существуют и другие библиотеки для разных языков программирования, но практически все они ограничены численными вычислениями.

Укажем также и некоторые недостатки модуля. Так, например, в нем отсутствует операция дополнения, что впрочем компенсируется геометрическим делением на базисный элемент максимального ранга.

Более существенным недостатком является невозможность задания метрики с нулями на диагонали. Такая метрика описывает проективное пространство, для которого в рамках геометрической алгебры развита довольно проработанная теория. Проективная геометрическая алгебра (PGA – Projective geometric algebra) наиболее удобна для использования в задачах компьютерной графики.

9. БЛАГОДАРНОСТИ

Публикация выполнена при поддержке Программы стратегического академического лидерства РУДН.

СПИСОК ЛИТЕРАТУРЫ

1. *Grassmann H.G.* Die mechanik nach den principien der ausdehnungslehre // Mathematische Annalen. 1877. Bd. 12. S. 222–240.
2. *Kuipers J.B.* Quaternions and rotation sequences. Princeton University Press, 1999.
3. *Clifford W.K.* Applications of grassmann's extensive algebra // American Journal of Mathematics. 1878. V. 1. № 4. P. 350–358.
4. *Казанова Г.* Векторная алгебра / Под ред. М.К. Поливанова. Современная математика. М.: Мир, 1979.
5. *Hestenes D., Sobczyk G.* Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics. Fundamental Theories of Physics. Springer Netherlands, 1987. ISBN: 9789027725615.
6. *Delanghe R., Sommen F., Soucek V.* Clifford algebra and spinor-valued functions. Mathematics and Its Applications. Kluwer Academic Publishers, 1992.
7. *Doran C., Lasenby A.* Geometric Algebra for Physicists. Morgan Kaufmann Publishers, 2003. ISBN: 9780123694652.
8. *Dorst L., Fontijne D., Mann S.* Geometric algebra for computer science. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 2007. ISBN: 0123694655.
9. *Vince J.* Geometric algebra for computer graphics. Springer-Verlag, 2008. ISBN: 9781846289965.
10. *Lengyel E.* Mathematics. Lincoln, California: Terathon Software LLC, 2016. V. 1. ISBN: 9780985811747.
11. *Kanatani K.* Understanding Geometric Algebra. Taylor and Francis Group/CRC, 2015. ISBN: 9781482259513.
12. ten Bosch M. Let's remove quaternions from every 3d engine. URL: https://marctenbosch.com/quaternions/.
13. *Perwa C.B.U.* Geometric Algebra with Applications in Engineering. Geometry and Computing. Springer-Verlag Berlin Heidelberg, 2009. ISBN: 9783540890676.
14. *Joot P.* Geometric Algebra for Electrical Engineers: Multivector Electromagnetism. CreateSpace Independent Publishing Platform, 2019. ISBN: 9781987598971.
15. *Winitzki S.* Linear Algebra via Exterior Products. 2020. URL: https://github.com/winitzki/linear-algebra-book.
16. *Chappell J.M., Drake S.P., Seidel C.L. et al.* Geometric algebra for electrical and electronic engineers // Proceedings of the IEEE. 2014. V. 102. № 9. P. 1340–1363.
17. Galgebra – symbolic geometric algebra/calculus package for sympy. 2022. URL: https://galgebra.readthedocs.io/en/latest/index.html.
18. *Геворкян М.Н., Демидова А.В., Велиева Т.Р. и др.* Аналитико-численная реализация алгебры поливекторов на языке julia // Программирование. 2022. № 1. С. 54–64.

19. Sympy. 2022. URL: http://www.sympy.org/ru/index.html.

20. *Кострикин А.И.* Линейная алгебра. М.: МЦНМО, 2009. Т. 2. ISBN: 9785940574545.

21. Bivector.net: Geometric algebra resources. 2022. URL: https://bivector.net/index.html.

22. *Hadfield H., Wieser E.*, *Arsenovic A.* et al. pygae/clifford. 2022.

23. *De Keninck S.* ganja.js. 2020.

24. *Grassmann.jl*. 2022. URL: https://github.com/chakravala/Grassmann.jl.

25. *Breuils S., Nozick V., Fuchs L.* Garamon: A geometric algebra library generator // Advances in Applied Clifford Algebras. 2019. 7. V. 29. № 4. P. 69.

26. *Gunn C.G., Keninck S.D.* Geometric algebra and computer graphics // ACM SIGGRAPH 2019. Courses. ACM, 2019. 7.

27. *Colapinto P.* Versor: Spatial computing with conformal geometric algebra. 2011. Available at http://versor.mat.ucsb.edu. URL: http://versor.mat.ucsb.edu.

28. *Кулябов Д.С., Королькова А.В.* Компьютерная алгебра на julia // Программирование. 2021. № 2. С. 44–50. 2108.12301.

29. *Gevorkyan M.N., Kulyabov D.S., Korolkova A.V. et al.* Symbolic implementation of multivector algebra in julia language // Computer algebra: 4th International Conference Materials. LCC MAKS Press, 2021. 5. P. 57–60.

30. *Kulyabov D.S., Korolkova A.V., Sevastianov L.A.* Complex numbers for relativistic operations. MDPI AG, 2021. 12.

31. *Зи Э.* Квантовая теория поля в двух словах. Регулярная и хаотическая динамика, 2009. ISBN: 978-5-93972-770-9.

32. *Kulyabov D.S., Korolkova A.V., Gevorkyan M.N.* Hyperbolic numbers as einstein numbers // Journal of Physics: Conference Series. 2020. 5. V. 1557. P. 012027.

ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 56–60

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 517.55 + 004.421.6

АЛГОРИТМ ВЫЧИСЛЕНИЯ СРЕЗКИ ДИСКРИМИНАНТА МНОГОЧЛЕНА

© 2023 г. А. П. Ляпин[a,b,*] (ORCID: 0000-0002-0149-7587),
Е. Н. Михалкин[a,] (ORCID: 0000-0002-1410-9117)**

[a]*Сибирский федеральный университет*
660041 Красноярск, пр. Свободный, 79, Россия
[b]*Фэрмонтский государственный университет*
26554 Фэрмонт, Западная Вирджиния, Локуст ул., 1201, США
**E-mail: aplyapin@sfu-kras.ru*
***E-mail: mikhalkin@bk.ru*

Поступила в редакцию 14.07.2022 г.
После доработки 14.08.2022 г.
Принята к публикации 22.08.2022 г.

Разработана программа, вычисляющая срезку дискриминанта многочлена одной переменной на грани многогранника Ньютона дискриминанта данного многочлена, а также результат ее факторизации в произведение дискриминантов многочленов меньших степеней.

DOI: 10.31857/S0132347423010065, **EDN:** GSECON

1. ВВЕДЕНИЕ

Дискриминантом многочлена степени n:

$$f(y) = a_0 + a_1 y + \ldots + a_n y^n \qquad (1.1)$$

называется неприводимый многочлен $\Delta_n(a_0, a_1, \ldots, a_n)$ с целочисленными коэффициентами, который обращается в нуль тогда и только тогда, когда f имеет кратные корни. Иногда дискриминант многочлена f будем обозначать $\Delta_n(f)$.

Структура многогранника Ньютона дискриминанта отражает геометрию дискриминантной гиперповерхности. В частности, асимптотическое поведение гиперповерхности “на бесконечности” контролируется “экстремальными” мономами дискриминанта, которые соответствуют вершинам многогранника Ньютона. Дискриминанты, в свою очередь, играют фундаментальную роль в теории алгебраических функций, в теории особенностей, в алгебраической геометрии и математической физике, о чем свидельствует значительное число публикаций (см., например, [1–6]).

В работах [7–9] было доказано интересное свойство срезок дискриминанта на грани его многогранника Ньютона, а именно, они допускают факторизацию в дискриминанты многочленов меньших степеней. *Данная работа посвящена алгоритму вычисления этих срезок. Также приводится результат факторизации полученных срезок в дискриминанты других многочленов. В основе алгоритма лежат формулы, приведенные и доказанные в статьях [8, 10] для граней многогранника Ньютона дискриминанта общего многочлена одного переменного. Используются факторизационные формулы, полученные в недавних статьях [7, 8].*

Отметим, что количество слагаемых в дискриминанте многочлена быстро растет с увеличением степени многочлена. Так, если дискриминант кубического уравнения содержит пять слагаемых, то дискриминант многочлена четвертой степени состоит из шестнадцати слагаемых, для многочлена пятой степени дискриминант содержит 59 слагаемых, а для многочлена 10-й степени он состоит из 133881 монома. Поэтому вычисление срезки дискриминанта весьма затруднительно без специальной программы.

