СОДЕРЖАНИЕ

Номер 1, 2022

КОМПЬЮТЕРНАЯ АЛГЕБРА

CONTENTS

No. 1, 2022

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 3–22

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.421.6

ДОКАЗАТЕЛЬНАЯ ПРОГРАММА ДЛЯ СПОСОБА КАРАЦУБЫ УМНОЖЕНИЯ МНОГОЧЛЕНОВ

© 2022 г. С. Д. Мешвелиани[a,*]

[a] Институт программных систем им. А.К. Айламазяна РАН,
ул. Петра Первого, "а", с. Веськово, Переславский р-н, 152021 Ярославская обл., Россия
**E-mail: mechvel@scico.botik.ru*

Поступила в редакцию 28.05.2021 г.
После доработки 17.06.2021 г.
Принята к публикации 01.09.2021 г.

Представлена разработка доказательной программы для способа Карацубы умножения многочленов одной переменной. Построено машинно-проверяемое доказательство равносильности функции способа Карацубы простейшему умножению многочленов "раскрытием скобок". Это малая часть библиотеки `DoCon-A` машинно-проверяемых программ для вычислительной алгебры, созданной автором. Утверждение доказано для произвольного коммутативного кольца коэффициентов, а в системе `DoCon-A` задано доказательное построение структуры коммутативного кольца для области `R[x]` многочленов над коммутативным кольцом R. В системе `DoCon-A` программы включают определения соответствующих математических понятий и доказательства главных свойств применяемых алгоритмов. Эти доказательства проверяются компилятором. Применяется чисто функциональный язык программирования `Agda`, который также предоставляет конструкцию зависимых типов. Также рассматриваются некоторые общие вопросы построения доказательных программ в алгебре на языке `Agda`. Делается сравнение с подходом другого автора к построению в системе `Coq` некоторой разновидности алгоритма Карацубы.

DOI: 10.31857/S0132347422010071

1. ВВЕДЕНИЕ

Словоупотребление. Мы используем следующее словоупотребление.

Мп-доказательство – машинно-проверяемое полное формальное доказательство, воплощение – реализация или модель теории, свидетельство – доказательство. Слово "библиотека" обозначает библиотеку `DoCon-A` [1] доказательных программ компьютерной алгебры. Имя `Agda` иногда пишем кириллицей и склоняем по падежам.

Простейшим способом умножения целых чисел, представленных списком разрядов, является общеизвестный способ "в столбик". В худшем случае для умножения чисел из n двоичных разрядов он требует порядка n^2 действий над разрядами. В 1960 году А.А. Карацуба нашел более быстрый способ умножения [2, 3]. Его стоимость вычисления имеет порядок $O(n^{\log_2 3})$. Позже был открыт еще более быстрый (асимптотически) способ [4], основанный на быстром преобразовании Фурье. Способ Карацубы также переносится на многочлены: [5], раздел 8.1.

Здесь мы занимаемся способом Карацубы, так как он проще выражается, имеет низкий порог выгодного применения, и до него дошла очередь.

Более подробно: статья относится главным образом к предмету "способы программирования вычислительной алгебры". В программных библиотеках (системах) вычислительной алгебры есть такая разновидность библиотек: доказательные (сертифицированные) – в них методы снабжены мп-доказательствами. Автором разработана такая библиотека `DoCon-A`. Естественно, в каждой такой библиотеке все (как можно больше) включенные методы должны иметь доказательные программы. Для каждого значимого метода построение доказательной программы – это отдельная задача. В данном случае это задача построения доказательной программы для способа Карацубы умножения многочленов. Описание принципов построения библиотеки в целом могло бы быть предметом другой статьи. Если главные свойства умножения многочленов не снабжены мп-доказательствами, то тогда и почти все более сложные методы для многочленов тоже окажутся не полностью мп-доказанными (вычисление НОД, разложение на неприводимые, и так далее), так как доказательства их свойств опираются на доказательства для свойств арифметических действий для многочленов.

Кроме того, эта задача используется как площадка для проверки подходов к доказательному программированию алгебры вообще: эти же приемы могут быть использованы в других задачах. При этом проверяются способы использования средств языка Agda, подходы и практика применения отличаются от тех, что наработаны, например, в системе Coq (см. разделы 4, 6–8). Насколько знает автор, DoCon-A это пока что единственная не игрушечная общая библиотека компьютерной алгебры на языке Agda.

О единице стоимости вычисления: предполагается, что стоимость вычисления измеряется количеством простейших действий над мономами: умножение мономов, сложение коэффициентов, сравнение коэффициента с нулем, сравнение показателей мономов.

Эта оценка $O(n^{\log_2 3})$ предполагает такое измерение стоимости. Здесь мы делаем такое же допущение.

В каких случаях это предположение является наиболее естественным с точки зрения оценки времени исполнения программы? Например, для случая коэффициентов из области $\mathbb{Z}/(b)$ остатков целых чисел по модулю некоторого b. Для этой области проводились испытания программы (раздел 6).

Программа испытывалась на возведении многочлена $x+1$ в степень 2^n и на умножении случайных многочленов.

За определение умножения многочленов принимается умножение простейшим способом “раскрытия скобок”, когда умножение $f * g$ многочленов сводится к сложению произведений $m * g$, для всех мономов из f, а умножение $m * g$ выстраивается как список произведений монома m на мономы многочлена g. При представлении многочлена списком мономов с условием упорядоченности показателей по убыванию и некотором естественном способе сложения многочленов получается, что простейшее умножение многочленов степени n выполняет $O(n^2)$ умножений и сравнений мономов, и это оценка точная.

В библиотеке DoCon-A ([1], выпуск 3.2rc) построена структура коммутативного кольца для многочленов над произвольным коммутативным кольцом коэффициентов, вместе с соответствующими машинно-проверяемыми доказательствами. При этом доказательства построены для определения произведения многочленов через простейший способ умножения.

Целью представленного здесь исследования было построение такой функциональной программы для способа Карацубы умножения многочленов, которая

- включает в себя мп-доказательство его правильности,
- обладает необходимым быстродействием на практике, соответствующим оценке $O(n^{\log_2 3})$,
- выражена исходным кодом приемлемого объема,
- компилируется за приемлемое время.

Правильность означает, что эта функция выдает результат, заданный определением умножения. Из доказательства такой равносильности легко строятся формальные доказательства главных свойств арифметики многочленов для способа Карацубы: законы переместительный, сочетательный, распределительный и другие. Ибо эти законы раньше формально доказаны для простейшего умножения.

Последнее условие о затратах компиляции включено по следующей причине. Компиляцией мы здесь называем такие действия системы Agda: а) проверку типов и б) порождение исполняемого кода. Проверка типов включает в себя проверку мп-доказательств путем проведения некоторых символьных вычислений с выражениями типов. Например, если программа содержит мп-доказательства утверждений из учебника алгебры объемом сто страниц, то для компилятора будет неприемлемо затратить неделю времени на проверку этих мп-доказательств.

Последовательность изложения такова.

В разделе 2 кратко даются общие сведения о представлении арифметики многочленов в программе.

В разделе 3 описывается алгоритм Карацубы для плотного и разреженного представления многочлена.

В разделе 4 даются предварительные общие сведения о программировании на языке с зависимыми типами, о возможном влиянии мп-доказательств в программе на производительность исполняемого кода.

В разделе 5 описывается подход к построению доказательной программы на языке Agda для способа Карацубы для разреженного представления многочлена, обсуждаются некоторые ее черты.

В разделе 6 описывается испытание на производительность функции karatsuba на примере возведения многочлена в степень 2^n двоичным способом и на других примерах, сравнение с программой на языке Haskell.

В разделе 7 обсуждаются издержки доказательного программирования на примере данной программы.

В разделе 8 наш подход к доказательному программированию способа Карацубы для многочленов сравнивается с подходом, описанным в диссертации [6] (с программированием в системе

Coq). Также говорится о других работах в области автоматизированных систем поддержки формальных доказательств.

В Приложении 11.2 дается доказательство оценки $O(n^{\log 3})$ стоимости вычисления способом Карацубы произведения многочленов для разреженного представления и нашей разновидности алгоритма. Также обсуждается оценка средней стоимости вычисления для этого алгоритма в зависимости от заданной степени разреженности 11.3.

1.1. Программа

Библиотека DoCon-A доказательных программ доступна в Интернете по адресу http://www.botik.ru/pub/local/Mechveliani/docon-A/3.2rc/

Программы написаны на языке Agda [9, 10]. Доказательная функция метода Карацубы содержится в модуле Pol/Karatsuba.agda, программа испытания содержится в модуле Pol/KTest.agda, установка библиотеки и запуск испытания описаны в файле install.txt архива библиотеки.

2. ОБ АРИФМЕТИКЕ МНОГОЧЛЕНОВ

Представляем моном (тип Mon в программе) в виде записи, содержащей коэффициент – элемент носителя (тип C) кольца R коэффициентов и показатель (тип $\mathbb{N}$ – натуральное число). Многочлен представлен в виде списка мономов, чьи коэффициенты ненулевые, а показатели упорядочены по убыванию. Например, многочлен $-2x^4+1$ над целыми числами представлен в программе выражением

```
mkPol ((mkMon -2 4) :: (mkMon 1 0) ::
                    []) nzCoefs ordExps
```

nzCoefs – это свидетельство неравенства нулю коэффициентов,

ordExps – свидетельство упорядоченности списка показателей 4 :: 0 :: [].

Равенство _=p_ многочленов определено как почленное равенство списков мономов, когда равенство коэффициентов выражено абстрактным равенством _≈_ в кольце R. Сложение многочленов выполнено как сложение их списков мономов: сочетание слияния двух упорядоченных списков мономов с приведением подобных членов и удалением получающихся нулевых мономов. Этот известный способ сложения $f+g$ многочленов затрачивает количество действий не больше $\max(\deg f)(\deg g)$.

Умножение f * g многочленов определяется как “раскрытие скобок”: сложение произведений m * g для всех мономов m из f. А умножение m * g на моном выстраивается как список произведений монома m на мономы многочлена g. При этом учитывается, что кольцо R может иметь делители нуля, и появляющиеся нулевые мономы удаляются.

Для таким образом определенной арифметики многочленов в библиотеке DoCon-A доказаны законы коммутативного кольца для области многочленов R[x]. Соответствующий модуль имеет заголовок

```
module Pol.Over-decComRing
  (R : DecCommutativeRing)
  (open DecCommutativeRing R using
  (Carrier))
  (C-Show : Show Carrier) (C-Read :
   Read Carrier)
  (var : Variable)
  where
```

Этот модуль имеет четыре параметра:

R – коммутативное кольцо коэффициентов (приставка Dec означает разрешимое равенство, то есть задана функция распознавания равенства на носителе R),

C-Show – структура, задающая способ распечатки в строку элемента R,

C-Read – структура, задающая способ чтения из строки элемента R,

var – строка, изображающая переменную в распечатке многочлена.

В этом модуле запрограммированы функции арифметики многочленов и доказательства для них. Например, функция

```
polDecCommutativeRing : DecCommutativeRing
```

строит запись (record), представляющую структуру коммутативного кольца с разрешимым равенством для многочленов. То есть это функтор, строящий кольцо многочленов из данного коммутативного кольца R коэффициентов.

Чтобы воспользоваться функциями построенной области многочленов, программа должна открыть модуль, дав ему значения параметров. Например, арифметика целочисленных многочленов откроется по объявлению

```
open Pol.Over-decComRing Int.decCommutativeRing
                         Int.Show Int.Read "x"
```

Компилятор Агды проверит, содержит ли структура Int.decCommutativeRing доказательства структуры коммутативного кольца, и так далее. В области действия такого объявления компилятор воспримет функции +p, *p из этого модуля как действия над многочленами с целыми коэффициентами. Если же, например, подставлено значение параметра R = IntResidue, то эти функции будут восприниматься в смысле арифметики многочленов с коэффициентами по модулю заданного m (коэффициенты из $\mathbb{Z}/(m)$).

3. УМНОЖЕНИЕ МНОГОЧЛЕНОВ СПОСОБОМ КАРАЦУБЫ

В дальнейшем применяем следующие определения и словоупотребление.

log обозначает логарифм по основанию 2.

Определение 1. Для метода M вычисления некоторой функции для пар многочленов выражение $C(M,n)$ обозначает наибольшую из стоимостей применения этого метода для многочленов степени не больше n, когда стоимость выражена числом простейших действий над мономами (см. начало Введения).

Оценка стоимости $M \in O(n^\alpha)$ для такого метода M означает, что существует натуральное число c такое, что для любого $n > 0$ выполнено неравенство

$$C(M,n) \le cn^\alpha.$$

3.1. Случай плотного представления многочлена

Пусть deg f = deg g = 2k, где k степень двойки. Представим

$$f = x^k f_1 + f_2, \quad g = x^k g_1 + g_2,$$

где $\deg f_1 = \deg f_2 = \deg g_1 = \deg g_2 = k$. Тогда

$$f * g = x^{2k} f_1 g_1 + x^k * (f_1 g_2 + f_2 g_1) + f_2 g_2 \qquad \text{(I)}$$

В этой формуле умножение на моном вида x^i и сложения имеют линейную стоимость по k. Но появляются четыре умножения многочленов степени k, и при простейшем способе эти умножения имеют стоимость квадратичную по k.

Учитывая равенство

$$f_1 g_2 + f_2 g_1 = (f_1 + f_2) * (g_1 + g_2) - f_1 g_1 - f_2 g_2$$

и обозначая $s = f_1 + f_2$, $s' = g_1 + g_2$, запишем (I) как

$$f * g = x^{2k} f_1 *_k g_1 + x^k * (s *_k s' - f_1 *_k g_1 - f_2 *_k g_2) + f_2 *_k g_2, \qquad \text{(II)}$$

где $*_k$ это умножение рекурсивным применением способа (II). Теперь имеем только три умножения многочленов степени k:

$$f_1 *_k g_1, \quad f_2 *_k g_2, \quad s *_k s'$$

Конечно, способ предполагает, что каждое из этих произведений вычисляется однажды, а потом в готовом виде подставляется в выражение (II) в разных местах. Сложений (вычитаний) стало больше, но они имеют линейную стоимость по k. Все это приводит к тому, что способ Карацубы удовлетворяет оценке $O(k^{\log 3})$, (где $\log 3 < 1.59$). Это доказано в ([5], Theorem 8.3). Видно, что порядок роста стоимости существенно меньше, чем в простейшем способе умножения.

Случай произвольных степеней легко сводится к рассмотренному выше случаю таким образом:

$$fg = (x^m + f)(x^m + g) - x^{2m} - x^m(f + g).$$

Здесь $n = \max(\deg f, \deg g)$, $m = 2^{1+[\log n]}$, а квадратные скобки обозначают целую часть вещественного числа. Нетрудно доказать, что оценка стоимости $O(n^{\log 3})$ сохраняется (доказательство пропускаем, так как нам нужно доказательство для другой разновидности алгоритма).

3.2. Алгоритм для разреженного представления многочленов

О случаях разреженного и плотного представления многочлена. Обоснование.

В нашей библиотеке удобнее применять разреженное представление многочленов списками. Например: а) многочлен $x^{1000} + 1$ занимает в разреженном представлении много раз меньше памяти, чем в плотном, б) умножение $x^{100} * x^{90}$ многочленов занимает одно действие. Также имеется то удобство, что выбранное нами представление многочлена одной переменной оказывается частным случаем представления многочлена многих переменных. Ведь для многочленов многих переменных плотное представление не применимо в практике вычислений.

Арифметику многочленов в плотной записи мы не программировали, не относим это к первоочередным задачам.

Для плотного представления количество мономов в многочлене f равно $1 + (\deg f)$. Для разреженного представления выполнено неравенство

$$\text{число мономов в } f \le 1 + (\deg f).$$

Это учитывается в доказательстве оценок стоимости вычисления.

Какой смысл имеет асимптотическая оценка $O(n^\alpha)$ стоимости вычисления для какого-либо метода M, действующего на разреженно представленных многочленах, если n это степень многочлена?

Смысл такой же, как в Определении 1 из начала раздела. В обоих случаях стоимость вычисления измеряется количеством действий над мономами, а "размер" многочлена измеряется его степенью n – несмотря на количество мономов.

В частности, многочлен большой степени может содержать мало мономов, например, один.

Ниже описывается способ Kara умножения многочленов в разреженной записи, который мы считаем разновидностью способа Карацубы. Способ для плотной записи будем называть Kara-dense. Главное отличие способа Kara состоит в следующем. Для четного n, многочлен делится на старшую часть $x^{n/2}f_1$ и младшую часть f_2, $\deg f_1 = n/2$, $\deg f_2 < n/2$ – почти как в способе Kara-dense, только f_1 и f_2 могут сильно отличаться по степени и количеству мономов. В дальнейшем умножение применяется рекурсивно к парам многочленов, которые могут иметь разные степени и разное количество мономов. Для выравнивания степеней применяется прибавление монома x^m нужной степени, а потом делается поправка итога умножения. Для сведения метода к четной степени применяется (в дополнение к возможному предыдущему приему) расщепление многочлена на старший моном и хвост и последующая поправка итога.

Почему для исследования выбран метод умножения многочленов именно для разреженного представления многочлена?