2. МНОГОГРАННИК НЬЮТОНА ДЛЯ ДИСКРИМИНАНТА МНОГОЧЛЕНА

Напомним, что *многогранником Ньютона* $\mathcal{N}(\Delta_n)$ дискриминанта многочлена (1.1) называется выпуклая оболочка в $\mathbb{R}^{n+1}$ множества всех показателей $k = (k_0, k_1, \ldots, k_n)$ мономов, участвующих в Δ_n.

Доказанная в [1] теорема показывает, что каждая вершина многогранника Ньютона $\mathcal{N}(\Delta_n)$ для дискриминанта многочлена f определяется некоторым разбиением отрезка $[0, n]$ набором целых точек

$$0 = i_0 < i_1 < \ldots < i_s < i_{s+1} = n.$$

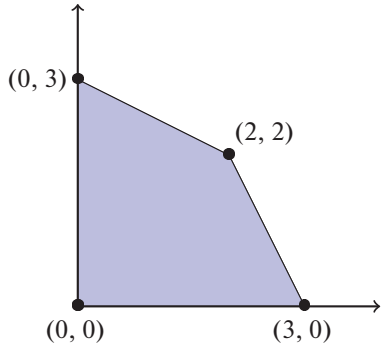

Рис. 1. Многогранник Ньютона приведенного дискриминанта кубического многочлена

Теорема 1 ([1], с. 412). *Многогранник Ньютона дискриминанта многочлена* (1.1) *комбинаторно эквивалентен* $(n-1)$*-мерному кубу*; *он содержит* 2^{n-1} *вершин*, *которые находятся в биективном соответствии со всевозможными подмножествами*

$$I \subset \{1,2,\ldots,n-1\}.$$

Вершина v_I, *соответствующая подмножеству* $I = \{i_1 < i_2 < \ldots < i_s\}$, *имеет координаты*

$$k_0 = i_1 - i_0 - 1, \quad k_n = i_{s+1} - i_s - 1,$$
$$k_{i_q} = i_{q+1} - i_{q-1}, \quad \text{для} \quad i_q \in I,$$
$$k_i = 0, \quad \text{для} \quad i \notin I \cup \{0,n\}.$$

Пусть $l_q = i_{q+1} - i_q$ $(0 \le q \le s)$. *Тогда моном*

$$a^{v_I} = a_0^{l_0-1} a_{i_1}^{l_1+l_0} a_{i_2}^{l_2+l_1} \ldots a_{i_s}^{l_s+l_{s-1}} a_n^{l_s-1}$$

встречается в Δ_n *с коэффициентом*

$$c_{v_I} = \prod_{q=0}^{s} (-1)^{\frac{l_q(l_q-1)}{2}} l_q^{l_q}.$$

Проиллюстрируем утверждение теоремы на примере кубического многочлена

$$f(y) = a_0 + a_1 y + a_2 y^2 + a_3 y^3.$$

Дискриминант Δ этого многочлена имеет вид:

$$-27a_0^2a_3^2 - 4a_1^3a_3 - 4a_0a_2^3 + a_1^2a_2^2 + 18a_0a_1a_2a_3.$$

В рассматриваемом случае имеется 4 подмножества $I \subset \{1,2\}$:

$$I_0 = \varnothing, \quad I_1 = \{1\}, \quad I_2 = \{2\}, \quad I_3 = \{1,2\}.$$

Соответствующие мономы будут следующими:

$$-27a_0^2a_3^2, \ -4a_1^3a_3, \ -4a_0a_2^3, \ a_1^2a_2^2.$$

Отметим, что моном $18a_0a_1a_2a_3$ соответствует внутренней целочисленной точке $(1,1,1,1) \subset \mathcal{N}(\Delta)$ и теорема о нем ничего не утверждает.

Многогранник $\mathcal{N}(\Delta_n)$ имеет $n-1$ гиперграней $\{h_k^0\}$, лежащих в координатных гиперплоскостях $\{t_k = 0\}$, $k = 1,\ldots,n-1$ (предполагается, что в объемлющем пространстве $\mathbb{R}^{n+1}$ выбраны координаты $t = (t_0,t_1,\ldots,t_{n-1},t_n)$), и $n-1$ некоординатных гиперграней. Здесь речь идет о гипергранях наивысшей размерности, т.е. коразмерности 1. Обозначим через h_k некоординатную грань многогранника $\mathcal{N}(\Delta_n)$, противоположную к координатной грани h_k^0.

В работах [8] и [10] показано, что многогранник $\mathcal{N}(\Delta_n)$ в пространстве $\mathbb{R}^{n+1}$ переменных $t_0,t_1,\ldots,t_n$ высекается следующими неравенствами:

$$\begin{cases} t_k \ge 0, \\ \sum\limits_{j=1}^{n-1} \min(j,k)[n - \max(j,k)]t_j \le nk(n-k), \end{cases} \tag{2.1}$$

$k = 1,2,\ldots,n-1$. Каждая его некоординатная гиперграннь h_k определяется уравнением

$$h_k := \left\{ t \in \mathcal{N}(\Delta_n) : \sum_{j=1}^{n-1} \min(j,k)[n - \max(j,k)]t_j = nk(n-k) \right\}.$$

Согласно известному свойству биоднородности дискриминанта, переменные t_0,t_n однозначно восстанавливаются через переменные $t_1,\ldots,t_{n-1}$ по формулам

$$\sum_{j=0}^{n} t_j = 2(n-1), \ \sum_{j=1}^{n} jt_j = n(n-1). \tag{2.2}$$

Отметим, что при вычислении дискриминанта многочлена (1.1) мы можем сокращать число переменных на два. А потом, по формулам (2.2), сокращенные переменные восстанавливать (см. пример в конце этого параграфа).

На рисунке 1 изображен многогранник Ньютона дискриминанта приведенного кубического многочлена

$$y^3 + a_2y^2 + a_1y + 1.$$

Это есть проекция многогранника $\mathcal{N}(\Delta_3)$ полного кубического многочлена на плоскость $t_0 = 0$, $t_3 = 0$. Для рассматриваемого многочлена дискриминант Δ равен

$$-27 - 4a_1^3 - 4a_2^3 + a_1^2a_2^2 + 18a_1a_2. \tag{2.3}$$

Согласно формулам (2.1) многогранник Ньютона приведенного дискриминанта рассматриваемого кубического многочлена задается системой неравенств

$$t_1 \ge 0, \quad t_2 \ge 0, \quad 2t_1 + t_2 \le 6, \quad t_1 + 2t_2 \le 6.$$

Что же касается дискриминанта полного кубического многочлена, то он находится из дискриминанта (2.3) приведенного кубического многочлена по формулам

$$t_0 = \frac{6 - 2t_1 - t_2}{3}, \quad t_3 = \frac{6 - t_1 - 2t_2}{3},$$

которые получаются из (2.2), и имеет вид

$$-27a_0^2a_3^2 - 4a_1^3a_3 - 4a_0a_2^3 + a_1^2a_2^2 + 18a_0a_1a_2a_3.$$

3. ФАКТОРИЗАЦИЯ СРЕЗОК ДИСКРИМИНАНТА НА ГРАНИ МНОГОГРАННИКА НЬЮТОНА

Срезкой дискриминанта Δ на гипергрань h_k его многогранника Ньютона мы называем многочлен $\Delta|_{h_k}$, состоящий из всех мономов Δ, показатели которых принадлежат h_k. В качестве примера вычислим срезку дискриминанта многочлена четвертой степени

$$f(y) = a_0 + a_1y + a_2y^2 + a_3y^3 + a_4y^4 \qquad (3.1)$$

на гипергрань h_2. Как показывают вычисления, дискриминант многочлена (3.1) следующий:

$$\begin{aligned} &56a_0^3a_4^3 - 27a_1^4a_4^2 - 128a_0^2a_2^2a_4^2 - 4a_3a_3^3 + 16a_0a_2^4a_4 - \\ &- 4a_1^2a_2^3a_4 - 27a_0^2a_3^4 - 192a_0^2a_1a_3a_4^2 - 6a_0a_1^2a_3^2a_4 + \\ &+ 144a_0^2a_2a_3^2a_4 + 144a_0a_1^2a_2a_4^2 + 18a_1^3a_2a_3a_4 + \\ &+ a_1^2a_2^2a_3^2 - 4a_0a_2^3a_3^2 - 80a_0a_1a_2^2a_3a_4 + 18a_0a_1a_2a_3^3. \end{aligned}$$

Грань h_2 многогранника Ньютона многочлена (3.1) определяется системой:

$$t_1 \ge 0, \quad t_2 \ge 0, \quad t_3 \ge 0, \quad t_1 + 2t_2 + t_3 = 8.$$

Тогда срезка дискриминанта рассматриваемого многочлена будет иметь вид:

$$\Delta_4|_{h_2} = 16a_0a_2^4a_4 - 4a_1^2a_2^3a_4 + a_1^2a_2^2a_3^2 - 4a_0a_2^3a_3^2.$$

В статье [7] доказано свойство факторизуемости срезки дискриминанта $\mathcal{N}(\Delta_n)$ на любую из некоординатных гиперграней в произведение двух дискриминантов меньших степеней. В частности, последняя срезка факторизуется в произведение двух дискриминантов квадратных уравнений:

$$\begin{aligned} &16a_0a_2^4a_4 - 4a_1^2a_2^3a_4 + a_1^2a_2^2a_3^2 - 4a_0a_2^3a_3^2 = \\ &= a_2^2(a_1^2 - 4a_0a_2)(a_3^2 - 4a_2a_4) = \\ &= a_2^2\Delta_2(a_0, a_1, a_2)\Delta_2(a_2, a_3, a_4). \end{aligned}$$

В статье [8] доказано свойство факторизуемости срезок дискриминанта на грани его многогранника Ньютона большей коразмерности. Именно, пусть

$$h_K := h_{k_1} \cap \ldots \cap h_{k_p},$$

грань $\mathcal{N}(\Delta_n)$, полученная пересечением p некоординатных гиперграней. Здесь мультииндекс $K = \{k_1, \ldots, k_p\}$ определяет разбиение набора $\{0,1,\ldots,n\}$ на $p+1$ поднаборов (отрезков)

$$K_i = \{k_i, k_i + 1, \ldots, k_{i+1}\}, \quad i = 0,1,\ldots,p,$$

считая $k_0 = 0$, $k_{p+1} = n$. Обозначим через $l_i := k_{i+1} - k_i$ длину K_i и

$$f_{K_i} := a_{k_i} + a_{k_i+1}y + \ldots + a_{k_{i+1}}y^{l_i}.$$

Приведем результат, доказанный в работе [8].

Теорема 2. *Срезка Δ_n на грань h_K факторизуется в виде произведения*

$$\Delta_n|_{h_K} = a_K^2 \prod_{i=0}^{p} \Delta_{l_i}(f_{K_i}), \qquad (3.2)$$

где $a_K^2 = a_{k_1}^2 \ldots a_{k_p}^2$, а Δ_{l_i} – дискриминанты многочленов f_{K_i} степеней l_i.

Проиллюстрируем данную теорему на примере. Для многочлена седьмой степени $f(y) = \sum_{k=0}^{7} a_k y^k$ срезка $\Delta_7|_{h_{2,5}}$ на грань $h_2 \cap h_5$ представится в виде произведения трех дискриминантов:

$$a_2^2a_5^2\Delta_2(a_0, a_1, a_2)\Delta_3(a_2, a_3, a_4, a_5)\Delta_2(a_5, a_6, a_7),$$

т.е. дискриминантов многочленов $a_0 + a_1y + a_2y^2$, $a_2 + a_3y + a_4y^2 + a_5y^3$ и $a_5 + a_6y + a_7y^2$.

4. ОПИСАНИЕ АЛГОРИТМА

Алгоритм MAIN1(n, *facets*1, *facets*2) построения срезки по определению:

1. по заданному натуральному числу n строим многочлен $f(y)$ степени n и его дискриминат *Dsc* по переменной y;

2. строим список *ListOfTerms* из мономов дискриминанта *Dsc*;

3. по заданному n формируем матрицу *MTRX* и столбец *CLMN* из коэффициентов в правой части системы (2.1);

4. из списка *ListOfTerms* выбираем мономы, у которых показатели степеней переменных $a_1, \ldots, a_{n-1}$ удовлетворяют системе (2.1) уравнений и неравенств, знаки которой задаются списком *facets*1; формируем многочлен *truncation*, являющийся искомой срезкой многочлена;

5. возвращаем факторизацию срезки, используя встроенную функцию *factor*;

6. если множество координатных граней *facets*2 непустое, то определенные в нем коэффи-

Таблица 1.

n	*facets*1	По определению	По теореме 2
3	[1]	<0.01 с	<0.01 с
4	[2]	<0.01 с	<0.01 с
5	[2]	0.094 с	<0.01 с
6	[3]	0.031 с	<0.01 с
7	[3]	0.094 с	0.016 с
8	[4]	0.750 с	<0.01 с
9	[4]	4.062 с	0.016 с
10	[5]	23.609 с	<0.01 с
11	[5]	594.688 с	0.015 с

циенты приравниваем к нолю; возвращаем факторизованную срезку на координатные грани.

Алгоритм MAIN2(n, *facets*1, *facets*2) построения срезки по Теореме 2:

1. по заданному натуральному числу n и номерам граней $facets = \{k_1,\dots,k_p\}$ строим $p+1$ многочленов f_{K_i} с коэффициентами $a_{k_i},\dots,a_{k_{i+1}}$, $i = 0,\dots,p$, причем $k_0 = 0$, $k_{p+1} = n$;

2. для каждого f_{K_i}, $i = 1,\dots,p$, находим дискриминат, используя встроенную функцию *discrim*;

3. находим приведение полученных дискриминатов и домножаем его на $a_{k_1}^2\cdots a_{k_p}^2$ согласно формуле (3.2);

4. если множество координатных граней *facets*2 непустое, то определенные в нем коэффициенты приравниваем к нолю; возвращаем факторизованную срезку на координатные грани.

Алгоритм был реализован в среде Maple 18. Полный код программы доступен по ссылке https://github.com/lyapinap/LM2022. Вычисления производились на машине Intel(R) Core(TM) i5-1135G7 CPU 2.40 GHz, 64bit, ОЗУ 8.00 Гб под управлением Windows 10.

В таблице 1 приведно сравнительное время вычисления срезок многочлена по определению и по теореме 2 с использованием команды *CPUTime* из пакета *CodeTools*.

5. ПРИМЕР

Команда

MAIN(5,[2],[])