С таким же успехом можно исследовать такую программу для плотного представления. Доказательство оценки стоимости для него уже известно, останется построить мп-доказательства. Но главное – это качество самой программы, с учетом системы, в которой она участвует, из этих соображений подбирается представление данных и изобретается алгоритм. Далее, для подтверждения надежности выводится теоретическая оценка стоимости вычисления и программируются мп-доказательства. Соображение "для этого представления данных легко получить такую-то оценку сложности, поэтому запрограммируем такой-то алгоритм" сомнительно с точки зрения разработки хорошей программы. В нашем же случае рассматриваются алгоритмы арифметики многочленов для разреженного представления.

Метод Kara умножения разреженно представленных многочленов.

Функция kara принимает сомножители f и g расставленные так, что $m = \deg f \le n = \deg g$. Она выдает произведение fg путем следующего вычисления. Разбираются случаи

- `(EE) m = n четное,`
- `(GE) m < n, n четное,`
- `(GO) m < n, n нечетное,`
- `(EO) m = n нечетное.`

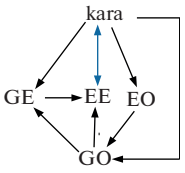


Рис. 1

Этим случаям соответствуют функции EE, GE, GO, EO, (в программе caseEqualEvenDeg, caseGreaterEvenDeg, caseGreaterOddDeg, caseEqualOddDeg), которые вместе с функцией kara вызывают друг друга согласно диаграмме (рис. 1).

Всюду ниже применяем обозначение EE(n,f,g) = = EE f g, где f и g многочлены, n = max (deg f) (deg g). Также пишем EE(n), когда считаем, что остальные два аргумента суть любые f и g такие, что n = = max (deg f) (deg g).

В случае EE

сначала применяется функция splitPolAtDegree. Она расщепляет f на старшую часть higher, чьи мономы имеют степени не меньше $k = n/2$, и младшую часть f_2, чьи мономы имеют степени меньше k. Многочлен higher представляется в виде $x^k * f_1$, соответственно, $f = x^k * f_1 + f_2$. Такое же расщепление применяется к $g : g = x^k * g_1 + g_2$. Функция splitPolAtDegree затрачивает не больше $2n$ действий. Затем применяется формула (II) раздела 3.1, где произведения f_1g_1, f_2g_2, ss' вычисляются рекурсивным применением функции kara с предварительной расстановкой сомножителей согласно их степеням.

Здесь выполнены равенства $\deg f_1 = \deg g_1 = \deg s = \deg s' = k$. Но $\deg f_2$ и $\deg g_2$ могут оказаться любыми натуральными числами меньшими k – из-за разреженного представления.

В случае GE положим $F = x^n + f$ и вычислим $f * g = (F - x^n) * g = \mathrm{EE}(F, g) - x^n g$, где вычисление $x^n g$ занимает не более n действий над мономами.

В случае GO

число $p = n - 1$ четное. Представим $g = \mathrm{mon} + g'$, где mon старший моном в g. Тогда $f * g = \mathrm{mon} * f + f * g'$, а $f * g'$ вычисляется через разбор следующих случаев для $\deg g'$, $\deg f$:

`case` ($\deg g' = ?p$, $\deg f = ?p$) `of`

`(True, True)` -> $\mathrm{EE}(f, g')$

`(True, False)` -> $\mathrm{GE}(f, g')$

`(False, True)` -> $\mathrm{GE}(g', f)$

`(False, False)` -> $\mathrm{GE}(f, x^p + g') - x^p f$

Во втором случае оказывается $\deg g' = p$, $\deg f < \deg g$, $\deg f \le p$, $\deg f \ne p$.

Поэтому $\deg f < \deg g'$, и применяется $\mathrm{GE}(f, g')$.

Подобным образом разбираются остальные случаи. В последнем случае выводятся неравенства $\deg f, \deg g' < p$, и потому применим вызов $\mathrm{GE}(f, x^p + g')$.

В случае EO представим $f = \mathrm{mon} + f$ где mon старший моном в f, и вычислим

$$f * g = \mathrm{mon} * g + \mathrm{GO}(f, g)$$

О пороге применения: для каждой разновидности алгоритма и каждой системы программирования есть порог выгодности применения способа Карацубы для многочленов. Назовем это число kThreshold. Если какой-то из сомножителей имеет число мономов меньше порога, то в среднем выгоднее применять простейшее умножение. Например, для систем Glasgow Haskell 8.8.3, Agda 2.6.1 и нашей программы порог равен 18. (Этот порог не влияет на порядок роста сложности вычисления)

3.3. Об оценках стоимости вычисления

Оценка стоимости $O(n^{\log 3})$ доказана в [5] (Theorem 8.3) для способа Карацубы для плотно представленных многочленов. Наша программа включает мп-доказательства правильности для несколько другого алгоритма (`kara`) – для разреженного представления.

В Приложении 11 дано доказательство оценки $O(n^{\log 3})$ для алгоритма `kara` (для разреженной записи) – опять, в смысле Определения 1. Оно несколько более сложное. Это влечет за собой, например, такие последствия. Умножение $x^n * x^n$ многочленов займет одно действие какой бы большой ни была степень n. А способ Kara-dense будет в этом примере обрабатывать списки длины $n+1$, и так далее, и затратит примерно $5 \cdot n^{\log 3}$ действий – очень много. Далее, добавим к x^n еще много любых мономов степеней меньше n и обозначим полученный многочлен f. Тогда при вычислении $f * f$ способ Kara во многих случаях затратит много действий – но не больше, чем $c \cdot n^{\log 3}$, где постоянная c приблизительно не больше 50.

Назовем *длиной* многочлена f (в разреженной записи) количество $\mathrm{S}(f)$ мономов в нем.

Об оценке в среднем для способа Kara в при неизменной степени разреженности:

автор проделал следующий опыт. При неизменной степени n перемножаемых многочленов (в примерах было положено $n = 6000$) программе `kara` подавались многочлены различных длин s. При изменении s от $n+1$ к меньшим значениям среднее время вычисления стремительно уменьшалось. В Приложении (раздел 11.3) дан набросок доказательства, который объясняет это явление. Именно, пусть $P(n, s)$ множество многочленов степени n, имеющих длину s, $C(n, s)$ – средняя стоимость вычисления по алгоритму Kara, когда сомножители берутся из множества $P(n, s)$. Если n является степенью двойки, а степени мономов в сомножителях распределены равномерно на отрезке $[0, n]$ (в среднем они и будут распределены равномерно), то выполнена оценка

$$C(n, s) \in O(s^2).$$

Вторая степень в этой оценке нам не мешает, так как важны значения s достаточно малые в сравнении с n. Например:

- при $s < \sqrt{n}$ порядок стоимости окажется не больше n, тогда как
- при наибольшей длине $n+1$ стоимость в наихудшем случае достигнет порядка не меньше $n^{3/2}$

(второе утверждение не доказано строго, оно взято из итогов испытаний программы). Уточнение этой оценки (по худшему случаю или в среднем) для общего случая а также построение строго доказательства может быть предметом отдельного исследования.

Перечислим итоги, которые показывают, что способ вычисления Kara (для разреженного представления) имеет смысл.

- Способ Kara имеет верхнюю оценку $O(n^{\log 3})$ стоимости вычисления – такого же порядка, как способ Kara-dense (когда последний применяется к плотным записям тех же многочленов). И это много меньше (точной) оценки $O(n^2)$ стоимости для простейшего способа умножения.
- Эта оценка равномерна по длине многочленов степени не больше n: верна как для многочленов малой длины, так и для многочленов большой длины (максимум длины равен $n+1$).
- Средняя (на опыте) стоимость вычисления способом Kara при неизменном n быстро убывает с уменьшением длины сомножителей. Это дает преимущество в среднем быстродействии на сильно разреженных многочленах перед способом Kara-dense.
- Испытания программы (раздел 6) показывают соответствие верхней оценке по степени, и вообще, быструю работу программы.
- Программа способа Kara снабжена мп-доказательством равносильности простейшему умножению.

4. ПРЕДВАРИТЕЛЬНЫЕ ОБЪЯСНЕНИЯ О ПРОГРАММИРОВАНИИ С ЗАВИСИМЫМИ ТИПАМИ

Обсудим черты доказательной программы на языке `Agda` для метода `kara` (раздел 3.2).

Язык `Agda` [9, 10] основан на чистой функциональности, “ленивом” способе вычисления по умолчанию, поддержке обобщенного программирования (generic programming), аппарате зависимых типов. Приблизительно можно считать, что это язык `Haskell`, расширенный аппаратом зависимых типов. Зависимые типы позволяют адекватно описать алгебраическую область, зависящую от обычного значения ([11], Введение). Кроме того, этот аппарат позволяет вставлять в программу доказательства утверждений, и эти доказательства будут проверяться компилятором. Становится возможным описывать метод вычисления в программе так, как это делается в учебниках и научных статьях, вместе с доказательством правильности. Доказательства понимаются по умолчанию в смысле конструктивной математики [12, 13].

От известной системы `Coq` [14] (и ее языка `Gallina`) главные отличия состоят в “ленивом” вычислении по умолчанию, отсутствием разделения на язык задания вычислений и язык построения доказательств, одинаковой поддержкой как программирования быстрых вычислений, так и доказательств.

От известной системы `Lean` [15] главные отличия состоят в “ленивом” вычислении по умолчанию, чисто-функциональном программировании, одинаковой поддержкой как программирования быстрых вычислений, так и доказательств, требованием конструктивного доказательства по умолчанию.

Техника проверки доказательств компилятором (точнее – проверяльщиком типов – type checker) основана на том, что всякое утверждение S представляется подходящим типом T (зависящим от значений). Доказательство для S есть любая функция (завершающийся алгоритм), которая выдает любое значение v типа T. Проверяльщик типов проверяет отношение v : T посредством символьных вычислений с выражениями типов. Таким образом доказательство утверждения проверяется до начала компиляции в исполняемый код.

Недоказуемое высказывание соответствует пустому типу $\perp$ с пустым множеством значений. Отрицание высказывания соответствует функции, отображающей соответствующий высказыванию тип в пустой тип. Импликация выражена типом S $\to$ T всех функций из S в T, где S и T представляют соответствующие утверждения. Конъюнкция выражена произведением S $\times$ T типов, дизъюнкция выражена суммой S $\uplus$ T типов. Доказательство индукцией по построению данного выражается в виде рекурсивно заданной функции. Применение леммы выражается в виде вызова функции, представляющей доказательство леммы.

Пока ограничиваемся только конструктивными доказательствами [12, 13] (без использования принципа Маркова для доказательства завершаемости алгоритмов). В частности, всякий объект существует лишь как итог некоторого данного алгоритма, и должно быть дано доказательство завершаемости этого алгоритма.

Мы часто пользуемся законом исключенного третьего – в конструктивной математике это возможно в тех (часто встречающихся) случаях, когда приведен алгоритм разрешения соответствующего отношения.

Наш подход в отношении конструктивизма таков. a) Конструктивное доказательство лучше, чем неконструктивное, так как дает больше знания. Например, завершающийся алгоритм построения объекта – это больше, чем неконструктивное доказательство его существования. b) Если нужно доказательство, а конструктивное доказательство не удается найти, то можно обойтись неконструктивным, использовав аксиому исключенного третьего – она просто записывается на языке `Agda`. Конечно, лучше явно выделить места, где она применяется.

Добавим, что есть широкая область в математике и научных вычислениях, где достаточно лишь конструктивного подхода, и пока что наша библиотека находится целиком в этой области. Но есть и еще большая область, где конструктивных доказательств не хватает.

Более подробные объяснения по данной системе программирования и примеры содержатся в [9, 11] и в руководстве по библиотеке [1].

4.1. О синтаксисе языка Agda

Имена операторов и отношений отделяются пробелами. Например, в строке

```
a+b≈0 : a + b ≈ 0#
```

программы a+b ≈ 0 есть имя значения, двоеточие отделяет имя значения от выражения типа, символы ‘+’ и ‘≈’ в выражении типа суть соответственно имя оператора сложения и имя двуместного отношения, 0# есть имя нулевой постоянной. Вся эта строка означает объявление: значение `a+b≈0` имеет тип `a + b ≈ 0#` (и этот тип зависит от значений `a, b, 0#`).

Знак подчерка в имени отношения или операции означает, что в синтаксисе программы во входном выражении на этой позиции ставится выражение аргумента этой операции.

Знак равенства ‘=’ – это определение идентификатора (присваивание) – часть синтаксиса программы.

Равенство _≈_ – это знак отношения равенства на некоторой области, это отношение задается для каждой области программы пользователя или стандартной библиотекой.

4.2. Внешние и внутренние доказательства

В каком смысле функция на языке Agda может быть доказательством?

Если функция строит доказательство, то само тело этой функции является доказательством некоего общего утверждения. И это доказательство проверяется проверяльщиком типов в общем, символьном виде, так же, как читатель проверяет доказательство, изложенное в книге. Если и когда эта функция вычисляется во время исполнения, то она выдает доказательство для какого-то частного случая, и строение этого доказательства в обычных случаях бывает не нужно разбирать. Например, можно запрограммировать функцию m≤m+n (это ее имя), выражающую доказательство утверждения ∀ m n (m ≤ m + n) для натуральных чисел. Ее тело выражает общее доказательство, скажем, по индукции. А вычисление вызова (m≤m+n 2 9) во время исполнения выдаст только доказательство неравенства 2 ≤ 2 + 9.

Обычный подход к доказательному программированию таков. Алгоритм *A* программируется как некоторая функция F на Агде в духе языка Haskell, без построения доказательств. Затем пишутся функции мп-доказательств для выбранных свойств F: если аргументы F удовлетворяют таким-то условиям, то итог функции обладает таким-то свойством. При этом обычно доказательство по индукции следует рекурсивному построению функции F. Например, таким способом в библиотеке запрограммированы доказательства переместительного и сочетательного законов для сложения и простейшего умножения многочленов (если не учитывать свидетельства, содержащиеся в данных типа Pol). Назовем такие доказательства *внешними*. Они проверяются компилятором (в первой его части – проверке типов). При этом вычисление функции F при исполнении программы заведомо не вызывает функции доказательств.

Но на практике для многих функций оказывается чрезвычайно затруднительным построение внеших мп-доказательств. Возможно это следствие каких-то технических трудностей в воплощении языка Agda. Поэтому мы снабжаем многие функции *внутренними* доказательствами, то есть такими, которые являются частью итога функции. Их легче задать, чем внешние доказательства, они строятся в теле функции заодно с основной частью итога. Этот подход выгоден еще и тем, что уменьшает повторы кода, так как внутреннее доказательство использует уже развернутую внутреннюю среду. С другой стороны, в среднем часть доказательства занимает довольно много кода, и этот код мешает читателю разглядеть строение самого способа вычисления. В таком случае может быть полезно предварять программу комментарием с короткой записью основной функции, не снабженной доказательством.

Но раз построение доказательства включено в тело функции, то не получится ли так, что при исполнении программы на доказательство потратится время сравнимое с тем, что тратится на главный итог?

Не получится – при естественном способе программирования. Ибо Agda это язык “ленивого” исполнения по умолчанию. Поэтому, если функция F выдает пару

(<главный итог> , <доказательство>),

а функция – клиент G не использует существенно вторую часть пары, то при вычислении G эта часть вычисляться не будет. Существенным использованием данного мы называем разбор строения этого данного. Доказательства в языке Agda являются данными. Их можно разбирать на части, делать с ними содержательные вычисления, распечатывать.

Но в обычных научных вычислениях не требуется разбирать строение доказательств во время вычисления, функции – клиенту всегда достаточно лишь само *наличие* какого-то доказательства соответствующего типа. А это наличие выясняется проверяльщиком типов в символьном виде до начала исполнения программы. То есть в обычных научных вычислениях доказательства никогда не используются по существу при исполнении и потому не занимают ресурсов при исполнении. Например, функция lm выдает старший моном многочлена f, но нужен дополнительный аргумент hm-f : HasMon f – доказательство того, что многочлен не нулевой. Видя вызов (lm f hm-f) взятия старшего монома, проверяльщик проверит, имеет ли значение hm-f подходящий тип. А во время исполнения, при вычислении этого вызова данное hm-f разбираться/вычисляться не будет.

Так обстоит дело и с функцией karatsuba.

Мертвый код: проверяльщик типов выдает код для компилятора, в этом коде часто бывают участки доказательств, часто большие. Внешние доказательства легко узнать, и после проверки они удаляются компилятором. Но некоторые внутренние и ненужные (после проверки) доказательства проверяльщик не может распознать как ненужные, если функция сложно устроена. И эти “мертвые” доказательства задают работу компилятору и проникают в объектный код – хотя и не работают во время исполнения. В главе “run-time irrelevance annotation” в документации на систему

Agda описаны обозначения Erased, @0, простановка которых в программу в правильных местах принуждает Агду удалить мертвый код до компиляции. Но в данном выпуске библиотеки эти дополнительные затраты компиляции малы в сравнении с остальными издержками, поэтому даже небольшое усложнение программы erased – обозначениями не имеет смысла.

4.3. Завершаемость

По умолчанию Agda должна убедиться в завершаемости каждой заданной функции (алгоритма). Указание TERMINATING избавляет проверяльщик типов от проверки завершаемости, но применение такого указания это отступление от доказательного программирования. Поэтому мы не применяем такое указание, так же, как не применяем указаний постулирования postulate, anything. Часто Agda сама убеждается в завершаемости путем обнаружения синтаксического уменьшения рекурсивных вызовов. Но также часто выдается сообщение “Termination checking failed”. В этом случае надо помочь Агде распознать завершаемость. Обычно проще всего ввести в функцию дополнительный аргумент в виде счетчика – натурального числа. Воспринимая неравенства, доказанные в теле функции для этого счетчика, Agda убеждается в завершаемости.

Доказательство завершаемости всегда неявное. Сравним на примере: на языке Agda можно выразить высказывание “функция f коммутативна”, но нет способа прямо выразить высказывание “функция f завершается” так, чтобы потом использовать в программе какое-то доказательство p для этого высказывания.

5. ПРОГРАММА СПОСОБА КАРАЦУБЫ НА ЯЗЫКЕ Agda

Функция способа Карацубы задана в модуле, объявленном как module Pol.Karatsuba (R : DecCommutativeRing).

Здесь коммутативное кольцо R с разрешимым равенством является параметром – это область коэффициентов. Подставляя различные воплощения для R, программист автоматически настраивает все функции из этого модуля на выбранную область коэффициентов.

Но в отличие от типа (Pol a) в Haskell-программе тип Pol в нашей Agda-программе в библиотеке DoCon-A скрывает в себе машинно-проверяемое условие определения представления многочлена: свидетельство неравенства нулю коэффициентов и свидетельство упорядоченности показателей (раздел 2). Функции арифметики многочленов строят эти свидетельства для итога действия, исходя из таких свидетельств для операндов. Есть и другие отличия.

Главным доказательством для функции karatsuba является доказательство ее равносильности простейшему умножению:

```
(f g : Pol) → (karatsuba f g) =p (f *p g)   (IV)
```

Здесь _=p_ равенство многочленов, _*p_ простейшее умножение. Последние две сущности взяты из модулей, посвященных многочленам. Но мы сначала задаем функцию Карацубы в формате, включающем основное доказательство.

```
karatsuba-withProof : (f g : Pol) →
                      ∃ (\h → h =p f *p g)
```

Выдается пара (h , eq), где h – произведение, eq – доказательство равенства h =p (f *p g). Это доказательство внутреннее, оно задается в теле функции в циклах, выражающих вычисления произведения.

Для вычисления произведения надо вызывать не эту функцию, а функцию

```
karatsuba : Op₂ Pol
karatsuba f = proj₁ ∘ karatsuba-withProof f
```

(_∘_ – функция композиции). Она выдает первую часть итога вызова karatsuba-withProof. Доказательства различных свойств умножения способом karatsuba теперь просто получить переносом соответствующих доказательств для функции _*p_. Например, переместительный закон:

```
karatsuba-comm : Commutative _=p_ karatsuba
karatsuba-comm f g =
    let open EqReasoning
        (h , h=fg ) = karatsuba-withProof f g
        (h′ , h′=gf) = karatsuba-withProof g f
    in
    begin ⟨ polSetoid ⟩ h ≈⟨ h=fg ⟩
                        f *p g ≈⟨ *p-comm f g ⟩
                        g *p f ≈⟨ =p-sym h′=gf ⟩
                        h′
    end
```

Дадим пояснения. Здесь h = karatsuba f g, h’ = karatsuba g f, и требуется доказать h =p h’.

h = fg это доказательство равенства h =p (f *p g), взятое из итога вызова karatsuba-withProof f g.

h'=gf это доказательство равенства h' =p (g *p f), взятое из итога вызова karatsuba-withProof g f. В силу доказательств h=fg и закона *p-comm перестановочности простейшего умножения, доказанного в библиотеке, выводится равенство h =p (g *p f). Правая его часть равна h' в силу равенства h'= gf, которое применяется справа налево и завершает доказательство.

В вызове (begin ... end) в левой колонке записана последовательность значений, в которой каждое значение равно предыдущему. В каждой строке правой колонки записан вызов функции, которая выдает доказательство равенства значения в левой колонке следующему значению. Замечательно, что (begin ... end) это не конструкция языка, а вызовы функций `begin_` и `_end` из стандартной библиотеки, написанной на Агде (а применение функции может записываться префиксно, инфиксно или постфикстно). Такова же функция ≈⟨_⟩_, применяемая в правой колонке. Эти вызовы позволяют наглядно записать композицию применений закона транзитивности равенства.

5.1. Функция splitPolAtDegree

Функция

```
splitPolAtDegree :
          {n : ℕ} {f : Pol} (hmF : HasMon f)
          (n≤degF : n ≤ deg f hmF) →
          SplitPolAtDegree n f
```

это главный шаг метода. Она воплощает расщепление из шага EE алгоритма `Kara` из раздела 3.2. Она принимает натуральное n (в программе – целая часть (deg f)/2), ненулевой многочлен f степени не меньше n и выдает структуру, описывающую разбиение f на старшую часть higher и младшую часть lower – как описано в разделе 3.2. Только некоторые аргументы и некоторые части итога суть доказательства свойств этого разбиения. В отдельном модуле дополнительно доказаны свойства этого разбиения. Например:

• hmH : HasMon higher – свидетельство того, что старшая часть ненулевая,

• splitEq : mons ≡ monsH ++ monsL – список мономов f синтаксически равен (propositional equality) соединению списка старших мономов и списка младших мономов,

• f≡p-higher+lower : f ≡p higher +p lower – f синтаксически равен сумме многочленов higher+lower (в смысле синтаксического равенства списков мономов),

• $x^n|h : x^n|$ h – свидетельство "x^n делит higher", и выдается частное h_0,

• f=$x^n h_0$+low : f = p (x^n *p h_0 +p lower)

Эти доказательства используются в мп-доказательстве теоремы (IV).

5.2. Функция karatsuba-withProof

Функция karatsuba-withProof разбирает случаи нулевых аргументов, упорядочивает два аргумента по степени и вызывает функцию

```
kara :
  (f g : Pol) (hmF : HasMon f) (hmG : HasMon g) →
  deg f hmF ≤ deg g hmG → (cnt : ℕ) →
  deg g hmG ≤ cnt → ∃ (\h → h =p f *p g)

kara f g _ _ _ 0 _ = f *p g , =p-refl {f *p g}
kara f g hmF hmG e≤e′ (suc cnt) e′≤1+cnt =
             considerCases (length mons <? kThreshold)
                           (length mons′ <? kThreshold)
                           (e =? e′) (2 |? e′)
```

Функция `kara` это главный цикл метода. Она принимает два многочлена, hmF и hmG суть свидетельства того, что они ненулевые, следующий аргумент это свидетельство неравенства deg$f$ ≤ ≤ deg$g$. Последние два аргумента суть счетчик cnt и свидетельство неравенства deg$g$ ≤ cnt. Они нужны для доказательства завершаемости функции.

При рекурсивном вызове функции kara счетчик уменьшается синтаксически от выражения (suc cnt) к выражению cnt, а степень второго многочлена уменьшается по меньшей мере на

единицу. В начальном вызове cnt = deg g, а когда значение deg g достигает нуля, функция выдает итог. Поэтому Agda решает, что алгоритм завершается.

Первое предложение разбирает случай нулевого счетчика. В нем выдается произведение, вычисленное простейшим способом (а нам известно, что в этом случае оба многочлена суть постоянные). Поэтому вторая часть итога это доказательство тождественного равенства (вида X =p X).

Второе предложение это выход на разбор случаев и шаг рекурсии. Вызывается функция

```
considerCases : Dec (length mons < kThreshold) →
                Dec (length mons′ < kThreshold) →
                Dec (e ≡ e′) → Dec (2 | e′) →
                ∃ (\h → h =p f *p g),
```

которая сначала проверяет, достаточно ли мономов в f и g (проверка занимает не более kThreshold шагов). Если не достаточно много, то выполняется простейшее умножение f * p g. Если достаточно, то проверяются условия e ≡ e', 2 | e (второе условие это делимость на 2). По их итогам вызывается одна из функций, воплощающих случаи (EE), (EO), (GE), (GO) из раздела 3.2. Например, случай (EE) вызывается в третьем предложении:

```
considerCases
 (no |mons|≮kTh) (no _) (yes e≡e′) (yes 2|e′) =
 let 2|e = subst (2 |_) (sym e≡e′) 2|e′
     2≤e = |f|≮kThreshold⇒2≤deg-f {f} hmF
                                    |mons|≮kTh
     e≤1+cnt = subst (_≤ 1+cnt) (sym e≡e′)
                                e′≤1+cnt
 in
 caseEqualEvenDeg f g hmF hmG e≡e′ 2≤e 2|e
                                   e≤1+cnt
```

– в обоих многочленах много мономов, e ≡ e', e' – четное. Здесь сначала доказывается, что e тоже делится на 2. Потом неравенство 2 ≤ e выводится из того, что число мономов не меньше порога (|mons| ≮ kTh). Потом неравенство e ≤ 1+cnt выводится из неравенства e' ≤ 1 + cnt, данного выше в вызове kara. Наконец, функции caseEqualEvenDeg кроме многочленов f и g даются еще эти построенные доказательства 2 ≤ e, 2|e, e ≤ 1 + cnt.

Условие 2 ≤ e здесь нужно для возможности дальнейшего разбиения многочлена f. Если оно не выполнено, то из четности e следует e = 0 = e/2, и разбиение не уменьшит степени многочлена, и проверяльщик типов сообщит о неудаче доказательства завершаемости.

Функции для случаев (EO), (GE), (GO) разбирать не будем, а дадим лишь пояснения к функции caseEqualEvenDeg:

```
caseEqualEvenDeg :
    (u v : Pol) (hmU : HasMon u)
    (hmV : HasMon v) →
    let dU = deg u hmU; dV = deg v hmV
    in
    dU ≡ dV → 2 ≤ dU →2 | dU → dU ≤ 1+cnt →
    ∃ (\h → h =p u *p v)

caseEqualEvenDeg u v hmU hmV E≡E′ 2≤E 2|E
                                  E≤1+cnt =
  let
    E = deg u hmU; E′ = deg v hmV
    (k , E≡k*2) = 2|E

    monE = mkMon 1# E – моном x^E
```

```
    monK = mkMon 1# k
    …
    …
in
eeDeg monE monK ss′ f₁g₁ f₂g₂ , eq
```

Выражение (k, E≡k*2) = 2|E это определение отношения делимости, примененное к паре 2, E. Оно включает в себя частное k и доказательство равенства E ≡ k * 2.

Обозначения f_1, g_1, f_2, g_2, s, s' – те же, что в формуле (II) из раздела 3 – с заменой f -> u, g -> v : $u = x^E * f_1 + f_2$, и так далее. В первой части пары итога выдается значение

$$H = (x^E * f_1 g_1) + \\ + ((x^K * (ss' - (f_1 g_1 + f_2 g_2))) + f_2 g_2) \quad \text{(IIA)}$$

Это формула (II) из раздела 3. Здесь + это сложение многочленов, '-' вычитание многочленов, '$x^E *$' и '$x^K *$' суть умножения на моном, а произведения f_1g_1, f_2g_2, ss' вычисляются рекурсивным применением функции kara.

Во второй части пары выдается доказательство eq равенства H = u *p v, где `_*p_` простейшее умножение многочленов. Для получения доказательства eq используется то, что kara тоже возвращает произведение в паре с доказательством, например:

```
(ss′ , ss′Eq) = kara s s′ hmS hmS′
                (≤-reflexive degS≡degS′)
                cnt degS′≤cnt,
```

где ss'Eq это доказательство равенства ss' = p (s *p s').

Равенство eq для H выводится из ss'Eq и из подобных же равенств для произведений путем замены в (IIA) f_1g_1 на f_1 *p g_1, f_2g_2 на f_2 *p g_2, ss' на s *p s' в силу соответствующих равенств и последующего применения законов коммутативного кольца (раскрытие скобок, приведение подобных членов и тому подобное). Все это формально расписывается в функции caseEqualEvenDeg.

А для этих формальных построений необходимы доказательства, которые надо подставлять в аргументы рекурсивных вызовов функции kara. Например, в вызов вычисления ss' входит доказательство degS' ≤ cnt неравенства deg s' ≤ cnt. А в вызове caseEqualEvenDeg есть доказательство E ≤ 1+cnt : deg u ≤ suc cnt (suc это конструктор "следующий" для натуральных чисел). Оно нужно для доказательства завершаемости. Это доказательство получается следующим образом. 2 ≤ E = deg u ≤ suc cnt = 1 + cnt, поэтому E/2 ≤ cnt.

Имеем s' = g_1 +p g_2, deg g_1 = E/2, deg g_2 < E/2.

В модулях для аддитивной части арифметики многочленов доказана лемма о том, что степень суммы не больше максимума степеней слагаемых. Поэтому deg s' ≤ E/2 ≤ cnt. И даже это рассуждение содержит пробелы, так что в программе добавлены необходимые подробности.

В доказательстве встречаются такие определения значений, как f_1*g_1 = f_1 *p g_1. Эти символьные выражения участвуют в доказательстве, но в силу "ленивого" вычисления и устройства функции они не вычисляются во время исполнения.

6. ИСПЫТАНИЕ НА ПРИМЕРАХ

Модуль `Pol/KTest.agda` библиотеки содержит программу испытания на производительность функции karatsuba.

При целых коэффициентах и больших степенях многочленов влияние распухания коэффициентов в ходе вычисления может оказаться существенным. Поэтому во всех испытаниях областью коэффициентов полагается кольцо R = $\mathbb{Z}$/(b) остатков по модулю b. Значение `b = 99991` выбрано из того соображения, что, нулевых мономов получается мало, и это дает более затратный случай для нашего алгоритма, опирающегося на разреженное представление многочлена.

Первое испытание – по худшему случаю, когда степень разреженности нулевая. Другие два

Таблица 1.

kThreshold = 18				
время[sec]				
n	p-p, deg p, length p	pk-pk, deg pk, length pk	deg p	l(mons)
9	0.3	0.2	512	513
10	1.3	0.8	1024	1025
11	5.1	2.5	2048	2049
12	21.1	7.7	4096	4097
13	84.2	25.0	8192	8193
14	349.6	78.0	16384	16385

испытания – на случайных многочленах средней степени разреженности.

Первое испытание таково.

Задается значение n : ℕ. Затем функция pk возводит многочлен f = x + 1 в степень 2^n двоичным способом – путем применения умножения способом Карацубы n раз. Например, для n = 12 умножение применяется 12 раз, и получается многочлен степени 4096.

Затем печатаются (1) степень pk, (2) многочлен pk – pk, (3) количество мономов в pk, (4) сумма коэффициентов pk в кольце R и измеряется время исполнения.

Эти данные выбраны так, чтобы распечатка не была большой и при этом все части многочлена pk были вычислены. Такие ухищрения нужны оттого, что вычисления выполняются “ленивым” способом.

Части (1), (2), (3) служат еще для проверки правильности (сама-то система Agda тоже может содержать ошибки).

Тот же многочлен вычисляется в этом модуле через простейшее умножение _*p_, он обозначен p. Замена pk -> p в функции main даст испытание для простейшего умножения.

Если в строке ‘check =’ заменить pk -p pk на p -p pk, то будет проверено равенство двух способов умножения на этом примере (хоть оно и доказано в программе для общего случая, но ошибки могут быть в самих системе Agda, компиляторах и, наконец, в компьютере).

Арифметика коэффициентов воплощена в самом общем виде. Библиотека содержит определение евклидова кольца (EuclideanDomain) и модуль Residue.Euclidean, в котором задана арифметика кольца E/(b) остатков для произвольных евклидова кольца E и элемента b в нем. В модуле Int.II задано воплощение euclideanDomain структуры евклидова кольца для целых чисел. Объявление

```
open import Int.II using ()
      renaming (euclideanDomain to E)
```

вносит евклидову структуру для ℤ, обозначая ее E. Далее, объявления

```
open Residue.Euclidean E b ℤ-Read ℤ-Show
     using (Show0-r; Read-r; semiring-r;
                     decCommutativeRing-r)
     renaming (EucResidue to R)
decCRing-r = decCommutativeRing-r b≠1
```

вносят структуры DecCommutativeRing и некоторые другие для области E/(b) остатков и дают имя R для носителя этой области. Например, действие умножения в области E/(b) можно теперь получить, объявив

```
open DecCommutativeRing decCRing-r
           using ()
           renaming (_*_ to _*r_),
```

при этом оно получит имя *r.

Наконец, объявления

```
open import Pol.Karatsuba decCRing-r
open import Pol.Over-decComRing
decCRing-r Show-r Read-r "x"
```

вносят в поле действия функцию karatsuba и арифметику многочленов – обе для коэффициентов из области R остатков.

Приведем таблицу испытания для системы Agda-2.6.1, MAlonzo, ghc-8.8.3, Ubuntu-Linux 18.04 и персонального компьютера частоты 3 гигагерц (табл. 1).

Во второй колонке (“p-p”) – время для простейшего умножения, в третьей колонке (“pk-pk”) – для функции karatsuba.

Обозначим

- $f = x + 1$,
- $T(p, n)$ время вычисления p(n) из второй колонки этой таблицы,
- $T(p_k, n)$ время вычисления pk(n) из третьей колонки,
- $T(n), T_k(n)$ время вычисления произведения $f^m * f^m$ для $m = 2^n$ (после того, как значение f^m готово) простейшим способом и функцией karatsuba соответственно.