для многочлена пятой степени

$$f = a_5y^5 + a_4y^4 + a_3y^3 + a_2y^2 + a_1y + a_0$$

и его дискриминанта $\Delta = 3125a_0^4a_5^4 - 2500a_0^3a_1a_4a_5^3 - 3750a_0^3a_2a_3a_5^3 + 2000a_0^3a_2a_4^2a_5^2 + 2250a_0^3a_3^2a_4a_5^2 - 1600a_0^3a_3a_4^3a_5 + 256a_0^3a_4^5 + 2000a_0^2a_1^2a_3a_5^3 - 50a_0^2a_1^2a_4^2a_5^2 + 2250a_0^2a_1a_2^2a_5^3 - 2050a_0^2a_1a_2a_3a_4a_5^2 + 160a_0^2a_1a_2a_4^3a_5 - 900a_0^2a_1a_3^3a_5^2 + 1020a_0^2a_1a_3^2a_4^2a_5 - 192a_0^2a_1a_3a_4^4 - 900a_0^2a_2^3a_4a_5^2 + 825a_0^2a_2^2a_3^2a_5^2 + 560a_0^2a_2^2a_3a_4^2a_5 - 128a_0^2a_2^2a_4^4 - 630a_0^2a_2a_3^3a_4a_5 + 144a_0^2a_2a_3^2a_4^3 + 108a_0^2a_3^5a_5 - 27a_0^2a_3^4a_4^2 - 1600a_0a_1^3a_2a_5^3 + 160a_0a_1^3a_3a_4a_5^2 - 36a_0a_1^3a_3^2a_5 + 1020a_0a_1^2a_2^2a_4a_5^2 + 560a_0a_1^2a_2a_3^2a_5^2 - 746a_0a_1^2a_2a_3a_4^2a_5 + 144a_0a_1^2a_2a_4^4 + 24a_0a_1^2a_3^2a_4a_5 - 6a_0a_1^2a_3^2a_4^3 - 630a_0a_1a_2^3a_3a_5^2 + 24a_0a_1a_2^3a_4^2a_5 + 356a_0a_1a_2^2a_3^2a_4a_5 - 80a_0a_1a_2^2a_3a_4^3 - 72a_0a_1a_2a_3^4a_5 + 18a_0a_1a_2a_3^3a_4^2 + 108a_0a_2^5a_5^2 - 72a_0a_2^4a_3a_4a_5 + 16a_0a_2^4a_4^3 + 16a_0a_2^3a_3^3a_5 - 4a_0a_2^3a_3^2a_4^2 + 256a_1^5a_5^3 - 192a_1^4a_2a_4a_5^2 - 128a_1^4a_3^2a_5^2 + 144a_1^4a_3a_4^2a_5 - 27a_1^4a_4^4 + 144a_1^3a_2^2a_3a_5^2 - 6a_1^3a_2^2a_4^2a_5 - 80a_1^3a_2a_3^2a_4a_5 + 18a_1^3a_2a_3a_4^3 + 16a_1^3a_3^4a_5 - 4a_1^3a_3^3a_4^2 - 27a_1^2a_2^4a_5^2 + 18a_1^2a_2^3a_3a_4a_5 - 4a_1^2a_2^3a_4^3 - 4a_1^2a_2^2a_3^3a_5 + a_1^2a_2^2a_3^2a_4^2$, выдает срезку на некоординатную гиперграни h_2 многогранника $\mathcal{N}(\Delta)$:

$$\Delta|_{h_2} = 108a_0a_2^5a_5^2 - 72a_0a_2^4a_3a_4a_5 + 16a_0a_2^4a_4^3 + 16a_0a_2^3a_3^3a_5 - 4a_0a_2^3a_3^2a_4^2 - 27a_1^2a_2^4a_5^2 + 18a_1^2a_2^3a_3a_4a_5 - 4a_1^2a_2^3a_4^3 - 4a_1^2a_2^2a_3^3a_5 + a_1^2a_2^2a_3^2a_4^2,$$

которая в свою очередь, согласно Теореме 2, факторизуется в произведение дискриминантов двух многочленов степеней два и три:

$$a_2^2(27a_2^2a_5^2 - 18a_2a_3a_4a_5 + 4a_2a_4^3 + 4a_3^3a_5 - a_3^2a_4^2)(4a_0a_2 - a_1^2) = a_2^2\Delta_2(a_0 + a_1y + a_2y^2)\Delta_3(a_2 + a_3y + a_4y^2 + a_5y^3).$$

Команда

MAIN(5,[2],[3])

находит факторизацию срезки рассматриваемого дискриминанта на грань многогранника $\mathcal{N}(\Delta)$, полученную пересечением некоординатной гиперграни h_2 с координатной h_3^0:

$$a_2^3(27a_2a_5^2 + 4a_4^3)(4a_0a_2 - a_1^2).$$

Время счета для приведенного примера составило менее 0.1 секунды.

6. БЛАГОДАРНОСТИ

Работа поддержана Красноярским математическим центром, финансируемым Минобрнауки РФ в рамках мероприятий по созданию и развитию региональных НОМЦ (Соглашение 075-02-2022-876).

СПИСОК ЛИТЕРАТУРЫ

1. *Gelfand I., Kapranov M., Zelevinsky A.* Discriminants, resultants and multidimensional determinants. Birkhäuser: Boston, 1994.
2. *Васильев В.А.* Ветвящиеся интегралы. М.: МЦНМО, 2000.
3. *Antipova I.A., Kleshkova E.A.* On Facets Of The Newton Polytope For The Discriminant Of The Polynomial System // Siberian Electronic Mathematical Reports. 2021. V. 18 (2.1). P. 1180–1188.
4. *Ardila F.* The Geometry of Matroids // Notices Amer. Math. Soc. 2018. V. 65 (8). P. 902–908.
5. *Dickenstein A., Feichtner E.M., Sturmfels B.* Tropical discriminants // J. Amer. Math. Soc. 2007. V. 20. P. 1111–1133.
6. *Krasikov V.A., Sadykov T.M.* On the analytic complexity of discriminants // Proceedings of the Steklov Institute of Mathematics. 2012. V. 279. P. 78–92.
7. *Михалкин Е.Н., Степаненко В.А., Цих А.К.* Геометрия факторизационных тождеств для дискриминантов // Доклады Российской академии наук. Серия: математика, информатика, процессы управления. 2020. Т. 493. С. 21–25.
8. *Mikhalkin E.N., Stepanenko V.A., Tsikh A.K.* Blow-ups for the Horn-Kapranov parametrization of the classical discriminant. В кн. “Partial Differential Equations, Spectral Theory, and Mathematical Physics”. EMS Series of Congress Reports. Publication house EMS. 2021. P. 315–329.
9. *Mikhalkin E.N., Nikzad M., Stepanenko V.A.* Detailed Factorization Identiies for Classical Discriminant // Journal of Siberian Federal University. Mathematics & Physics. 2022. V. 15 (1.1). P. 23–28.
10. *Passare M., Tsikh A.* Algebraic equations and hypergeometric series. In the book “The legacy of Niels Henrik Abel”. Springer: Berlin-Heidelberg-New York, 2004. P. 653–672.

ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 61–68

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 519.688

ПАКЕТ ПРОЦЕДУР И ФУНКЦИЙ ДЛЯ ПОСТРОЕНИЯ И ОБРАЩЕНИЯ АНАЛИТИЧЕСКИХ ОТОБРАЖЕНИЙ С ЕДИНИЧНЫМ ЯКОБИАНОМ

© 2023 г. Т. М. Садыков[a,*] (ORCID: 0000-0003-0741-2318)

[a] *Российский экономический университет им. Г.В. Плеханова, 117997 Москва, Стремянный пер., 36, Россия*

*****E-mail: Sadykov.TM@rea.ru*

Поступила в редакцию 19.05.2022 г.
После доработки 01.08.2022 г.
Принята к публикации 22.08.2022 г.

Множество полиномиальных отображений из n-мерного комплексного пространства в себя с постоянным ненулевым определителем матрицы Якоби является необозримо обширным для любой размерности $n > 1$. Известная гипотеза о якобиане утверждает, что любое такое отображение является полиномиально обратимым. В то время как вычисление определителя матрицы Якоби хорошо реализовано в современных системах компьютерной алгебры, обращение полиномиального отображения представляет собой задачу весьма высокой вычислительной сложности. В работе представлен пакет процедур и функций `JC` на языке программирования `Wolfram` для алгоритмического построения и обращения полиномиальных и некоторых более общих аналитических отображений с единичным определителем матрицы Якоби для заданной размерности пространства переменных и заданной степени компонент отображения. Программный код, наборы данных для его тестирования и результаты вычислительных экспериментов размещены в свободном доступе по адресу `https://www.researchgate.net/publication/358409332_JC_Package_and_Datasets`.

DOI: 10.31857/S0132347423010077, **EDN:** GSGRBF

1. ВВЕДЕНИЕ

Пусть $f = (f_1,\ldots,f_n) : \mathbb{C}^n \to \mathbb{C}^n$ – аналитическое отображение с n комплексными переменными $x = (x_1,\ldots,x_n) \in \mathbb{C}^n$, заданное в непустой области $D \subset \mathbb{C}^n$. Будем называть отображение f *якобиевым,* если определитель его матрицы Якоби есть ненулевая постоянная:

$$J(f;x)J(f_1,\ldots,f_n;x_1,\ldots,x_n) \coloneqq \begin{vmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_n}{\partial x_n} \end{vmatrix} \in \quad (1.1)$$

$$\in \mathbb{C}^* \equiv \mathbb{C}\backslash\{0\}.$$

В настоящей работе этот определитель называется *якобианом отображения f.*

Обращением отображения f называется аналитическое отображение $f^{-1} : \mathbb{C}^n \to \mathbb{C}^n$, такое, что $f \circ f^{-1} = f^{-1} \circ f = \mathrm{Id}$ в некоторой непустой области в пространстве $\mathbb{C}^n$. Область, в которой имеют место данные равенства, вообще говоря, существенным и сложным образом зависит от отображения f и области D. Все рассматриваемые в настоящей работе отображения задаются либо целыми функциями, либо функциями, допускающими аналитическое продолжение во все n-мерное комплексное пространство, за исключением некоторой особой гиперповерхности $\mathcal{H}$. Согласно теореме единственности для аналитических функций любое такое отображение (вообще говоря, многозначное) определяется любым своим ростком в окрестности произвольной неособой точки, который может быть аналитически продолжен в $\mathbb{C}^n\backslash\mathcal{H}$.