Тогда $T(n) = T(p, n+1) - T(p, n)$,

$T_k(n) = T(p_k, n+1) - T(p_k, n)$.

Получаем таблицу стоимости возведения в квадрат многочлена степени 2^n (табл. 2):

Отношения $T(n+1)/T(n)$, $T_k(n+1)/T_k(n)$ показывают, как увеличивается время возведения в квадрат при удвоении степени многочлена – соответственно при простейшем умножении и при применении функции karatsuba. В первом случае это отношение равно приблизительно 4, для функции karatsuba оно равно приблизительно 3.1.

Таблица 2.

n	T(n)	Tk(n)	Tk(n+1)/Tk(k)
9	1.0	0.6	
10	3.8	1.7	2.8
11	16.0	5.2	3.1
12	63.1	17.3	3.3
13	265.4	53.0	3.1

Таблица 3.

n	время[sec]		
	p	pk	отношение T(pk) к предыдущему
1000	1.9	0.8	
2000	8.1	2.5	3.1
4000	33.0	7.5	3.0
8000	136.5	22.8	3.0
16000		68.7	3.0
32000		219.4	3.2

Таблица 4

m	время[sec]	
	p	pk
100	1.5	2.3
201	2.9	3.2
...		
1000	16.0	7.7
1009	32.7	11.5
4002	65.6	17.1
8003	133.2	25.5

Первое как раз соответствует стоимости $c \cdot k^2$ вычисления произведения многочленов степени k, второе соответствует стоимости вычисления $c \cdot k^{\log 3.1}$.

Также программа испытывалась на случайных многочленах одинаковой степени n над коэффициентами из той же области R остатков. В табл. 3 p = f*g обозначает простейшее умножение, pk = karatsuba f g:

Здесь колонка “pk” соответствует стоимости вычисления приблизительно $c \cdot n^{\log 3.1}$.

Таблица 4 показывает испытание для разных степеней m = deg f, n = deg g, в том числе для нечетных m, при n = 9000.

Видно, что время вычисления в правой колонке растет гораздо медленнее, чем в случае одинаковых степеней.

6.1. Сравнение с программой без доказательств на языке Haskell

Также были запрограммированы на языке Haskell арифметика многочленов и способ Карацубы для нее, со всеми теми же алгоритмами, разумеется – без доказательств в программах. Коэффициенты считаются принадлежащими абстрактной области класса Num, что соответствует сигнатуре коммутативного кольца (но соблюдение законов кольца в данном случае лежит на программисте). В примере испытания в качестве коэффициентов берется та же область остатков по модулю b = 99991, но она построена не подстановкой в общую конструкцию с евклидовым кольцом, а прямо запрограммирована для частного случая целых чисел по модулю b.

В системе Glasgow Haskell 8.8.3 примеры из вышеприведенной таблицы вычисляются примерно в 4 раза быстрее. Множитель 2 происходит из того, что область $\mathbb{Z}/(b)$ задана более прямо. Другой множитель 2 – из того, что, вообще, Agda является языком несколько более высокого уровня абстракции, к тому же логическим языком, и это влечет за собой некоторую плату в части производительности.

7. ОБ ИЗДЕРЖКАХ ДОКАЗАТЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Сравним программы для способа Карацубы в случае программирования на языке Haskell – без доказательств и на языке Agda – с доказательствами в программе.

Нижеприводимые показатели даются только для модуля Karatsuba при условии, что остальные модули библиотеки уже скомпилированы (замечание об особом способе компиляции приведено в середине Введения), и – для систем Glasgow Haskell 8.8.3 и Agda-2.6.1 - MAlonzo соответственно.

Производительность программы на Агде в 2 раза ниже, а порядок роста стоимости вычисления такой же.

Объем исходной программы на Агде больше в 6 раз.

Программа на Агде собирается в исполняемую в 100 раз дольше (на машине в 3 гигагерц – 300 sec.).

Объектный код программы на Агде в 130 раз больше.

Его можно сократить в два раза за счет вставки в исходную программу в критических местах обозначения `@0` по удалению ненужного кода. Но мы посчитали, что для данного примера выигрыш не стоит даже небольшого усложнения программы, ибо время сборки и производительность заметно не меняются.

Такова на сегодня цена полностью адекватного способа программирования символьных математических вычислений в системе `Agda`. И на наш взгляд эта цена приемлема, ибо доказательств много, они – полные, а библиотека `DoCon-A` в настоящее время содержит много разных определений понятий, лемм и алгоритмов.

Но: трудоемкость составления доказательств велика.

Она может быть сильно сокращена за счет добавления в библиотеку специальных доказывателей для различных разделов алгебры и логики, например, для доказательства равенств в коммутативных алгебрах, в конечных группах, и так далее, отдельно для каждой особенной предметной области.

На конференции в городе Bath в 2013 году некий слушатель жаловался на то, что он, как ни старается, не может писать доказательства на Агде.

В последствии автор составил (пользуясь подсказками разработчиков системы Агда) перечень главных практических советов, следуя которым он сумел построить доказательства в системе Agda для многих утверждений, вошедших в библиотеку `DoCon-A` ([1], выпуск 3.2rc, руководство `manual.pdf` из архива библиотеки, раздел 1.2), и которых, похоже, достаточно для продолжения проекта библиотеки.

8. О ДРУГИХ РАБОТАХ ПО ДАННОМУ ПРЕДМЕТУ

Метод Карацубы для многочленов применяется в различных известных библиотеках научных вычислений, например, в `MAPLE`. Но в этих библиотеках речь не идет о доказательной программе.

В диссертации [6] 2014 года среди некоторых других замечательных методов и программ описана разработка как раз доказательной программы для способа Карацубы в системе `Coq` при поддержке математической библиотеки `MathComp`. Главные отличия от нашей разработки таковы.

• Выбрано плотное представление многочлена в виде списка коэффициентов (среди которых возможны нули).

• При расщеплении многочлена степени n выделяется многочлен степени $[n/2]$ (целая часть). Но для этого способа не приведено доказательства оценки сложности вычисления. Вывести эту оценку предложено в упражнении 8.5 книги [5], где это упражнение названо непростым.

• Применен подход data refinement, который годится, вообще, для многих методов вычисления.

Подход data refinement (в случае метода Карацубы для многочленов) состоит в том, что сначала, так же, как у нас, многочлен определяется в виде зависимого типа. В нашем подходе данное “многочлен” включает два доказательства (начало раздела 2), в подходе [6] это только доказательство неравенства нулю старшего коэффициента – назовем этот тип CPol (обозначения CPol, CPol’, toSimple, cKara, cKara’ введены здесь нами).

Потом программируются простейший алгоритм умножения многочленов и способ Карацубы – с учетом зависимого типа для многочлена, то есть с построением доказательств для старшего коэффициента. Назовем последнюю программу `cKara`. Потом программируется доказательство равносильности этих двух функций. После этого места наши подходы расходятся. Наша программа уже готова. А в подходе [6] делаются дополнительные построения, описанные ниже.

Доказательная программа в системе `Coq`, действующая над данными зависимых типов, может иметь сильно меньшую производительность, чем программа без доказательств на простых типах. Поэтому вводится простой тип для многочлена – не включающий доказательств, назовем его CPol’. И программируются арифметические действия и способ Карацубы для простого типа. Эта программа (назовем ее `cKara'`) свободна от доказательств и приспособлена для быстрого исполнения.

Далее, программируется отображение toSimple: $\mathrm{CPol} \to \mathrm{CPol'}$, которое просто удаляет доказательство из данного типа CPol. Программируется доказательство того, что отображение toSimple является инъективным гомоморфизмом относительно сложения. Это доказательство весьма простое. Так же просто программируется доказательство того, что отображение cKara равносильно отображению cKara’ – в смысле коммутативности диаграммы (рис. 2).

Отсюда следует, что **а)** правильность программы cKara’ выведена из правильности программы cKara, **б)** для быстрого вычисления можно применять функцию cKara’.

Это разумный подход. И он легко может быть применен в нашей библиотеке. Но для нашей библиотеки он пока что представляется излишним, так как в языке `Agda` программы по умолчанию выполняются “лениво”, и как описано в разделе 4.2, даже наличие внутренних доказательств не сказывается существенно на производительности исполняемого кода. Это подтверждается таблицей затрат времени из раздела 6, сравнением по порядку роста стоимости вычисления и по

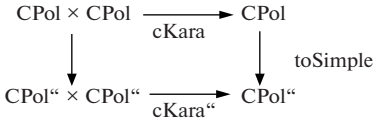


Рис. 2

производительности с программой, написанной на языке `Haskell` (раздел 6.1), и примерами некоторых других алгоритмов из библиотеки. К тому же отпадает необходимость повторного программирования метода.

Вообще же, что касается нашего подхода к программированию алгебры: он состоит в стремлении построить достаточно богатую библиотеку доказательных программ научных вычислений на основе минималистичного подхода, избегая введения различных излишних языков и парадигм. Не знаем, что потребуется дальше, но пока оказываются достаточными лишь язык `Agda` и библиотека, на нем же написанная. Главное же содержание проекта – выражение математических построений, методов, алгоритмов и доказательств в математически адекватном виде, в большой общности, на подходящем функциональном языке программирования, и с учетом производительности программ.

8.1. О важности мп-доказательств вообще

Система построения формальных машинно-проверяемых доказательств помогает найти ошибки в математических доказательствах. Например, в начале работы [7] названы несколько важных проблем, получивших сначала неверные решения, и это подвигло автора статьи [7] к разработке средств автоматической проверки доказательств и автоматизированной поддержки поиска доказательств (proof assistant) [8].

Введение диссертации [6] также содержит внушительный перечень важных примеров попыток решения проблем, которые подвигают исследователей к применению системы автоматизированной поддержки проверки и построения доказательств.

Далее, имеется особый новый (но не совсем полноценный) вид доказательств в математике: с привлечением вычисления, делаемого какой-то программой вычислительной алгебры. Естественно, мп-доказательство необходимых свойств такой программы делает целевое доказательство более полноценным.

9. ЗАКЛЮЧЕНИЕ

Написанная выше программа на языке `Agda` умножения многочленов способом Карацубы для разреженного представления и соответствующего особого алгоритма (раздел 3.2) обладает необходимой производительностью при исполнении, сопровождается машинно-проверяемым доказательством правильности, и для ее алгоритма доказана (в Приложении 11) требуемая оценка стоимости вычисления.

Испытание программы на производительность на разных примерах описано в разделе 6.

В разделе 4.2 объяснено, почему наличие “внутренних” доказательств в этой программе не влияет существенно на производительность исполняемого кода.

Исходный код программы имеет приемлемый объем и компилируется за приемлемое время (раздел 7).

Как и во всех системах доказательного программирования, существенной является проблема трудоемкости построения полных формальных доказательств.

Доказательная программа метода Карацубы для многочленов воплощена на основе библиотеки DoCon-A [1] доказательных программ вычислительной алгебры.

10. БЛАГОДАРНОСТИ

Исследование частично поддержано Министерством науки и высшего образования РФ, исследовательский проект АААА-А19-119020690043-9.

11. ПРИЛОЖЕНИЕ. ДОКАЗАТЕЛЬСТВО ОЦЕНКИ СЛОЖНОСТИ

Обозначим $C(n)$ наибольшую из стоимостей умножения способом Карацубы многочленов степени не больше n, когда стоимость выражена числом простейших действий над мономами (см. Введение).

Обозначим log логарифм по основанию 2. Будем доказывать оценку

$$C(n) \in O(n^{\log 3}) \qquad \text{(E.I)}$$

11.1. Случай плотного представления и степени двойки

Этот особый случай разбираем для показа основного замысла доказательства. И вводим определения, которые будут также использоваться в следующем разделе.

Здесь считаем, что представление многочленов плотное и оба сомножителя имеют степень $n = 2^l$ для некоторого натурального l.

Замысел доказательства для этого случая взят из видео в Интернете.

“Кружок – группа А – алгоритм Карацубы”

(А.С. Станкевич. Алгоритм Карацубы. Лекция на кружке олимпиадной информатики ИТМО, Санкт-Петербург) и преобразован здесь в следующее несколько более подробное рассуждение.

В данных обстоятельствах алгоритм Kara применяет только шаг (EE), и считаем, что на этом шаге степени f_2 и g_2 тоже равны $n/2$, так как искусственно добавляется моном степени $n/2$ с нулевым коэффициентом.

Обозначим a наибольшую стоимость умножения двух многочленов степени не больше 1 (при наших допущениях $a = 7$). Высказывание (E.I) по определению означает, что существует постоянная A такая, что для любого $n > 0$ выполнено неравенство

$$C(n) \le A \cdot n^{\log 3} \qquad \text{(E.II)}$$

Найдем такую постоянную A. Определим так называемое нами *дерево вычисления*. Каждая вершина этого дерева соответствует вызову функции kara и она помечается значением степени аргументов этого вызова. Эту степень назовем *степенью вершины*. Таким образом корневая вершина имеет степень n и расположена на нулевом уровне дерева.

Уровнем вершины в дереве назовем количество ребер на пути от корня к этой вершине. В частности, корень дерева находится на уровне 0.

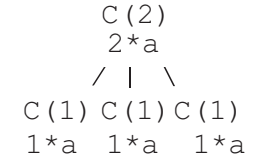
Согласно формуле (II) раздела 3.1 вычисление произведения двух многочленов степени n состоит из трех умножений многочленов степени $n/2$, одного прохода по списку мономов длины не более $n/2$ (splitPolAtDegree), умножения многочлена на x^n ценой не более n действий, умножения многочлена на $x^{n/2}$ ценой не более n действий, четырех сложений многочленов, каждое ценой не более n. Поэтому

$$C(n) \le an + 3C(n/2),$$

где a – постоянная – в наших допущениях она равна 7 – так же, как и стоимость $C(1)$. Эти три применения функции kara определяют три вершины уровня 1 дерева вычисления, их степени равны $n/2$. И так далее, получается троичное дерево. i-й уровень этого дерева содержит 3^i вершин, каждая степени $n/(2^i)$.

Эти три вызова функции kara для каждой вершины назовем *управляющими действиями* вершины, а остальные действия для этой вершины назовем *пересчетными действиями* вершины. Так, для корневой вершины имеются три управляющих действия для степени $n/2$, а пересчетные действия стоят вместе не больше an.

Стоимость вычисления вызова в корне дерева вычисления равна сумме стоимостей пересчетных действий по всем вершинам этого дерева. Стоимость управляющих действий в эту сумму не входит потому, что управляющее действие в вершине, не являющейся листом, для степени m только составляет три вызова функции kara для степени $m/2$. А стоимость вычисления этих вызовов определяется под-деревьями, исходящими из вершин этих вызовов. И это построение рекурсивно. Для наглядного показа правильности этого суждения рассмотрим пример вычисления произведения многочленов степени 2:



Корневая вершина вызывает функции, соответствующие трем вершинам, помеченным как C(1). Каждая из функций вершин, помеченных C(1), затрачивает не более a шагов вычисления – это только пересчетное действие стоимости не больше a. Вызов в корневой вершине делает три управляющих вызова, и это ничего не стоящие передачи управления. Но еще он делает вычисления над итогами этих вызовов, и эти пересчетные действия стоят не больше $2a$. Видно, что стоимость всего этого вычисления является суммой стоимостей пересчетных действий по всем вершинам.

Пересчетные действия корневой вершины имеют стоимость не больше an. Пересчетные действия для каждой вершины уровня 1 имеют стоимость не больше $an/2$. Вообще, пересчетные действия для каждой вершины уровня i имеют стоимость не больше $an/2^i$. Количество вершин уровня i равно 3^i. Поэтому для стоимости $C(n)$ всего вычисления верна оценка

$$\begin{aligned} C(n) &\le an + 3an/2 + 3^2 an/4 + \ldots + 3^{l-1} an/(2^{l-1}) = \\ &= an \cdot (1 + 3/2 + (3/2)^2 + \ldots + (3/2)^{l-1}) \end{aligned} \qquad \text{(10.I)}$$

По формуле суммы геометрической прогрессии, и ввиду равенства $2^l = n$, правая часть равна

$$\begin{aligned} &an((3/2)^l - 1)/(3/2 - 1) = \\ &= 2an((3/2)^l - 1) \le 2an(3/2)^l = \\ &= 2an3^l/n = 2a(2^{\log 3})^l = 2a(2^l)^{\log 3} = 2an^{\log 3} \end{aligned}$$

Подстановка $A = 2a$ в это неравенство доказывает утверждение (E.II).

11.2. Разреженное представление, общий случай

Построим дерево вычисления для алгоритма Kara (раздел 3.2) для разреженного представления в общем случае. Назовем степенью пары многочле-

нов максимум из степеней этих многочленов. Обозначаем ($kara\,d\,f\,g$) применение функции kara к многочленам f и g, где $d = \max(\deg f)(\deg g)$. Если аргументы f, g пропущены, то подразумеваются любые многочлены, удовлетворяющие данному условию.

Здесь ссылаемся на определения из раздела 11.1 и описание функций EE, GE, GO, EO из раздела 3.2. Запишем вычисление по алгоритму Kara в виде такого троичного дерева, в котором каждая вершина соответствует вызову kara некоторой степени d и либо является листом, либо имеет три "сына", и степень каждого из сыновей не больше $d/2$. Назовем такое дерево *правильным*.