Разработка методов и алгоритмов для поиска решений уравнений и их систем различного типа в заданных классах функций (в частности, в кольце многочленов или в поле рациональных функций над заданными числовыми полями) представляет собой актуальную задачу современной компьютерной алгебры [1]. В настоящей работе представлен пакет процедур и функций `JC` на языке программирования `Wolfram` для алгоритмического построения и обращения полиномиальных и некоторых более общих аналитических отображений с постоянным ненулевым определителем матрицы Якоби для заданной размерности пространства переменных и заданной степени компонент отображения.

В одномерном случае (то есть, при $n = 1$) любое отображение, удовлетворяющее равенству (1.1), является аффинным, то есть, имеет вид $ax + b$ для некоторых $a \in \mathbb{C}^*$, $b \in \mathbb{C}$. Обратное к нему отображение также аффинно. Всюду в дальнейшем мы предполагаем, что $n \geq 2$.

Известная гипотеза о якобиане, остающаяся открытой на протяжении длительного времени (см. [7]), утверждает, что отображение $f = (f_1,\ldots,f_n)$, заданное многочленами $f_j \in \mathbb{C}[x_1, \ldots, x_n]$, является якобиевым в том и только том случае, когда обратное к нему отображение также является полиномиальным. В любой размерности $n \geq 2$ семейство всех полиномиальных отображений $f : \mathbb{C}^n \to \mathbb{C}^n$ с единичным якобианом чрезвычайно обширно и обладает весьма сложной структурой (см. [3], главы 3.5, [8] и ссылки на литературу в этих работах). Построение полиномиальных автоморфизмов заданного пространства – важное направление исследований, которому посвящены многочисленные работы, в частности, [4–6, 10].

Многочисленность безрезультатных попыток доказать гипотезу о якобиане или построить контрпример к ней, предпринимавшихся на протяжении более чем восьми десятилетий, обуславливает необходимость систематического изучения множества отображений с единичным якобианом в произвольной размерности с помощью методов и технических средств современной компьютерной алгебры. В настоящей работе представлен один из возможных подходов к разработке программного обеспечения для решения этой задачи на основе алгоритмического поиска параметризаций семейств якобиевых отображений.

Согласно фундаментальной теореме Дружковского [2] для обоснования или опровержения гипотезы о якобиане достаточно изучить ее истинность в весьма частном случае кубических отображений вида

$$\left\{(f_1,\ldots,f_n) : \mathbb{C}^n \to \mathbb{C}^n,\right.$$
$$\left. f_j(x) = x_j + \left(\sum_{k=1}^{n} a_{jk}x_k\right)^3,\ j = 1,\ldots,n\right\}. \tag{1.2}$$

Здесь (a_{jk}) – квадратная матрица размера n с комплексными элементами.

В настоящей работе изучается более широкое семейство аналитических отображений, каждое из которых задается выбором квадратной матрицы $A = (a_{jk})$ размера $n \geq 2$ и функции одного комплексного переменного $\varphi(\zeta) \in \mathbb{O}(\Omega)$, аналитической в непустой области $\Omega \subset \mathbb{C}$. Данное семейство состоит из отображений вида

$$f = (f_1,\ldots,f_n) : \mathbb{C}^n \to \mathbb{C}^n,$$
$$f[A,\varphi](x) := x + \varphi(Ax) \tag{1.3}$$

с координатами

$$f_j : x \mapsto x_j + \varphi\left(\sum_{k=1}^{n} a_{jk}x_k\right), \quad j = 1,\ldots,n,$$

якобиан которых тождественно равен ненулевой постоянной для всех x из пересечения областей определения функций $f_j(x)$. В частном случае, когда $\varphi(\zeta) = \zeta^3$, отображение (1.3) является отображением Дружковского (1.2).

При изучении отображений вида (1.3) достаточно ограничиться рассмотрением случая, когда линейная часть функции $\varphi(\cdot)$ равна нулю. В этом случае линейная часть отображения (1.3) задается единичной матрицей и оно может быть якобиевым лишь в том случае, если его якобиан тождественно равен единице. Всюду в дальнейшем мы будем говорить, что матрица A и аналитическая функция $\varphi(\zeta)$, определяющие в совокупности отображение (1.3) с единичным якобианом, образуют *хорошую пару.* Процедуры и функции пакета `JC` позволяют вычислять хорошие пары матриц и функций, исследовать их свойства и, при некоторых дополнительных условиях, обращать определяемые ими аналитические отображения вида (1.3).

Пусть U – квадратная матрица, такая, что якобиан отображения $f[U,\varphi](x)$ есть ненулевая постоянная для любого x из области определения и, вдобавок, для произвольной аналитической функции $\varphi \in \mathbb{O}(\Omega)$. Всюду в дальнейшем матрицы, обладающие данным свойством, будут называться *универсальными.*

Многочисленные компьютерные эксперименты с помощью процедур и функций пакета `JC` позволяют предполагать, что универсальная матрица размера n задается выбором целочисленного разбиения $p = (p_1,\ldots,p_m)$ размерности n на m слагаемых и перестановкой длины m однозначно с точностью до перестановочного подобия матриц и выбора значений алгебраически независимых комплексных параметров.

В настоящей работе действие аналитической функции одного переменного $\varphi(\cdot) \in \mathbb{O}(\Omega)$ на векторе комплексных переменных $\xi = (\xi_1, \ldots, \xi_n) \in \Omega\times\ldots\times\Omega \subset \mathbb{C}^n$ определяется покоординатным образом: $\varphi(\xi_1,\ldots,\xi_n) := (\varphi(\xi_1),\ldots,\varphi(\xi_n))$. Для всех $d = 2,3,\ldots$ существует n-параметрическое семейство квадратных матриц $H(s), s \in \mathbb{C}^n$, таких, что для произвольной универсальной матрицы U отображение $x + \left((U \odot H(s))x\right)^d$, определенное произведением Адамара (то есть, поэлементным произведением) $U \odot H(s)$, является якобиевым.

Любое такое отображение является полиномиально обратимым, а обратное к нему отображение может быть построено с помощью рекуррентного процесса. В настоящей работе якобиевы отображения исследуются с помощью алгоритмов, реализованных в пакете процедур и функций JC.

Все рассмотренные в настоящей работе полиномиальные отображения являются полиномиально обратимыми. Более того, для любой универсальной матрицы U и произвольной аналитической функции $\varphi(\zeta)$ обращение отображения $f[U,\varphi]$ есть конечная суперпозиция $\varphi(\zeta)$ и арифметических операций с аргументами $x = (x_1, \ldots, x_n) \in \mathbb{C}^n$ (ср. с основным результатом работы [9]). Однако, обращение якобиева отображения $x + \varphi(Ax)$, заданного аналитической функцией $\varphi(\zeta)$ и не универсальной матрицей A, не обладает, вообще говоря, этим свойством. С помощью функций пакета JC в настоящей работе построен пример якобиева отображения вида $f[A, \ln](x) := x + \ln(Ax)$, заданного матрицей Тёплица A, для которого обратное отображение $f[A,\ln]^{-1}$ не допускает представления в виде конечной суперпозиции логарифмической функции и арифметических операций (см. раздел 4.3).