Правильное дерево строится следующим образом. Пусть в текущей вершине стоит вызов ($kara\,d\,h\,h'$). Если степень вершины равна нулю, то это лист.

Разберем случай вершины ненулевой степени.

В случае EE непосредственно порождаются три сына, каждый степени не больше $d/2$. Еще делается пересчетное действие стоимости не больше $7d$ – как в разделе 11.1. По индукции, строится правильное дерево для каждого из этих сыновей, и все дерево получается правильным.

В случае GE вызывается $\mathrm{EE}(d, x^d + h, h')$. Это порождает три вызова kara степени не больше $d/2$, и их вершины ставятся в дерево. При этом делается пересчетное действие стоимости не больше $2d$, дополнительно к пересчетному действию для $\mathrm{EE}(d, x^d + h, h')$. По индукции, строятся правильные деревья для этих трех сыновей, и вместе с текущей вершиной получается правильное дерево.

В случае GO в зависимости от итогов двух сравнений степеней возможны четыре под-случая. В первом из них вызывается EE, в остальных вызывается GE. Но в каждом из этих вызовов степень не больше d. Первый под-случай порождает три сына, каждый степени не больше $d/2$. Каждый из остальных под-случаев, согласно пункту GE, порождает три сына, каждый степени не больше $d/2$. При этом в каждом под-случае делаются пересчетные действия. Нетрудно подобрать такую постоянную a_1, что стоимость пересчетных действий во всех под-случаях не превосходит величины $a_1 d$. Полученные три сына, каждый степени не больше $d/2$, добавляются к дереву. Как и выше, к сыновьям применяется индуктивное построение, и получается правильное дерево.

В случае EO вызывается $\mathrm{GO}(d)$, и как в предыдущем пункте, получается правильное дерево. При этом делается пересчетное действие стоимости не больше $2d$ сверх того, что требует случай GO.

Таким образом, построено правильное дерево для вычисления по алгоритму Kara, и для каждой его вершины стоимость пересчетных действий не превосходит $a'd$, где d степень вершины, $a' = a_1 + 2$.

Назовем полученное дерево $\mathfrak{T}'$, а дерево из раздела 11.1 назовем $\mathfrak{T}$.

В дереве $\mathfrak{T}$ на каждом уровне находится 3^i вершин, $i = 0, \ldots l$, и на одном уровне все пары многочленов имеют степень $n/2^i$ (понятие уровня, – или глубины вершины определено в разделе 11.1).

В дереве $\mathfrak{T}'$ вершины одного уровня i могут иметь разные степени многочленов, и притом разной четности. Ветки дерева могут иметь разную длину. Но **(а)** глубина дерева не превосходит величины $l = 1 + \log(n)$ – так как степень каждой вершины по меньшей мере в два раза больше степени каждого ее сына, **(б)** по той же причине степень в каждой вершине уровня i не превосходит $n/2^i$, **(в)** количество вершин на уровне i не превосходит 3^i.

Последнее свойство выполнено по той причине, что, если не учитывать значения степеней в вершинах, то дерево $\mathfrak{T}'$ получается из $\mathfrak{T}$ обрезанием некоторых веток. А для любого уровня в дереве при обрезании какой-либо из веток, количество вершин на этом уровне не увеличивается.

Так же, как для дерева $\mathfrak{T}$, стоимость вычисления по дереву $\mathfrak{T}'$ равна сумме стоимостей пересчетных действий уровней $i = 0, \ldots l - 1$. Для дерева $\mathfrak{T}$ стоимость пересчетных действий уровня i не превосходит $an(3/2)^i$. Ввиду свойств **(а)**, **(б)**, **(в)**, стоимость пересчетных действий уровня i для дерева $\mathfrak{T}'$ не превосходит $a'n(3/2)^i$. Поэтому рассуждение с суммой геометрической прогрессии, как в разделе 11.1, доказывает неравенство

$$C(n) \le 2a'n^{\log 3}$$

и оценку $C(Kara) \in O(n^{\log 3})$.

11.3. Об оценке по степени разреженности многочленов

Назовем *длиной* многочлена f (в разреженной записи) количество $\mathrm{S}(f)$ мономов в нем. И будем учитывать неравенство $\mathrm{S}(f) \le (\deg f) + 1$.

Ниже приведены нестрогие наводящие рассуждения для вывода оценки средней стоимости вычисления по алгоритму Kara (раздел 3.2) (для разреженного представления).

Обозначим $P(n, s)$ множество многочленов степени n, имеющих длину s. Рассмотрим функ-

цию $C(n, s)$ средней стоимости вычисления по алгоритму Kara, когда сомножители берутся из множества $P(n, s)$. На нашем опыте $C(n, s)$ быстро уменьшается при уменьшении s от $n+1$ до 0. Приведем нестрогие соображения, объясняющие это явление.

Рассмотрим частный случай, когда n является степенью двойки. Усредненное распределение мономов при выборе всех многочленов из множества $P(n, s)$ является равномерным на этом отрезке, то есть в усредненных сомножителях для алгоритма Kara степени мономов распределены равномерно на отрезке $[0, n]$. Поэтому сделаем допущение, что алгоритм в начале получает пару таких усредненных многочленов. Степень свободы остается только в выборе ненулевых коэффициентов при этих степенях. Поэтому выражение "в среднем" ниже означает "в среднем по всем наборам ненулевых коэффициентов при неизменных степенях (в количестве s штук), заданных изначально и расположенных равномерно на отрезке $[0, n]$".

Для этого частного случая дадим некоторое нестрогое доказательство оценки

$$C(n, s) \in \cdot O(s^2) \qquad (10.\text{II})$$

Из этой оценки, например, следует, что **a)** при длине меньше $\sqrt{n}$ порядок средней стоимости вычисления окажется не больше n, тогда как **b)** при наибольшей длине $n+1$ стоимость в наихудшем случае достигнет порядка не меньше $n^{3/2}$ (второе утверждение не доказано строго, оно взято из итогов испытаний программы).

Из принятых допущений следует, что дерево вычисления и степени многочленов в его вершинах в среднем получаются такие же, как в разделе 11.1 для плотных многочленов. Только записи многочленов – разреженные и количество мономов меньше. Рассмотрим дерево вычисления, описанное в разделе 11.1, и вывод оценки (10.I), который мы преобразуем в вывод оценки для выбранного здесь случая.

Стоимость вычисления согласно дереву вычисления равна сумме стоимостей пересчетных действий по всем вершинам. В корне дерева перемножаемые многочлены имеют длину $s \le n+1$. Рассмотрим пересчетные действия в корневой вершине.

Разделение списка мономов по степени $n/2$ в нашем случае занимает $s/2$ шагов. Многочлены f_1, f_2, g_1, g_2, полученные расщеплением, имеют длину около $s/2$.

Произведение $f_1 g_1$ имеет степень n и длину не больше $n+1$. Но здесь нам нужна верхняя оценка длины, выраженная через s. Это $(s/2)^2$ – количество произведений мономов из f_1 на мономы из g_1.

Поэтому умножение $x^n \cdot f_1 g_1$ монома на многочлен (из формулы Карацубы) имеет порядок стоимости не больше $(s/2)^2$.

Другие умножения на моном в этой части тоже стоят не больше $(s/2)^2$ – по тем же причинам.

Еще есть 3–4 сложения/вычитания многочленов длины не больше $(s/2)^2$. Порядок их стоимости не больше $(s/2)^2$.

Верхняя оценка в разделе 11.1 для пересчетных действий в корне дерева – это an. Соответственно, в рассматриваемом здесь случае для пересчетных действий в корне дерева получается верхняя оценка

$$bs^2/4,$$

где b постоянная, не сильно отличающаяся от a.

Дальше расщепление продолжается рекурсивно, развертывается троичное дерево вычисления, как в разделе 11.1. На уровне i количество вершин равно 3^i, каждая вершина имеет степень $n/2^i$. Длина перемножаемых многочленов в такой вершине в среднем равна $s/2^i$. Стоимость пересчетных действий в ней в равномерной по s оценке не больше $an/2^i$. А в оценке в среднем через величину s она не больше $b(s/2^i)^2 = bs^2/4^i$.

Поэтому оценка (10.I) из раздела 11.1 в случае оценки через длину s в среднем принимает вид

$$C(n, s) \le bs^2 \cdot (1 + 3/4 + (3/4)^2 + \ldots + (3/4)^{l-1}) =$$
$$= bs^2 \cdot (1 - (3/4)^l)/(1 - 3/4) =$$
$$= 4bs^2 \cdot (1 - (3/4)^l) \le 4bs^2.$$

Уточнение этой оценки (по худшему случаю или в среднем) для общего случая а также построение строго доказательства может быть предметом отдельного исследования.

СПИСОК ЛИТЕРАТУРЫ

1. Мешвелиани С.Д. DoCon-A. Библиотека доказательных программ компьютерной алгебры. Переславль-Залесский, 2021. http://www.botik.ru/pub/local/Mechveliani/docon-A/
2. *Карацуба А.А., Офман Ю.П.* Умножение многозначных чисел на автоматах // Доклады АН СССР. Т. 145. № 2. С. 293–294. http://www.mathnet.ru/links/d3321404ffd85ead867863cb5e410f00/dan26729.pdf
3. *Карацуба А.А.* Сложность вычисления. Тр. МИАН. 1995. Т. 211. С. 186–202. http://www.mathnet.ru/links/f3297054555b746ec1241c8805a22c62/tm1120.pdf
4. *Шёнхаге А., Штрассен В. (Schönhage A., Strassen V.).* Scnelle Multiplikation grosser Zahlen. Computing.

Т. 7. № 3–4. С. 281–292. Русский перевод: Кибернетический сборник, нов. сер. вып. 10. М.: Мир, 1973. С. 87–98.

5. Gathen J., Gerhard J. Modern Computer Algebra. 3rd ed. Cambridge University Press, 2013.
6. *Mörtberg A.* Formalizing Refinements and Constructive Algebra in Type Theory. Тезисы диссертации. Department of Computer Science and Engineering Chalmers University of Technology and University of Gothenburg SE-412 96 Gothenburg, Sweden Gothenburg, 2014. 142 с. https://www-sop.inria.fr/members/Anders.Mortberg/thesis/doc/v0.1/karatsuba.html
7. *Voevodsky V.* Univalent Foundations. Princeton, Nj, March 26, 2014. https://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/2014_IAS.pdf
8. *Voevodsky V.* UniMath. Библиотека программ в системе Coq, формализующая существенную часть математики с точки зрения юнивалентной теории. https://github.com/UniMath/UniMath
9. Norell U. Dependently Typed Programming in Agda. AFP 2008: Advanced Functional Programming, Lecture Notes in Computer Science, V. 5832. Springer, Berlin Heidelberg, 2008. P. 230–266.
10. Agda. A proof assistant. A dependently typed functional programming language and its system. http://wiki.portal.chalmers.se/agda/pmwiki.php.
11. *Мешвелиани С.Д.* О зависимых типах и интуиционизме в программировании математики. В электронном журнале Программные системы: теория и приложения. 2014. Т. 5. Вып. 3. С. 27–50. http://psta.psiras.ru/read/psta2014_3_27-50.pdf
12. *Martin-Loef, Per.* Intuitionistic type theory. Bibliopolis, ISBN 88-7088-105-9. 1984. 91 с.
13. Марков А.А. О конструктивной математике. Проблемы конструктивного направления в математике. 2. Конструктивный математический анализ, Сборник работ // Тр. МИАН СССР. Т. 67. М., Л.: Изд-во АН СССР, 1962. С. 8–14.
14. *Chlipala A.* Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant. MIT Press, 2013. http://adam.chlipala.net/cpdt/
15. *de Moura L., Kong S., Avigad J., van Doorn F., von Raumer J.* The Lean Theorem Prover. 25th International Conference on Automated Deduction (CADE-25), Berlin, Germany, 2015. https://leanprover.github.io/papers/system.pdf.
16. The Coq Effective Algebra Library. https://github.com/CoqEAL/CoqEAL/

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 23–33

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 517.5+004.434

ЛИНИИ УРОВНЯ МНОГОЧЛЕНА НА ПЛОСКОСТИ

© 2022 г. А. Д. Брюно[a,*], А. Б. Батхин[a,b,]**

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

[b] *Московский физико-технический институт (государственный университет), Институтский переулок, д. 9, 141701 Долгопрудный Московской области, Россия*

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

***E-mail: batkhin@gmail.com*

Поступила в редакцию 20.07.2021 г.
После доработки 13.08.2021 г.
Принята к публикации 15.09.2021 г.

Предлагается метод вычисления расположения всех типов линий уровня вещественного многочлена на вещественной плоскости. Для этого надо вычислить его критические точки и критические кривые, а затем – критические значения многочлена (их конечное число). По ним вычисляются все критические линии уровня и по одному представителю некритических линий уровня, соответствующих интервалам значений между соседними критическими. Предлагается схема вычислений линий уровня, основанная на алгоритмах полиномиальной компьютерной алгебры: базисах Гребнера, примарной декомпозиции идеала. Указано программное обеспечение для реализации этих вычислений. Разобраны нетривиальные примеры.

DOI: 10.31857/S0132347422010034

1. ВВЕДЕНИЕ

Пусть $X = (x_1, x_2) \in \mathbb{R}^2$. Рассмотрим вещественный многочлен $f(X)$. При постоянной $c \in \mathbb{R}$ кривая на плоскости $\mathbb{R}^2$

$$f(X) = c \tag{1.1}$$

является *линией уровня* многочлена $f(X)$.

Наша задача – описать все типы линий уровня многочлена $f(X)$ на вещественной плоскости $X \in \mathbb{R}^2$. Пусть $C_* = \inf f(X)$ и $C^* = \sup f(X)$ по $X \in \mathbb{R}^2$. Основной результат:

Теорема 1. *Имеется конечное множество критических значений c:*

$$C_* < c_1^* < c_2^* < \ldots < c_m^* < C^*, \tag{1.2}$$

которым соответствуют критические линии уровня

$$f(X) = c_j^*, \quad j = 1, \ldots, m, \tag{1.3}$$

а для значений c из каждого из $m+1$ интервала

$$I_0 = (C_*, c_1^*), \quad I_j = (c_j^*, c_{j+1}^*), \quad j = 1, \ldots, m-1, \quad I_m = (c_m^*, C^*) \tag{1.4}$$

линии уровня топологически эквивалентны. Если $C_* = c_1^*$ или $C^* = c_m^*$, то интервалы I_0 или I_m отсутствуют.

Поэтому для выявления расположения всех типов линий уровня многочлена $f(X)$ надо найти все критические значения c_j^*, изобразить m критических линий уровня (1.3) и по одной линии уровня для произвольного значения c из $m+1$ интервала (1.4). Способ вычисления этих линий уровня описан в [1] и отчасти в [2, гл. 1, § 2] с помощью степенной геометрии. Более традиционный подход см. в [3, гл. 1] или [4]. Локальное строение линий уровня многочлена рассматривалось в [2, гл. 1, § 3]. Здесь некоторые результаты из [2] дополнены.

2. КРИТИЧЕСКИЕ ТОЧКИ И КРИТИЧЕСКИЕ КРИВЫЕ

Точка $X = X^0$ называется *простой* для многочлена $f(X)$, если в ней отлична от нуля хотя бы одна из частных производных $\partial f/\partial x_1$, $\partial f/\partial x_2$.

Определение 1. *Точка $X = X^0$* для многочлена $f(X)$ называется *критической порядка k*, если в точке $X = X^0$ равны нулю все частные производные от $f(X)$ до порядка k, т.е. все

$$\frac{\partial^l f}{\partial x_1^i \partial x_2^j}(X^0) = 0, \quad 1 \le i + j = l \le k,$$

и отлична от нуля хотя бы одна частная производная порядка $k+1$.

Определение 2. *Кривая*

$$g(X) = 0 \tag{2.1}$$

называется *критической* для многочлена $f(X)$, если

1. Она лежит на какой-то линии уровня (1.1) и

2. На ней $\partial f/\partial x_1 \equiv 0$, или $\partial f/\partial x_2 \equiv 0$.

Значения постоянной $c = f(X)$ в критических точках $X = X^0$ и на критических кривых (2.1) назовем *критическими* и обозначим c_j^* согласно (1.2).

3. ЛОКАЛЬНЫЙ АНАЛИЗ ЛИНИЙ УРОВНЯ

В дальнейшем вблизи точки $X = X^0$ будем рассматривать аналитические обратимые замены координат

$$y_i = x_i^0 + \varphi_i(x_1 - x_1^0, x_2 - x_2^0), \quad i = 1,2, \tag{3.1}$$

где φ_i – аналитические функции от $X - X^0$.

Лемма 1. ([2, гл. 1, § 3]). *Если точка X^0 простая и в ней $\partial f/\partial x_2 \ne 0$, то существует замена* (3.1), *приводящая уравнение* (1.1) *к виду*

$$f(X) = y_2 = c. \tag{3.2}$$

Она следует из теоремы о неявной функции.

Линии уровня (3.2) – это прямые, параллельные оси y_1.