2. ЯКОБИЕВЫ УРАВНЕНИЯ И ИХ ОДНОРОДНОСТИ

В свете фундаментального результата Дружковского [2] важный класс якобиевых отображений вида (1.3) образуют отображения, заданные мономиальной функцией $\varphi(\zeta) = \zeta^d$. Этот класс состоит из полиномиальных отображений вида $x + (Ax)^d$, где $d \in \mathbb{N}$, $x = (x_1, \ldots, x_n) \in \mathbb{C}^n$, $A = (a_{jk})$ – квадратная матрица размера n. Координаты данного отображения имеют вид

$$x_j + \left(\sum_{k=1}^{n} a_{jk} x_k\right)^d, \quad j = 1, \ldots, n. \tag{2.1}$$

При $d = 0$ или $d = 1$ отображение (2.1) является (аффинно) линейным, а его якобиан равен определителю соответствующей матрицы. В дальнейшем мы не рассматриваем эти тривиальные частные случаи.

Отображение (2.1) имеет единичный якобиан в том и только том случае, когда элементы матрицы A удовлетворяют некоторой системе алгебраических уравнений, зависящей от размерности n пространства переменных и степени d определяющих его многочленов. Всюду в дальнейшем мы будем использовать следующее определение.

Определение 1. Под *системой якобиевых уравнений в размерности $n \geq 2$ для степени d*, $d \in \mathbb{N}$ понимается система алгебраических уравнений с переменными a_{jk}, чьи решения определяют матрицы $A = (a_{jk})$, $j,k = 1,\ldots,n$, для которых якобиан отображения $x + (Ax)^d$ тождественно равен 1.

Например, система якобиевых уравнений в размерности 2 для степени 3 имеет вид

$$\begin{cases} a_{11}^3 + a_{21}^2 a_{22} = 0, \\ a_{11} a_{12}^2 + a_{22}^3 = 0, \\ a_{12} a_{11}^2 + a_{21} a_{22}^2 = 0, \\ a_{12}^2 a_{22}^2 \det A = 0, \\ a_{11}^2 a_{21}^2 \det A = 0, \\ a_{12} a_{22} \det A \cdot \operatorname{perm} A = 0, \\ a_{11} a_{21} \det A \cdot \operatorname{perm} A = 0, \\ \det A (a_{12}^2 a_{21}^2 + 4 a_{11} a_{12} a_{22} a_{21} + a_{11}^2 a_{22}^2) = 0. \end{cases} \tag{2.2}$$

Здесь $\operatorname{perm} A = a_{11}a_{22} + a_{12}a_{21}$ – перманент матрицы $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$.

Общая система якобиевых уравнений имеет труднообозримую структуру. Число образующих ее уравнений и их степень быстро растут с увеличением размерности пространства переменных и степени полиномиального отображения (2.1). Несмотря на это, любая система якобиевых уравнений содержит некоторую подсистему, образованную уравнениями, структура которых весьма прозрачным образом зависит от n и d. А именно, пусть $A = (a_{jk})$ – квадратная матрица размера $n \geq 2$. Для любого $d = 2,3,\ldots$ система якобиевых уравнений в размерности n для степени d содержит подсистему *простых якобиевых уравнений:*

$$(A^T)^{\odot(d-1)} \operatorname{diag} A = 0. \tag{2.3}$$

Здесь A^T обозначает транспонированную матрицу, $\operatorname{diag} A$ – вектор диагональных элементов матрицы A. Например, в двумерном случае для степени 2 подсистема простых якобиевых уравнений образована первыми двумя уравнениями в системе (2.2).

Под однородностями якобиева отображения $x + (Ax)^d$ мы будем всюду в дальнейшем понимать произвольную квадратную матрицу H того же размера, что и матрица A, такую, что отображение $x + (A \odot H)^d$ также является якобиевым. Здесь через $A \odot H$ обозначается произведение Адамара (то есть, поэлементное произведение) матриц A и H.

3. ОБЗОР ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПАКЕТА JC

Пакет JC для системы компьютерной алгебры Mathematica содержит функции для построе-

Таблица 1. Время вычисление якобиана отображения $x + (Ux)^d$

d	10	11	12	13	14
Время, с.	1.67	2.57	4.07	5.51	7.54

Таблица 2. Время обращения отображения (4.4) с помощью функции `Solve` СКА `Mathematica`

d	3	4	5	6	7
Время, с.	1.65	13.26	71.1	287.67	991.23

ния, параметризации, обращения и изучения свойств аналитических отображений с единичным якобианом. Исходный код этих функций, а также библиотека наборов данных для их тестирования и данных, полученных в результате многочисленных компьютерных экспериментов, размещены в репозитории `https://www.researchgate.net/publication/358409332_JC_Package_and_Datasets`.

Пакет состоит из трех основных блоков процедур и функций. Первый из них образуют функции, реализующие операции и объекты линейной алгебры, не поддержанные стандартными функциями ядра системы: функции выявления перестановочного подобия числовых и символьных матриц, вычисления главных миноров квадратной матрицы и сумм этих миноров, функции для построения матриц Вандермонда и циркулянтов, умножения матриц по Адамару и подобные им.

Функции из второго блока обеспечивают алгоритмическое построение полиномиальных и аналитических отображений с единичным якобианом. Одна из центральных функций данного блока позволяет строить общую универсальную матрицу в заданной размерности, определенную целочисленным разбиением этой размерности и перестановкой, чья длина равна мощности разбиения. Второй блок содержит также функции для вычисления системы якобиевых уравнений для заданных размерности и степени, построения матриц, определяющих однородности данной системы, построения подсистемы простых якобиевых уравнений, а также уравнений, чьи решения образуют хорошие пары с логарифмической функцией.

Третий блок функций пакета позволяет обращать полиномиальные и аналитические отображения вида $f(x) = x + \varphi(Ax)$ с единичным якобианом, заданные квадратной матрицей A и аналитической функцией одного переменного $\varphi(\zeta)$.

4. ВЫЧИСЛЕНИЕ СИСТЕМЫ ЯКОБИЕВЫХ УРАВНЕНИЙ, ПОИСК ХОРОШИХ ПАР И ОБРАЩЕНИЕ ЯКОБИЕВЫХ ОТОБРАЖЕНИЙ

Настоящий раздел иллюстрирует возможности процедур и функций пакета `JC` для построения якобиевых отображений в различных размерностях. Разработанный программный код позволяет строить как полиномиальные, так и более общие якобиевы отображения, заданные аналитическими функциями. Сложность структуры множества полиномиальных якобиевых отображений весьма быстро растет как с увеличением размерности пространства переменных, так и с ростом степени определяющих их многочленов. Результаты расчетов могут быть представлены в статье лишь в случае достаточно низкой размерности, так как с ее ростом они становятся необозримо громоздкими. По этой причине мы в первую очередь рассматриваем здесь кубические отображения четырех независимых комплексных переменных и отсылаем читателя к процедурам и функциям пакета, а также библиотеке наборов данных и результатов компьютерных экспериментов в более высоких размерностях.

Обозначим через $A = (a_{jk})$, $j,k = 1,\ldots,4$ произвольную квадратную матрицу размера 4×4 и через $d = 3$ – степень многочленов, определяющих отображение

$$f = (f_1,\ldots,f_4) : \mathbb{C}^4 \to \mathbb{C}^4, \quad f(x) := x + (Ax)^3. \tag{4.1}$$

С помощью функции `JEquations[A,d]` вычисляем систему якобиевых уравнений, которой элементы матрицы A удовлетворяют в том и только том случае, когда якобиан отображения (4.1) тождественно равен 1. Вычисление данной системы однородных алгебраических уравнений с неизвестными a_{jk} занимает 268.43 секунды. Система содержит 294 уравнения и не может быть полностью помещена в настоящей статье. Наибольшая из степеней входящих в нее уравнений по совокупности переменных a_{jk} равна 48.

4.1. Подсистема простых якобиевых уравнений

Используя функцию `simpleJEquations[A,d]`, находим следующую подсистему простых якобиевых уравнений в размерности 4 для степени 3:

$$\begin{cases} a_{11}^3 + a_{21}^2 a_{22} + a_{31}^2 a_{33} + a_{41}^2 a_{44} = 0, \\ a_{11} a_{12}^2 + a_{22}^3 + a_{32}^2 a_{33} + a_{42}^2 a_{44} = 0, \\ a_{11} a_{13}^2 + a_{22} a_{23}^2 + a_{33}^3 + a_{43}^2 a_{44} = 0, \\ a_{11} a_{14}^2 + a_{22} a_{24}^2 + a_{33} a_{34}^2 + a_{44}^3 = 0. \end{cases} \tag{4.2}$$

В то время как структура множества решений полной системы якобиевых уравнений является, судя по имеющимся в настоящий момент дан-

ным, весьма сложной в любой размерности, превосходящей 2, многочисленные компьютерные эксперименты с использованием алгоритмов, реализованных в пакете JC, позволяют предполагать, что антидиагональные элементы матрицы (a_{jk}) могут быть выражены через остальные ее элементы. Результаты этих экспериментов являются частными случаями следующего утверждения.