Рассмотрим решения уравнения (1.1) вблизи критической точки $X^0 = 0$ порядка 1. Тогда

$$f(X) = f_0 + ax_1^2 + bx_1x_2 + cx_2^2 + \dots$$

Дискриминант Δ выписанной квадратичной формы есть $\Delta = b^2 - 4ac$.

Лемма 2. ([2, гл. 1, § 3]). *Если в критической точке первого порядка $X^0 = 0$ дискриминант $\Delta \ne 0$, то существует замена* (6), *приводящая уравнение* (1.1) *к виду*

$$f(X) = f_0 + \sigma y_1^2 + y_2^2 = c, \tag{3.3}$$

где $\sigma = 1$ (если $\Delta < 0$) или $\sigma = -1$ (если $\Delta > 0$).

Это двумерный вариант известной леммы Морса [5, гл. 1, § 2].

Лемма 3. *Если в критической точке первого порядка $X^0 = 0$ дискриминант $\Delta = 0$, то существует замена* (3.1), *приводящая уравнение* (1.1) *к виду*

$$f(X) = f_0 + y_2^2 + \tau y_1^n = c, \tag{3.4}$$

где целое $n > 2$ и число $\tau \in \{-1, 0, +1\}$.

Доказательство. Сначала невырожденным линейным преобразованием $X = ZB$ приводим квадратичную часть к z_2^2. Разложение многочлена $f(X)$ в ряд по Z имеет вид

$$f = f_0 + z_2^2 + \sum_{q_1+q_2 \ge 3} f_Q Z^Q = \tilde{f}(Z).$$

Согласно теореме о неявной функции уравнение

$$\frac{\partial \tilde{f}(Z)}{\partial z_2} = 2z_2 + \dots = 0$$

имеет аналитическое решение

$$z_2 = \varphi(z_1).$$

Теперь сделаем замену

$$z_1 = w_1, \quad z_2 = w_2 + \varphi(w_1).$$

Тогда $\tilde{f}(Z) = \tilde{\tilde{f}}$ и $\partial\tilde{\tilde{f}}/\partial w_2 \equiv 0$ при $w_2 = 0$. Следовательно,

$$\tilde{\tilde{f}} = \varphi_0(w_1) + w_2^2(1 + h^{(2)}(W)),$$

где $\varphi_0(w_1) = \sum_{k \ge 3} \alpha_k w_1^k$ и $h^{(2)}(W)$ – степенной ряд от W без свободного члена. При этом возможны два случая:

1. $\varphi_0 \not\equiv 0$,

2. $\varphi_0 \equiv 0$.

В первом случае пусть n – это младшая степень w_1 в ряду $\varphi_0(w_1) = a_n w_1^n + \cdots$ Тогда после замены

$$y_1 = \sqrt[n]{\frac{\varphi_0(w_1)}{\operatorname{sign} a_n}}, \quad y_2 = w_2\sqrt{1 + h^{(2)}(W)}$$

получаем формулу (3.4) с $\tau = \pm 1 = \operatorname{sign} a_n$.

Во втором случае формулу (3.4) с $\tau = 0$ получаем после замены

$$y_1 = w_1, \quad y_2 = w_2\sqrt{1 + h^{(2)}(W)}.$$

Доказательство окончено. □

Выражения (3.3) и (3.4) – это *нормальные формы* многочлена $f(X)$ вблизи его критической точки первого порядка $X^0 = 0$.

Пусть теперь критическая точка $X^0 = 0$ имеет порядок $k > 1$. Согласно [1, раздел 5] соответствующая ей линия уровня либо не имеет ветвей, входящих в критическую точку $X^0 = 0$, либо имеет несколько таких ветвей.

В первом случае критическая линия уровня состоит из этой точки $X^0 = 0$, а остальные линии уровня являются замкнутыми кривыми вокруг нее и соответствуют одному знаку разности $c - f(X^0)$.

Во втором случае критическая линия уровня состоит из конечного числа ветвей разных кратностей, входящих в критическую точку $X^0 = 0$. Они разбивают окрестность этой критической точки на криволинейные секторы. Остальные линии уровня заполняют эти секторы, оставаясь на некотором расстоянии от критической точки $X^0 = 0$. При этом в соседних секторах они соответствуют разным знакам разности $c - f(X^0)$, если разделяющая их ветвь имеет нечетную кратность, и одному знаку этой разности, если разделяющая их ветвь имеет четную кратность.

4. ГЛОБАЛЬНЫЙ АНАЛИЗ ЛИНИЙ УРОВНЯ

Определение 3. Пусть ребро $\Gamma_j^{(1)}$ многоугольника Ньютона $\Gamma(f)$ многочлена $f(X)$ имеет внешнюю нормаль с одной или двумя положительными координатами и соответствует укороченному многочлену $f_j^{(1)}(X)$. Тогда пересечение корня $X = Tt^{\alpha}$, T, $\alpha = \mathrm{const}$, $t \to \pm\infty$, укороченного многочлена $f_j^{(1)}(X)$ с бесконечностью $x_i = \pm\infty$ назовем *бесконечной точкой пересечения*. Кратность корня – это *кратность* этой точки. Только в этих точках линии уровня достигают бесконечности.

У каждого многочлена имеется лишь конечное множество бесконечных точек пересечения и все они с конечными кратностями. Можно исследовать характер линий уровня вблизи бесконечных точек пересечения в зависимости от их кратности. Здесь нет места для такого исследования. Вблизи однократной бесконечной точки пересечения они устроены просто (пример 1). Вблизи двукратной бесконечной точки пересечения они устроены сложнее, как показано в примере 2. Трехкратная точка бесконечного пересечения разобрана в примере 3, а четырехкратная имеется в примере 4.

Лемма 4. *Для всех значений постоянной c из одного из $m + 1$ интервалов* (4) *линии уровня* (1.1) *топологически эквивалентны*.

Доказательство леммы основано на анализе линий уровня вблизи бесконечных точек пересечения, и здесь его не приводим. Оно будет опубликовано в отдельной статье.

Теорема, сформулированная во введении, следует из лемм 1–4 и описанных выше свойств линий уровня вблизи критической точки $X^0 = 0$ порядка $k > 1$.

5. ВЫЧИСЛЕНИЕ КРИТИЧЕСКИХ ЗНАЧЕНИЙ, ТОЧЕК И КРИВЫХ

В общем случае вычисление критических значений c_j^* и соответствующих им критических точек и кривых удобно проводить с использованием алгоритмов компьютерной алгебры, в первую очередь, с помощью базисов Грёбнера [6], [7, Гл. 2, 3].

Идеал, определяющий критические точки и кривые, состоит из следующих полиномов:

$$\mathcal{J} = \left\{ f(X) - c, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right\}. \qquad (5.1)$$

Согласно теореме 1 число критических значений c конечно. Тогда базис Грёбнера $\mathcal{GBJ}$ идеала (5.1) для чистого лексикографического порядка $x_1 \prec x_2 \prec c$ содержит полином $h(c)$, зависящий только от переменной c. Среди вещественных корней этого полинома следует искать критические значения c_j^*, для которых имеются вещественные критические линии уровня.

Как следует из раздела 2, аффинное многообразие, определяемое идеалом $\mathcal{GBJ}$, имеет размерность либо 0, либо 1. В первом случае критических кривых нет, ибо идеал $\mathcal{GBJ}$ нульмерен. Во втором случае некоторым критическим значениям c_j^* соответствуют критические кривые. В обоих случаях, согласно теореме о примарном разложении [7, Гл. 4] или [8, Гл. 4], идеал $\mathcal{GBJ}$ состоит из конечного числа примарных идеалов, каждый из которых задает либо критическую точку, либо критическую кривую. Определение размерности идеала легко выполняется с помощью вычисленного ранее базиса Грёбнера $\mathcal{GBJ}$.

В большинстве систем компьютерной алгебры имеются процедуры построения базисов Грёбнера для различных лексикографических порядков, а также некоторые дополнительные процедуры для проверки нульмерности идеала, вычисления его размерности и др. Подробнее рассмотрим как могут быть реализованы подобные вычисления в системе `Maple`. Можно воспользоваться пакетами `Groebner` и PolynomialIdeals, а также входящими в него процедурами: `Basis` – для вычисления базиса Гребнера, `HilbertDimension` – для вычисления размерности идеала, `IsZeroDimensional` – для проверки нульмерности идеала, `PrimaryDecomposition` – для примарного разложения идеала, `EquidimensionalDecomposition` – для декомпозиции идеала на идеалы различных размерностей.

Предлагаем следующий порядок вычислений.

1. Составляется идеал $\mathcal{J}$ и для него вычисляется базис Грёбнера $\mathcal{GBJ}$ с чисто лексикографическим порядком $x_1 \prec x_2 \prec c$ с использованием процедуры `Basis` пакета `Groebner`.

2. Базис $\mathcal{GBJ}$ позволяет найти все критические значения c_j^* с помощью полинома $h(c)$ в виде алгебраических чисел. Такой полином автоматически определяется для указанного выше лекси-

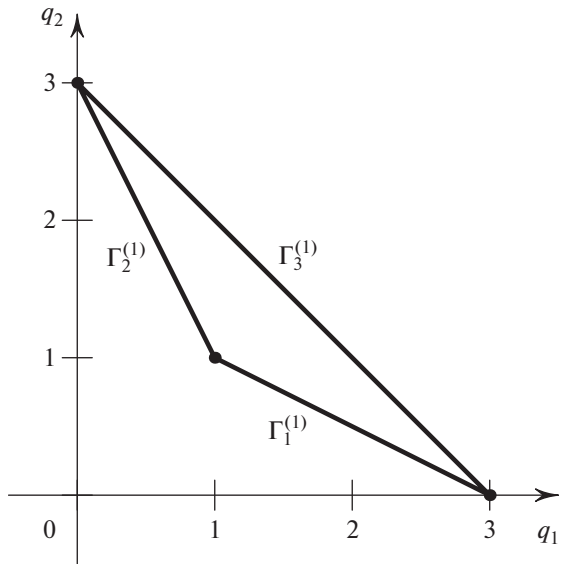


Рис. 1

кографического порядка, либо с помощью процедуры `UnivariatePolynomial`.

3. Среди критических значений c_j^* нужно отобрать те, которым соответствуют вещественные критические точки и критические кривые. Это можно сделать, например, выполнив примарную декомпозицию базиса $\mathcal{GBJ}$, а затем определить нули полученных идеалов в поле $\mathbb{R}$.

4. Перед примарной декомпозицией идеала $\mathcal{GBJ}$ можно вычислить его размерность с помощью процедуры `HilbertDimension`. Если она не равна нулю, то вначале с использованием процедуры `EquidimensionalDecomposition` вычисляется последовательность идеалов с размерностями 0 и 1 соответственно, а затем уже выполняется их декомпозиция. Идеалы с размерностью 0 дадут критические точки, а с размерностью 1 – критические кривые (а также вещественные критические точки).

5. Теперь для каждого примарного идеала следует найти его множество вещественных нулей. Если критическое значение c_j^* принадлежит $\mathbb{Q}$, то все вычисления выполняются точно. Если же оно есть алгебраическое число, то можно поступить так, как описано в [9, п. 5.6], т.е. проводить все вычисления по модулю идеала, определяющего критическое значение и критическую точку.

6. Теперь характер каждой критической точки определяется с использованием дискриминанта Δ квадратичной формы разложения функции $f(X)$ вблизи нее согласно леммам 2 и 3.

Выполненные вычисления позволяют перейти к построению эскизов линий уровня.

6. ПОСТРОЕНИЕ ЭСКИЗОВ ЛИНИЙ УРОВНЯ

Для построения эскиза линии уровня можно воспользоваться какой-либо системой компьютерной алгебры, имеющей программы построения изолиний (изоповерхностей) для двумерных или трехмерных скалярных полей. Эти программы используют различные вычислительные алгоритмы, основанные на применении конечных элементов, которые триангулируют (покрывают) некоторую часть плоскости (обычно используются треугольные или квадратные конечные элементы) [10]. Затем вычисляют значения функции (1.1) в вершинах сетки и они интерполируются на весь конечный элемент. Такие алгоритмы хорошо справляются с ситуацией, когда у линии уровня нет особенностей. Наличие особенностей заставляет существенно уменьшать шаг разбиения и, соответственно, увеличивать объем вычислений. В примерах раздела 7 все рисунки линий уровня построены с использованием процедуры `contour` пакета `matplotlib` [11] для языка программирования Python.

В ряде случаев удается улучшить качество эскиза линии уровня, если для определенного значения c уравнение (1.1) может быть разложено на множители, а нули этих множителей (или часть их) задают алгебраическую кривую рода 0. В этом случае можно вычислить рациональную параметризацию такой кривой [12], а по ней построить эскиз с любой точностью. С такой задачей неплохо справляется пакет `algcurves` системы `Maple`. Этот пакет, в частности, позволяет исследовать плоские алгебраические кривые. С его помощью можно изобразить эскиз кривой $f(x_1, x_2) = 0$ путем численного интегрирования соответствующего дифференциального уравнения $\dfrac{\partial f}{\partial x_1} + \dfrac{\partial f}{\partial x_2} \cdot \dfrac{dx_2}{dx_1} = 0$ для некоторого набора начальных условий, определяемых точками, в которых хотя бы одна из частных производных функции $f(x_1, x_2)$ равна нулю. Использование данного пакета для исследования набора кривых с разными порядками особенностей показало, что в случае особенностей высоких порядков качество эскиза получается не слишком высоким.

7. ПРИМЕРЫ

Пример 1. Рассмотрим многочлен

$$f_1(X) = x_1^3 - 3x_1x_2 + x_2^3.$$

Уравнение $f_1(X) = 0$ определяет алгебраическую кривую, которую называют “Декартов лист”. Многоугольник Ньютона $\Gamma(f)$ показан на рис. 1.

Он содержит три ребра, но только ребро $\Gamma_3^{(1)}$ имеет внешнюю нормаль $N_3 = (1, 1)$ с положи-

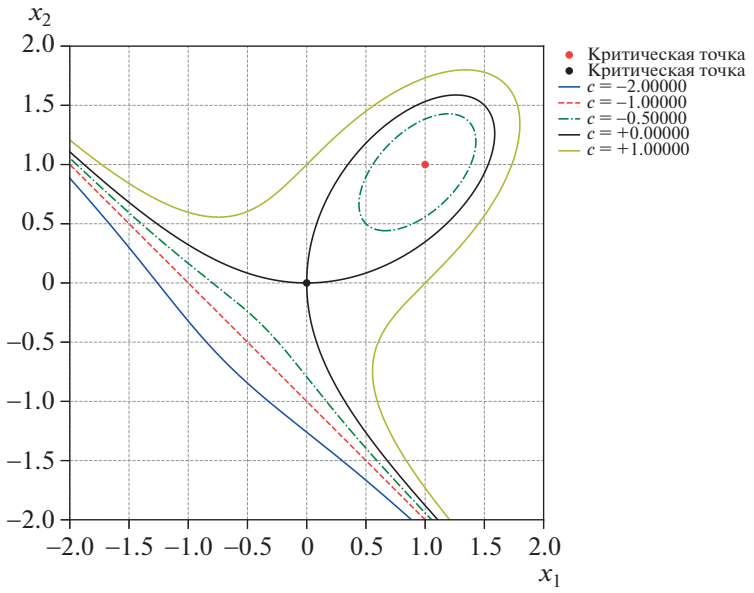


Рис. 2

тельной координатой. Соответствующий укороченный многочлен

$$\hat{f}_3^{(1)}(X) = x_1^3 + x_2^3 = (x_1 + x_2)(x_1^2 - x_1x_2 + x_2^3)$$

имеет однократный корень

$$X = (1, -1)t. \tag{7.1}$$

Согласно схеме вычислений раздела 5 вычисляем базис Грёбнера $\mathcal{GBJ}$ идеала (5.1) и определяем его многочлен $h(c) = c(c+1)$. Идеал имеет нулевую размерность, его примарная декомпозиция дает три идеала:

$$\mathcal{GBJ}_1 = \{c, x_1, x_2\}, \tag{7.2}$$

$$\mathcal{GBJ}_2 = \{c+1, x_1 - 1, x_2 - 1\}, \tag{7.3}$$

$$\mathcal{GBJ}_3 = \{c+1, x_1^2 + x_1 + 1, x_1 + x_2 + 1\}. \tag{7.4}$$

Идеал (7.3) дает критическую точку $X_1 = (1, 1)$ с критическим значением $c_1^* = -1$ и дискриминантом $\Delta_1 = -27 < 0$ (изолированная точка), идеал (7.2) – критическую точку $X_2 = 0$ с критическим значением $c_2^* = 0$ и дискриминантом $\Delta_2 = 9 > 0$ (в ней пересекаются две ветви). Идеал (7.4) не имеет вещественных нулей. Линии уровня показаны на рис. 2.

При $x_1 \to \pm\infty$ линии уровня стремятся к $x_2 \to \mp\infty$, накапливаясь у прямой (7.1) или у линии уровня с $c_2^* = 0$.