Гипотеза 1. *Множество решений простых якобиевых уравнений в размерности n для степени d имеет размерность $n^2 - n$ и допускает алгебраическую параметризацию, в которой независимыми параметрами являются элементы матрицы (a_{jk}), не лежащие на ее антидиагонали. Для четных размерностей n при $d = 2$ данная параметризация является рациональной.*

Например, множество решений простых якобиевых уравнений (4.2) в размерности 4 для степени 3 допускает следующую алгебраическую параметризацию:

$$\begin{aligned} a_{14} &= -i\frac{\sqrt{a_{44}^3 + a_{22}a_{24}^2 + a_{33}a_{34}^2}}{\sqrt{a_{11}}},\\ a_{23} &= -i\frac{\sqrt{a_{33}^3 + a_{11}a_{13}^2 + a_{43}^2a_{44}}}{\sqrt{a_{22}}},\\ a_{32} &= -i\frac{\sqrt{a_{22}^3 + a_{11}a_{12}^2 + a_{42}^2a_{44}}}{\sqrt{a_{33}}},\\ a_{41} &= -i\frac{\sqrt{a_{11}^3 + a_{21}^2a_{22} + a_{31}^2a_{33}}}{\sqrt{a_{44}}}, \end{aligned} \qquad (4.3)$$

в которой все элементы матрицы A, за исключением антидиагональных (то есть, те элементы a_{jk}, для которых $j + k \neq 5$), играют роль алгебраически независимых параметров, $i = \sqrt{-1}$. Непосредственная проверка показывает, что элементы любой квадратной матрицы размера 4, заданные в виде (4.3), удовлетворяют соответствующей системе простых якобиевых уравнений (4.2).

Гипотеза 1 проверена с помощью функций пакета JC для всех значений размерности пространства переменных и степени определяющих отображение многочленов $n, d \leq 10$. Решая простые якобиевы уравнения (2.3) относительно антидиагональных элементов матрицы A, можно построить семейства алгебраических или рациональных решений полной системы якобиевых уравнений. Например, утверждение гипотезы в размерности 4 позволяет сделать вывод о том, что матрица

$$M = \begin{pmatrix} 1 & 0 & t & -t-1\\ 0 & 1 & -t-1 & t\\ -\dfrac{t}{2t+1} & -\dfrac{t+1}{2t+1} & 1 & 0\\ -\dfrac{t+1}{2t+1} & -\dfrac{t}{2t+1} & 0 & 1 \end{pmatrix},$$

зависящая от параметра $t \in \mathbb{C}$, образует хорошую пару с функцией $\varphi(\zeta) = \zeta^2$. Непосредственная проверка с помощью функции `JMatrDeg[M,2]` показывает, что это на самом деле так.

4.2. Универсальные матрицы, однородности и вычисление обратных отображений

Вычисление однородностей якобиевых уравнений в размерности 4 для степени 3 с помощью команды `HomogeneitiesOfJEquations [4, 3]` пакета JC дает матрицу

$$H_{4,3}(s_1,\ldots,s_4) = \begin{pmatrix} s_1^3s_2s_3s_4 & s_2^4s_3s_4 & s_2s_3^4s_4 & s_2s_3s_4^4\\ s_1^4s_3s_4 & s_1s_2^3s_3s_4 & s_1s_3^4s_4 & s_1s_3s_4^4\\ s_1^4s_2s_4 & s_1s_2^4s_4 & s_1s_2s_3^3s_4 & s_1s_2s_4^4\\ s_1^4s_2s_3 & s_1s_2^4s_3 & s_1s_2s_3^4 & s_1s_2s_3s_4^3 \end{pmatrix}.$$

Используя функцию `allUniversalMatrices` пакета JC, находим, что существует 49 различных (для параметров общего положения) универсальных матриц размера 4 (см. таблицу 3). Обозначим через p целочисленное разбиение (2, 2) размерности $n = 4$ и через $\varepsilon = (1,2)$ тривиальную перестановку на множестве из двух элементов. С помощью функции `universalMatrix[p, ε]` вычисляем соответствующую универсальную матрицу, которую, переобозначая ее элементы для исключения громоздких индексов, можно представить в следующем виде:

$$U = \begin{pmatrix} a & -a & b & c\\ a & -a & b & c\\ u & -u & v & -v\\ u & -u & v & -v \end{pmatrix}.$$

Данная матрица нильпотентна: $U^3 = 0$. Непосредственная проверка показывает, что ее элементы удовлетворяют системе якобиевых уравнений в размерности 4 для степени 3 и, в частности, ее подсистеме (4.2).

С помощью команды `allSumsOfPrincipalMinors[U]` мы убеждаемся в том, что сумма главных миноров любого порядка $k = 1,\ldots,4$ матрицы U равна нулю. Команда `universalMatrixQ[U]` позволяет проверить, что матрица U действительно является универсальной, то есть, что якобиан отображения

$$f = (f_1,\ldots,f_4) : \mathbb{C}^4 \to \mathbb{C}^4,$$
$$f(x) := x + \varphi(Ux) \tag{4.4}$$

тождественно равен 1 для произвольной аналитической функции одного переменного $\varphi(\zeta)$, такой, что координаты отображения (4.4) корректно определены. Отметим, что время непосредственного вычисления якобиана отображения $x + (Ux)^d$ быстро растет с увеличением степени d, см. таблицу 1.

Несмотря на это, запуск функции `NewtonInverseGeneral` позволяет заключить, что обращение отображения (4.4) может быть представлено в виде

$$x(f) = f - \varphi(f - \varphi(Uf)). \tag{4.5}$$

Заметим, что непосредственное обращение отображения (4.4) с помощью стандартной функции `Solve` в системе компьютерной алгебры `Mathematica` является задачей высокой вычислительной сложности, см. таблицу 2.

4.3. Случай аналитических отображений

Процедуры и функции пакета JC позволяют вычислять матрицы, образующие хорошие пары с заданными неполиномиальными аналитическими функциями, а также с произвольными аналитическими функциями одного переменного. Например, результатом работы команды `JEquationsLOG[A]` является следующая система из 35 однородных алгебраических уравнений с неизвестными a_{jk}, решения которой определяют всевозможные матрицы A, образующие хорошие пары с логарифмической функцией, то есть, такие, для которых отображение $x + \ln(Ax)$ имеет единичный якобиан. Одно из семейств решений данной системы уравнений состоит из нильпотентных матриц вида

$$\begin{pmatrix} 1 & a_{12} & a_{13} & -1 \\ a_{21} & 1 & -1 & -a_{21} \\ a_{21} & 1 & -1 & -a_{21} \\ 1 & a_{12} & a_{13} & -1 \end{pmatrix},$$

где a_{12}, a_{13} и a_{21} – произвольные комплексные параметры.

Многочисленные компьютерные эксперименты показывают, что другое важное семейство матриц, образующих хорошие пары с логарифмической функцией, состоит из матриц Тёплица некоторого специального вида. Сужая систему уравнений, построенную с помощью команды `JEquationsLOG[A]`, на случай матриц Тёплица (то есть, предполагая, что $a_{jk} = a_{pq}$ для всех j, k, p и q, таких, что $j - k = p - q$) и дополнительно предполагая, что $a_{11} = 0$, мы приходим к выводу, что любая матрица в данном семействе, образующая хорошую пару с логарифмической функцией, пропорциональна матрице

Таблица 3. Количество и время вычисления всех универсальных матриц размера $2 \le n \le 10$

Размерность n пространства переменных	Число всех унив. матриц размера n	Время расчета, с.
2	3	<0.01
3	11	<0.01
4	49	<0.01
5	261	0.06
6	1631	0.45
7	11743	4.67
8	95901	52.37
9	876809	610.57
10	8877691	9551.02

$$T_{\ln} = \begin{pmatrix} 0 & 1 & -1+i & -i \\ -i & 0 & 1 & -1+i \\ -1+i & -i & 0 & 1 \\ 1 & -1+i & -i & 0 \end{pmatrix}$$

или же комплексно сопряженной с ней матрице. Здесь $i = \sqrt{-1}$.