Заметим, что для $c \notin \{-1, 0\}$ кривая $f_1(X) - c = 0$ имеет род 1, т.е. является эллиптической кривой, уравнение которой может быть приведено к нормальной форме Вейерштрасса

$$y_2^2 = 4y_1^2 + \frac{27}{2}\left(4c - \frac{1}{2}\right)y_1 + \frac{27}{8}(1 + 20c - 8c^2) \tag{7.5}$$

с помощью преобразования

$$x_{1,2} = \frac{-12y_1 \pm 4y_2 + 36c + 9}{24y_1 + 54}.$$

Параметризация нормальной формы (7.5) задается с помощью функции Вейерштрасса $\wp(z, g_2, g_3)$

$$y_1 = \wp(z, g_2, g_3), \quad y_2 = \wp'(z, g_2, g_3),$$

инварианты g_2, g_3 которой суть коэффициенты, взятые со знаком минус, правой части уравнения (7.5) при степенях 1 и 0 переменной y_1 соответственно. Эта параметризация может быть использована для визуализации линий уровня, однако требует подготовительной работы – вычисления периодов и определения множества вещественности функции $\wp(z, g_2, g_3)$. Авторы не считают возможным уделить здесь достаточно внимания этим вопросам и рекомендуют обратиться к соответствующим источниками, например, к книге [13].

Критические значения $c_1^* = -1$ и $c_2^* = 0$ как раз соответствуют ситуации, когда кривая $f_1(X) - c = 0$ имеет род 0 и допускает рациональную параметризацию. При $c = c_1^*$ уравнение линии уровня факторизуется на линейный и квадратичный множители:

$$f_1 + 1 = (x_1 + x_2 + 1)(x_1^2 - x_1x_2 + x_2^2 - x_1 - x_2 + 1),$$

нуль последнего есть критическая точка $X_1 = (1, 1)$. При $c = c_2^*$ имеем рациональную параметризацию Декартового листа

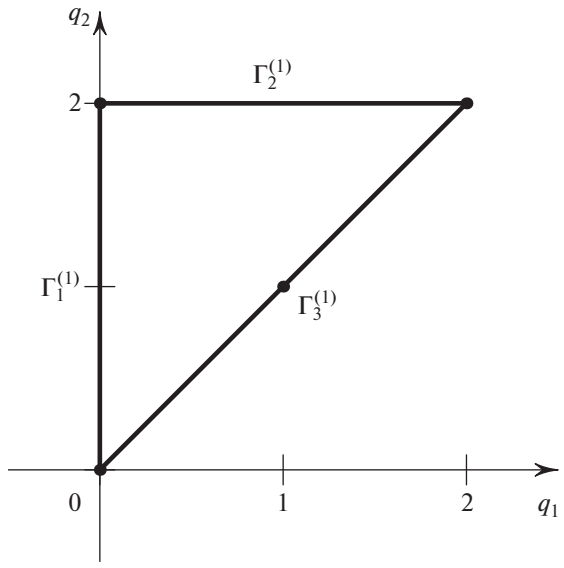


Рис. 3

$$x_1 = \frac{3t}{1+t^3}, \quad x_2 = \frac{3t^2}{1+t^3}.$$

Пример 2. Рассмотрим многочлен

$$f_2 = x_2^2 + (x_1x_2 - 1)^2. \tag{7.6}$$

Его многоугольник Ньютона показан на рис. 3.

Он состоит из трех ребер, но только двум из них соответствуют внешние нормали с положительной координатой – это ребра $\Gamma_2^{(1)}$ и $\Gamma_3^{(1)}$. Ребру $\Gamma_2^{(1)}$ соответствует укороченный многочлен

$$\hat{f}_2^{(1)} = x_2^2 + x_1^2x_2^2 = x_2^2(1 + x_1^2).$$

Он не имеет вещественного корня, уходящего в бесконечность.

Ребру $\Gamma_3^{(1)}$ соответствует укороченный многочлен

$$\hat{f}_3^{(1)} = (x_1x_2 - 1)^2,$$

имеющий двукратный корень

$$x_2 = \frac{1}{x_1},$$

уходящий в точки $x_1 = \pm\infty$, $x_2 = 0$. Линии уровня многочлена (7.6) показаны на рис. 4.

Базис Гребнера $\mathcal{GBJ}$ идеала (5.1) для многочлена (7.6) имеет вид $\{c - 1, x_1, x_2\}$, и, следовательно, здесь имеется одна критическая точка $x_1 = x_2 = 0$ с критическим значением $c_1^* = 1$. Линия уровня $f_2(X) = 1$ состоит из прямой $x_2 = 0$ и рациональной кривой $x_2 = 2x_1/(x_1^2 + 1)$, ее пересекающей. Справа между этой кривой и осью x_1 находятся линии уровня с $c \in (0, 1)$, но нет критической точки. При $c \to 0$ эти линии расположены все правее и как бы сложены. Таково поведение линий уровня вблизи бесконечной точки пересечения $x_1 = \infty$, $x_2 = 0$ кратности 2. Вблизи бесконечной точки пересечения $x_1 = -\infty$, $x_2 = 0$ кратности 2 расположение линий уровня симметрично относительно начала координат.

Кстати, здесь $C_* = \inf f(X) = 0$, ибо на кривой $x_2 = 1/x_1$ значения $f(X) = x_2^2 = 1/x_1^2$, они стремят-

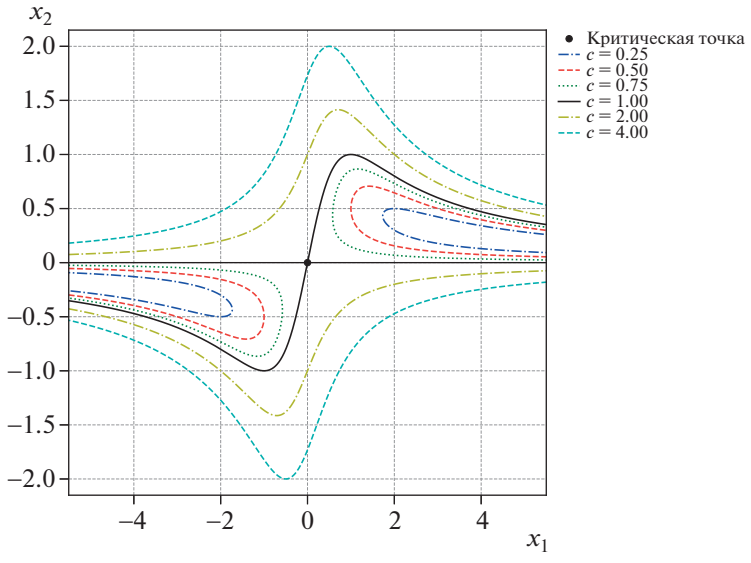


Рис. 4

ся к нулю при $x_1 \to +\infty$. При этом $C^* = +\infty$, что видно при $x_1 = 0$.

Заметим, что поскольку многочлен f_2 является суммой квадратов, то для любого $c > 0$, $c \neq 1$, легко построить рациональную параметризацию кривой $f_2(X) - c = 0$, используя известную рациональную параметризацию окружности. Для этой кривой она имеет вид

$$x_1 = \frac{2\sqrt{c}\tau + \tau^2 + 1}{\sqrt{c}(1 - \tau^2)}, \quad x_2 = \frac{\sqrt{c}(1 - \tau^2)}{\tau^2 + 1}.$$

При $\tau \in (-1, 1)$ получаем линию уровня в верхней полуплоскости, при $\tau \notin [-1, 1]$ – в нижней.

Пример 3. Рассмотрим вычисление линий уровня многочлена

$$\begin{aligned} f_1(X) = {} & x_1^5 + 2x_1^4x_2 + 4x_1^4 + x_1^3x_2^2 + x_1^3x_2 + 4x_1^3 + \\ & + x_1^2x_2^3 - 6x_1^2x_2^2 - 12x_1^2x_2 + 2x_1x_2^4 + x_1x_2^3 - \\ & - 12x_1x_2^2 - 12x_1x_2 + x_2^5 + 4x_2^4 + 4x_2^3 = \\ & = (x_1^3 + x_2^3 - 3x_1x_2)(x_1 + x_2 + 2)^2. \end{aligned} \tag{7.7}$$

Здесь $C_* = -\infty$, $C^* = +\infty$. Они достигаются при $x_1 = 0$.

Базис Грёбнера $\mathcal{GBJ}$ идеала (5.1) здесь не приводим в силу его громоздкости, а многочлен $h(c)$ вычисленного базиса имеет вид

$$h(c) = c\,(c + 1)(3125c^2 + 56736c + 54000). \tag{7.8}$$

Среди его корней отбираем те критические значения c_j^*, для которых соответствующие значения будут вещественными. Можно поочередно к ранее вычисленному идеалу $\mathcal{GBJ}$ добавлять по одному множителю многочлена (7.8) и вычислять базис Гребнера с лексикографическим порядком $c \prec x_1 \prec x_2$. Только для множителя $c + 1$ получаем идеал $\{3x_2^2 + 3x_2 + 1, x_1 + x_2 + 1, c + 1\}$, который не имеет вещественных нулей. Таким образом, критическими значениями будут:

$$c^* \in \left\{-\frac{28368}{3125} - \frac{3036\sqrt{69}}{3125}, -\frac{28368}{3125} + \frac{3036\sqrt{69}}{3125}, 0\right\}.$$

Значению $c_1^* \approx -17.1478$ соответствует критическая точка $X_1 : (x_1^{(1)} = x_2^{(1)} = (3 + \sqrt{69})/10)$. В ней дискриминант $\Delta \approx -14221.2 < 0$, и она является изолированной.

Значению $c_2^* \approx -1.0077$ соответствует критическая точка $X_2 : (x_1^{(2)} = x_2^{(2)} = (3 - \sqrt{69}/10)$. В ней дискриминант $\Delta \approx 1.3415 > 0$ и через нее проходят две ветви. Критическим значениям $c_{1,2}^*$ соответствуют случаи леммы (2).

Значению $c_3^* = 0$ соответствует критическая точка $X_3 : (x_1^{(1)} = x_2^{(1)} = 0)$. В ней дискриминант $\Delta \approx 144 > 0$, и через нее проходят две ветви.

Наконец, при $c_3^* = 0$ имеется критическая прямая $x_1 + x_2 + 2 = 0$.

Заметим, что многочлен (7.7) раскладывается на два множителя: $x_1^3 + x_2^3 - 3x_1x_2$ и $(x_1 + x_2 + 2)^2$. Нулям первого соответствует лист Декарта, нулям второго – пара совпадающих прямых. Используя это разложение многочлена $f(X)$ на множители, можно легко вычислить все три указанные критические точки и критическую прямую $x_1 + x_2 + 2 = 0$. На ней тождественно аннулируются обе частные производные $\partial f_1/\partial x_1$ и $\partial f_1/\partial x_2$.

Критические линии уровня для многочлена (7.7) показаны на рис. 5 и 6 в разных масштабах. Критические точки X_1, X_2 и $X_3 = 0$ и критическая прямая $x_1 + x_2 + 2 = 0$ показаны полужирными.

Линия уровня для критического значения $c_1^* \approx -17.14$ изображена пунктирной. На рис. 5 она состоит из точки $X^{(1)}$ и кривой в левом нижнем углу. На рис. 6 она имеет еще две ветви в левом верхнем и правом нижнем углах.

Линия уровня для критического значения $c_2^* \approx -1$ показана штрих-пунктирной. Она имеет 3 компоненты: овал в первом квадранте, две пересекающиеся ветви выше критической линии и одну кривую ниже нее.

Линия уровня для критического значения $c_3^* = 0$ показана сплошной. Она состоит из двух компонент: критической прямой и листа Декарта.

Четыре типа некритических линий уровня легко восстанавливаются, как лежащие целиком между соседними критическими.

Многоугольник Ньютона многочлена (7.7) показан на рис. 7.

У него только одно ребро $\Gamma_1^{(1)}$ соответствует бесконечности. Ему соответствует укороченный многочлен

$$\begin{aligned} \hat{f}_1^{(1)} &= (x_1^3 + x_2^3)(x_1 + x_2)^2 = \\ &= (x_1 + x_2)^3(x_1^2 - x_1x_2 + x_2^2). \end{aligned}$$

Он имеет трехкратный корень $x_2 = -x_1$.

При $x_1 \to +\infty$ имеются четыре критических линии: две для $c = c_3^* = 0$ (включая критическую прямую $x_1 + x_2 + 2 = 0$) и две ветви для $c = c_2^*$, к которым скапливаются остальные линии уровня. Некоторые из них имеют складку. Аналогичная картина имеет место при $x_1 \to -\infty$. Характер этих линий уровня можно видеть на рис. 6.

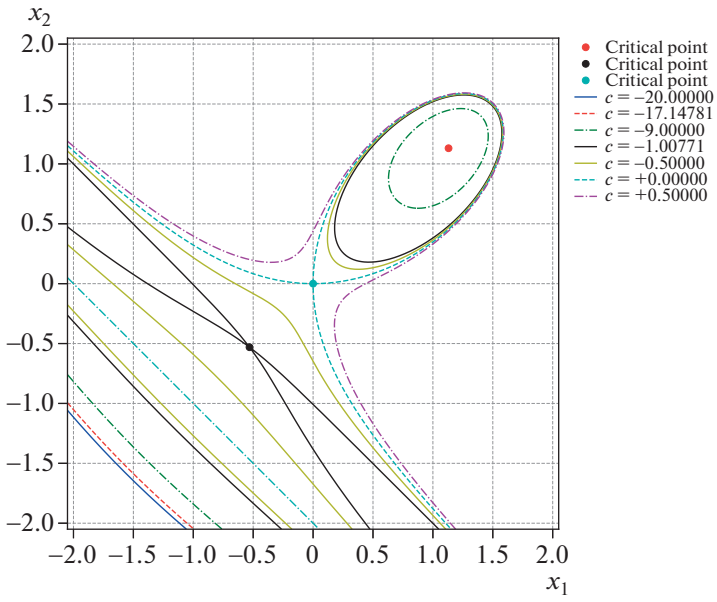


Рис. 5

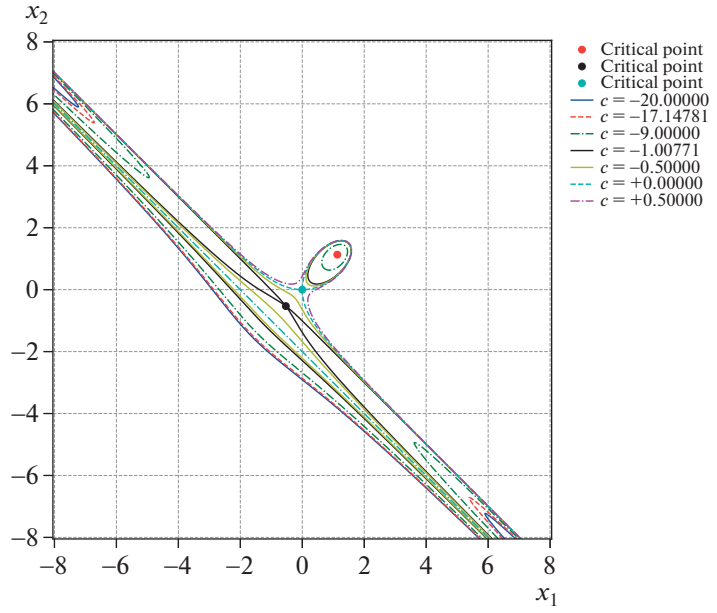


Рис. 6

Пример 4. В работах авторов [14–16] рассматривалась одна вещественная поверхность в $\mathbb{R}^3$, соответствующая особым точкам инвариантных метрик Эйнштейна. На плоскости в $\mathbb{R}^3$, где многочлен имеет кратный корень, сечение поверхности задается нулями многочлена

$$f_4(X) = -(1+2x_2)(8x_1x_2 + 8x_2^2 - 4x_1 - 4x_2 + 1)\times \\ \times (16x_1^3 + 16x_1^2x_2 - 4x_1 - 2x_2 + 1)^3. \quad (7.9)$$

Здесь, как и в примере 1, $C_* = -\infty$, $C^* = +\infty$.

Базис Гребнера $\mathcal{GBJ}$ идеала (5.1), составленный для многочлена (7.9), здесь не приводим в силу его громоздкости, а соответствующий многочлен $h(c)$ вычисленного базиса имеет вид

$$h(c) = c(c+16)(8707129344\times 10^{10}c^4 + \\ + 861132808668019359744c^3 -$$

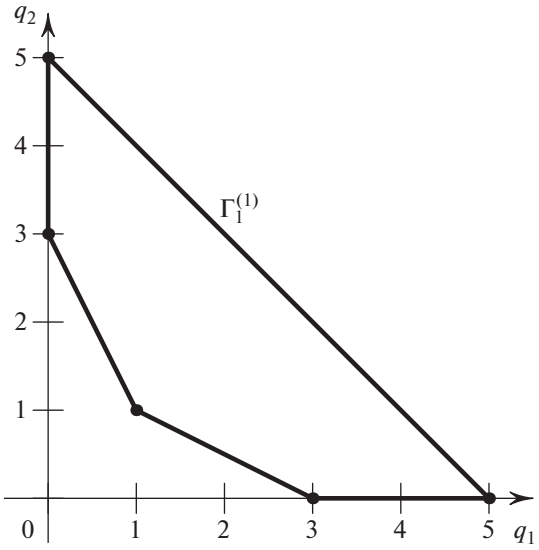


Рис. 7

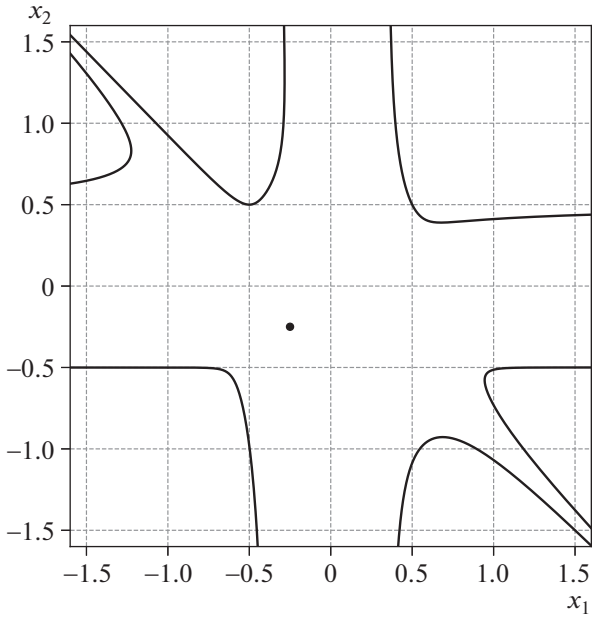


Рис. 8

$$- 956419978944596480c^2 +$$
$$+ 4296319584629409c - 940369969152).$$

Многочлен четвертой степени имеет два вещественных корня, и, следовательно, множество критических значений c_j^* состоит из значений

$$c^* \in \{-16, -9.8910848, 0, 0.00022808\}.$$

Перестройка линий уровня происходит при этих четырех значениях.