Соответствующее якобиево отображение $x + \ln(T_{\ln}x)$ не допускает элементарного обращения. В частности, в отличие от всех рассмотренных выше случаев, обратное отображение не является конечной суперпозицией логарифмической функции и арифметических операций. Отметим, что $T_{\ln}^3 + 8iT_{\ln} = 0$. Применение функции `permuteRowsAndColumns` позволяет заключить, что любая матрица, перестановочно подобная матрице $T_{\ln}$, также образует хорошую пару с логарифмической функцией. Вопрос об алгоритмическом описании множества всех матриц Тёплица, образующих хорошую пару с заданной аналитической функцией, является, по-видимому, открытым.

Отметим, что в трехмерном случае матрица

$$\begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

образует хорошую пару с логарифмической функцией. Обращение соответствующего якобиева отображения требует решения трансцендентной системы уравнений

$$\left(\frac{\eta}{\zeta}\right)^{\xi} = s, \quad \left(\frac{\zeta}{\xi}\right)^{\eta} = t, \quad \left(\frac{\xi}{\eta}\right)^{\zeta} = u$$

относительно переменных ξ, η, ζ.

4.4. Многомерный случай: семейство матриц ранга 2, образующих хорошие пары с функцией возведения в квадрат

Параметризация множества всех якобиевых отображений вида $x + (Ax)^d$ в произвольной размерности n и для произвольной степени d (здесь, как и ранее, $x = (x_1,\ldots,x_n) \in \mathbb{C}^n$ – вектор комплексных переменных, A – квадратная матрица размера n) находится, по-видимому, далеко за пределами современных возможностей компьютерной алгебры. Несмотря на это, процедуры и функции пакета `JC` позволяют строить и изучать свойства семейств якобиевых отображений в произвольной размерности, удовлетворяющих ряду дополнительных предположений.

Для целого положительного ℓ введем обозначения $\overline{s} = (s_1,\ldots,s_\ell) \in \mathbb{C}^\ell$, $\overline{t} = (t_1,\ldots,t_\ell) \in \mathbb{C}^\ell$ и положим $|\overline{s}| := s_1 + \ldots + s_\ell$. С помощью функций `JacobianEquations` и `allPrincipalMinors`, входящий в пакет `JC`, сформируем следующую квадратную матрицу размера $\ell + 2$:

$$M(\overline{s},\overline{t}) := \begin{pmatrix} -|\overline{s}| - |\overline{t}| & -\dfrac{|\overline{s}|(|\overline{s}| + |\overline{t}|)}{|\overline{t}|} & \overline{s} \\ -\dfrac{|\overline{t}|(|\overline{s}| + |\overline{t}|)}{|\overline{s}|} & -|\overline{s}| - |\overline{t}| & \overline{t} \\ -\dfrac{(|\overline{s}| + |\overline{t}|)^2}{|\overline{s}|} & -\dfrac{(|\overline{s}| + |\overline{t}|)^2}{|\overline{t}|} & \overline{s} + \overline{t} \\ \cdots & \cdots & \cdots \\ -\dfrac{(|\overline{s}| + |\overline{t}|)^2}{|\overline{s}|} & -\dfrac{(|\overline{s}| + |\overline{t}|)^2}{|\overline{t}|} & \overline{s} + \overline{t} \end{pmatrix}.$$

Последние ℓ строк матрицы $M(\overline{s},\overline{t})$ одинаковы и равны сумме первых двух ее строк. Команда `JMatrDeg` позволяет осуществить непосредственную проверку того факта, что матрица $M(\overline{s},\overline{t})$ образует хорошую пару с функцией $\varphi(\zeta) = \zeta^2$. Соответствующее якобиево отображение $x + (M(\overline{s},\overline{t})x)^2$ может быть обращено с помощью функции `NewtonInverse`, однако результат ее работы слишком громоздок для включения в текст статьи. Мы отсылаем читателя к результатам компьютерных экспериментов, представленных на сайте пакета `JC`.

Таблица 4. Время обращения якобиева отображения с помощью функций `Solve` и `NewtonInverse`

Разбиение p	Время работы функции `Solve`, с.	Время работы функции `NewtonInverse`, с.
{1, 2}	0.04	0.01
{1, 3}	0.23	0.04
{1, 4}	1.48	0.31
{2, 2}	0.26	0.10
{2, 3}	1.34	0.07
{2, 4}	7.48	0.35
{2, 5}	21.56	1.21
{3, 5}	68.23	4.34
{1, 1, 2}	0.28	0.01
{1, 1, 3}	9.68	0.10
{1, 1, 4}	184.28	1.04
{1, 2, 2}	12.68	0.09
{1, 2, 3}	322.23	1.00
{2, 2, 2}	182.17	3.64

5. ОБОРУДОВАНИЕ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ОЦЕНКА СКОРОСТИ РАБОТЫ ПРОГРАММНОГО КОДА

Представленные в настоящей работе расчеты выполнены в системе компьютерной алгебры `Wolfram Mathematica 11.3` на рабочей станции HP Z Workstation с центральным процессором Intel Xeon Gold 6146 с тактовой частотой 3.20 ГГц и 128 Гб оперативной памяти.

Число различных универсальных матриц быстро растет с увеличением размерности. В таблице 3 приведены результаты компьютерных экспериментов с функцией `allUniversalMatrices` для всех $n \le 10$. Каждая из найденных универсальных матриц определяет n-параметрическое семейство якобиевых отображений вида (2.1), полученных путем взятия произведения Адамара с матрицей однородностей, которая может быть построена с помощью команды `HomogeneitiesOfJEquations`. В следующей таблице время обращения якобиева отображения, заданного универсальной матрицей в степени $d = 2$, с помощью функции NewtonInverse пакета `JC` сравнивается с временем, которое требуется стандартной функции Solve системы компьютерной алгебры `Mathematica 11.3` для решения этой же задачи. Универсальная матрица здесь определяется целочисленным разбиением p размерности пространства переменных и тождественной перестановкой. Время работы узкоспециализированного алгоритма, учитывающего ключевые свойства якобиева отображения, ожидаемо значительно меньше времени, которое требуется для решения

этой задачи с помощью функции общего назначения.

6. БЛАГОДАРНОСТИ

Данное исследование выполнено в рамках государственного задания в сфере научной деятельности Министерства науки и высшего образования Российской Федерации, номер проекта FSSW-2020-0008.

СПИСОК ЛИТЕРАТУРЫ

1. *Абрамов С.А.* Поиск рациональных решений дифференциальных и разностных систем с помощью формальных рядов // Программирование. 2015. № 2. С. 69–80.
2. *Drużkowski L.M.* An effective approach to Keller's Jacobian Conjecture // Math. Ann. 1983. № 264. P. 303–313.
3. *van den Essen A.* Polynomial Automorphisms and the Jacobian Conjecture. Birkhäuser, 2000.
4. *van den Essen A. and Washburn S.* The Jacobian Conjecture for symmetric Jacobian matrices // Journal of Pure and Applied Algebra. 2004. № 189. P. 123–133.
5. *Fernandes F.* A new class of non-injective polynomial local diffeomorphisms on the plane // Journal of Mathematical Analysis and Applications. 2022. № 507. 125736.
6. *Grigoriev D. and Radchenko D.* On a tropical version of the Jacobian Conjecture // Journal of Symbolic Computation. 2022. № 109. P. 399–403.
7. *Keller O.H.* Ganze Cremona-Transformationen // Monatshefte für Mathematik und Physik. 1939. № 47. P. 299–306.
8. *Peretz R.* The 2-dimensional Jacobian Conjecture: A computational approach // Algorithmic Algebraic Combinatorics and Gröbner Bases. 2009. P. 151–203.
9. *Stepanova M.A.* Jacobian conjecture for mappings of a special type in $\mathbb{C}^2$ // Journal of Siberian Federal University. Mathematics & Physics. 2018. № 11(2.3). P. 776–780.
10. *Truong T.T.* Some new theoretical and computational results around the Jacobian Conjecture // International Journal of Mathematics. 2020. № 31(4.1). 2050050.