Размерность идеала $\mathcal{GBJ}$ равна единице, поэтому здесь вначале находится его декомпозиция на два идеала размерностей 0 и 1 соответственно, а затем уже выполняется их примарное разложение. Для критического значения $c_3^* = 0$ структура критических точек и кривых, а также линий уровня была описана в цитируемых выше работах и будет дана ниже. Поэтому рассмотрим структуру примарных идеалов, соответствующих другим критическим значениям параметра c.

Значению $c_1^* = -16$ соответствует критическая точка $x_1^{(1)} = x_2^{(1)} = -1/4$. В ней дискриминант $\Delta_1 = -204800 < 0$ и она является изолированной. Линии уровня для значения $c = c_1^*$ показаны на рис. 8.

Критические значения $c_2^* \approx -9.89108$ и $c_4^* \approx 0.000228$ являются корнями многочлена 4-й степени, т.е. они алгебраические числа. Вычисления показывают, что при $c = c_2^*$ имеется особая точка $X_2 \approx (-0.4021306894, 0.2360338414)$, в которой дискриминант $\Delta_2 \approx 63676.18108 > 0$ и, следовательно, в ней происходит пересечение линий уровня (см. рис. 9).

При $c = c_4$ имеется особая точка $X_4 \approx \approx (0.8995846960, -0.8095257544)$, в которой дискриминант $\Delta_4 \approx -0.01855315801 < 0$, т.е. она изолированная (см. рис. 10).

Наконец, как следует из (7.9), при $c = c_3^* = 0$ многочлен $f_4(X)$ раскладывается на множители.

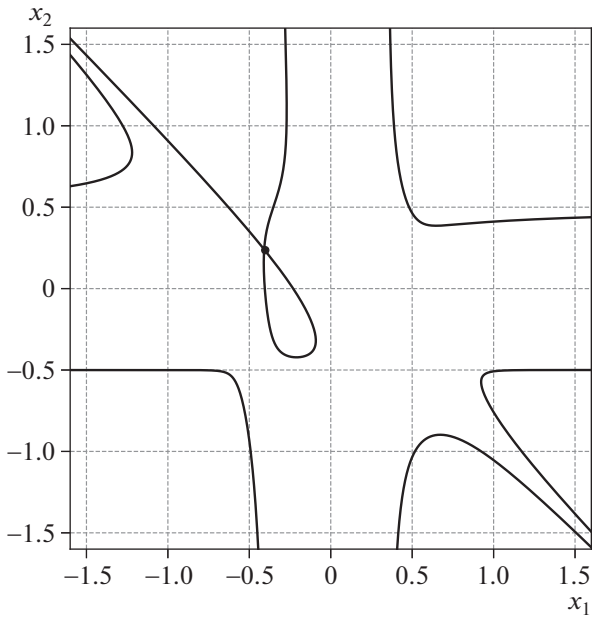


Рис. 9

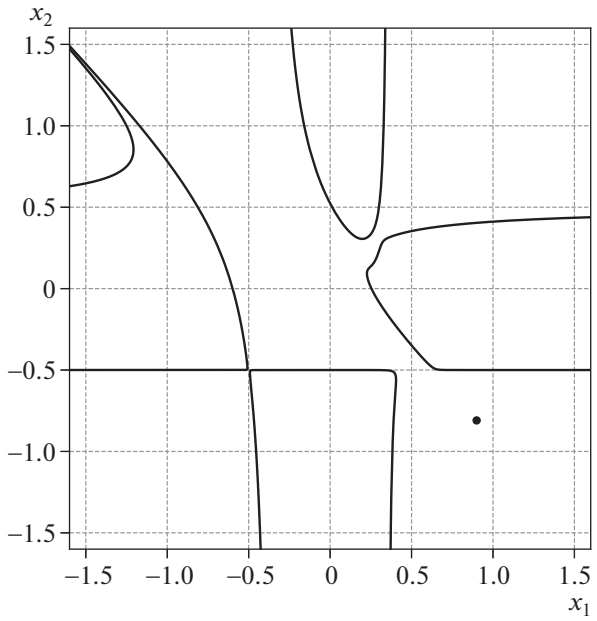


Рис. 10

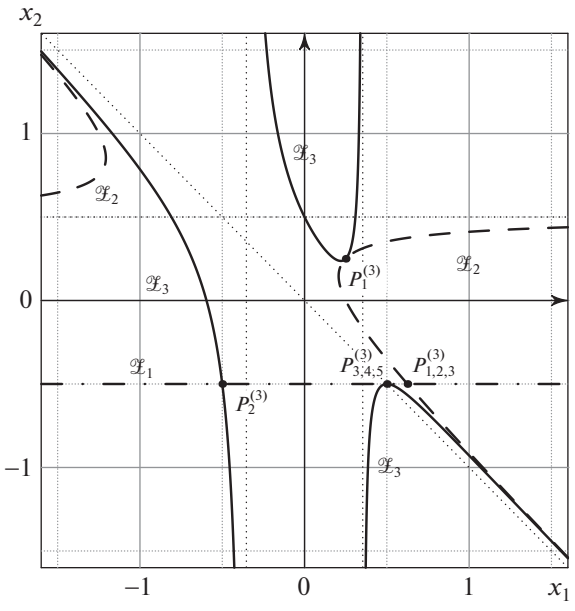


Рис. 11

В этом случае имеются критические точки

$$P_1^{(3)} = \left(\frac{1}{4}, \frac{1}{4}\right), \quad P_2^{(3)}\left(-\frac{1}{2}, -\frac{1}{2}\right),$$
$$P_{3,4,5}^{(3)} = \left(\frac{1}{2}, -\frac{1}{2}\right), \quad P_{1,2,3}^{(1)} = \left(\frac{5}{8}, -\frac{1}{2}\right),$$

которые суть точки пересечения рациональных кривых $\mathscr{L}_j$, $j = 1, 2, 3$, задаваемых нулями соответствующих множителей формулы (7.9), а также критическая кривая $\mathscr{L}_3$ (см. рис. 11). Критической кривой $\mathscr{L}_3$ (показана сплошной линией на рис. 11) соответствует тройной корень. Здесь сохранены обозначения, используемые в указанных выше работах авторов.

Для анализа критических линий на бесконечности построим многоугольник Ньютона многочлена (7.9) (см. рис. 12) и выделим укороченные многочлены, соответствующие ребрам, чьи нормали имеют хотя бы одну положительную координату.

Подходящие ребра многоугольника обозначены $\Gamma_1^{(1)}$, $\Gamma_2^{(1)}$, $\Gamma_3^{(1)}$ на рис. 12, а соответствующие им укороченные многочлены суть

$$\hat{f}_1^{(1)} = -128x_2^6(8x_1^2 - 1)^3, \tag{7.10}$$

$$\hat{f}_2^{(1)} = -65536x_1^6x_2^2(x_1 + x_2)^4, \tag{7.11}$$

$$\hat{f}_2^{(1)} = -16384x_1^{10}(2x_2 - 1)(2x_2 + 1). \tag{7.12}$$

Следовательно, из разложения многочлена (7.10) следует, что на бесконечности имеются трехкратные корни $X = (\pm 1/\sqrt{8}, \infty)$. Из разложения многочлена (7.11) следует наличие четырехкратного корня $x_2 = -x_1$ при $x_1 \to \pm\infty$. Наконец, из разложения многочлена (7.12) следует наличие простых корней $X = (\pm\infty, \pm 1/2)$. Эти корни легко отслеживаются в виде вертикальных, наклонных и горизонтальных асимптот на рис. 8–11, причем на последнем рисунке они показаны точечными линиями.

Напоследок опишем перестройку линий уровня многочлена (7.9) при изменении c от c_3^* к c_4^* и далее. При $c > c_3^*$ ветви кривых $\mathscr{L}_2$ и $\mathscr{L}_3$, идущие от критических точек $P_{1,2,3}^{(1)}$ и $P_{3,4,5}^{(3)}$ соответственно в правый нижний угол рис. 12, соединяются и образуют замкнутую кривую. Эта кривая, по мере

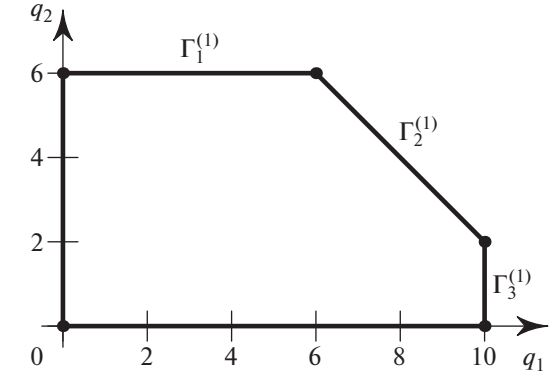


Рис. 12

приближения c к значению c_4^* снизу, стягивается к изолированной критической точке X_4.

В левом верхнем углу рис. 11 ветви кривых $\mathscr{L}_2$ и $\mathscr{L}_3$, идущие вдоль асимптоты $x_2 = -x_1$, также соединяются. Таким образом, при $c > c_4^*$ не остается линий уровня, уходящих на бесконечность вдоль асимптоты $x_2 = -x_1$. В этом можно убедиться, если построить разложение $f_4(X) - c$ в ряд Пюизе при $x_1 \to \infty$, например, используя процедуру puisex из пакета algcurves системы Maple. Это разложение имеет три ветви:

$$x_2 = -\frac{1}{2}, \tag{7.13}$$

$$x_2 = \frac{1}{2} - \frac{1}{8x_1} + \frac{1}{16x_1^2} + \cdots, \tag{7.14}$$

$$x_2 = -x_1 + \frac{1}{8x_1} + \frac{\alpha}{x_1^2} + \cdots, \tag{7.15}$$

где α есть корень многочлена

$$2^{16}y^4 + 2^14y^3 + 3 \cdot 2^9y^2 + 2^6y + c + 1. \tag{7.16}$$

Разложение (7.13) соответствует прямой $\mathscr{L}_1$, разложение (7.14) соответствует ветви кривой $\mathscr{L}_2$ при $x_1 \to \infty$, $x_2 \to 1/2$. Несложно показать либо с использованием последовательности Штурма, либо с помощью теоремы Якоби–Борхарда (см., например, [17, Гл. XVI, § 2]), либо вычисляя последовательность субдискриминантов многочлена (7.16) [18] и применяя теорему 4.33 из [19], что при $c > 0$ многочлен (7.16) имеет только комплексные корни. Следовательно, разложение (7.15) не соответствует никакой вещественной кривой.

СПИСОК ЛИТЕРАТУРЫ

1. *Брюно А.Д., Батхин А.Б.* Введение в нелинейный анализ алгебраических уравнений // Препринты ИПМ им. М.В. Келдыша. 2020. № 87. 31 с.
2. *Брюно А.Д.* Локальный метод нелинейного анализа дифференциальных уравнений. М.: Наука, 1979. 252 с.
3. *Kollár J.* Lectures on Resolution of Singularities. Princeton and Oxford: Princeton University Press, 2007.
4. *Kollár J.* Resolution of Singularities – Seattle Lecture. 2007. math/0508332v3.
5. *Милнор Д.* Теория Морса: Пер. с англ. 3-е изд. М. : Издательство ЛКИ, 2011. 184 с.
6. *Buchberger B.* A theoretical basis for the reduction of polynomials to canonical forms // ACM SIGSAM Bulletin. 1976. V. 10. № 3. P. 19–29.
7. *Кокс Д., Литтл Дж., О'Ши Д.* Идеалы, многообразия и алгоритмы. Введение в вычислительные аспекты алгебраической геометрии и коммутативной алгебры. М.: Мир, 2000. 687 с.
8. *Атья М., Макдональд И.* Введение в коммутативную алгебру. М.: Мир, 1972. 160 с.
9. *Брюно А.Д., Батхин А.Б.* Алгоритмы и программы вычисления корней многочлена от одной или двух неизвестных // Программирование. 2021. № 5. С. 22–43.
10. *Singh C., Singh J.* Accurate contour plotting using 6-node triangular elements in 2D // Finite Elements in Analysis and Design. 2009. V. 45. № 2. P. 81–93.
11. *Hunter J.D.* Matplotlib: A 2D graphics environment // Computing in Science & Engineering. 2007. V. 9. № 3. P. 90–95.
12. *Hoeij M.* Rational parametrizations of algebraic curves using a canonical divisor // J. Symbolic Computation. 1997. V. 23. P. 209–227.
13. *Lawden D.F.* Elliptic Functions and Applications. New York, Berlin, Heidelberg, London, Paris, Tokyo, Hong Kong: Springer-Verlag, 1989. V. 80 of *Applied Mathematical Sciences*. 350 p.
14. *Батхин А.Б., Брюно А.Д.* Исследование одной вещественной алгебраической поверхности // Программирование. 2015. № 2. С. 7–17.
15. *Батхин А.Б.* Глобальные параметризации одной вещественной алгебраической поверхности // Препринты ИПМ им. М.В. Келдыша. 2016. № 76. 24 с.
16. *Батхин А.Б.* Одно вещественное многообразие с краем и его глобальная параметризация // Программирование. 2017. № 2. С. 17–27.
17. *Гантмахер Ф.Р.* Теория матриц. 4-е изд. М.: Наука, 1988. 552 с.
18. *Батхин А.Б.* Параметризация дискриминантного множества вещественного многочлена // Программирование. 2016. Т. 42, № 2. С. 8–21.
19. *Basu S., Pollack R., Roy M.-F.* Algorithms in Real Algebraic Geometry. Algorithms and Computations in Mathematics 10. Berlin Heidelberg New York: Springer-Verlag, 2006. ix p+662.

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 34–39

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 517.5+004.434

НЕКОТОРЫЕ АСИМПТОТИЧЕСКИЕ РАЗЛОЖЕНИЯ РЕШЕНИЙ ВТОРОГО ЧЛЕНА ЧЕТВЕРТОЙ ИЕРАРХИИ УРАВНЕНИЙ ПЕНЛЕВЕ

© 2022 г. В. И. Аношин[a],*, А. Д. Бекетова[a],, А. В. Парусникова[a],***, К. В. Романов[a],******

[a] *Национальный исследовательский университет "Высшая школа экономики", Таллинская ул., д. 34, 123458 Москва, Россия*

**E-mail: vianoshin@edu.hse.ru*

***E-mail: adbeketova@edu.hse.ru*

****E-mail: aparusnikova@hse.ru*

*****E-mail: kvromanov_1@edu.hse.ru*

Поступила в редакцию 02.08.2021 г.
После доработки 19.08.2021 г.
Принята к публикации 15.09.2021 г.

В данной статье строятся асимптотики и асимптотические разложения решений второго члена четвертой иерархии Пенлеве с использованием методов степенной геометрии [1]. Приводятся результаты только для случая общего положения: при значениях параметров уравнения $\beta, \delta \neq 0$. Для построения асимптотических разложений используется код, написанный в пакете символьных вычислений.

DOI: 10.31857/S0132347422010022

1. ВВЕДЕНИЕ

Целью данной работы является исследование асимптотических разложений решений второго члена четвертой иерархии Пенлеве [1], [2]. Функции Пенлеве используются в статистической физике, квантовой теории поля, геометрии минимальных поверхностей, теории чисел и других областях.

2. МЕТОДЫ СТЕПЕННОЙ ГЕОМЕТРИИ

Для построения асимптотических разложений используются методы степенной геометрии [3]. Выпишем часть шагов этих методов.

Рассматривается дифференциальное уравнение, которое имеет вид дифференциальной суммы, т.е. его левая часть является многочленом от независимой переменной, зависимой переменной и ее производных – суммой дифференциальных мономов. Для такого уравнения строится многоугольник Ньютона на координатной плоскости (q_1, q_2) – выпуклая оболочка имеющихся точек. В итоге каждой точке полученного многоугольника Ньютона соответствует один или несколько мономов. Последовательно рассматриваются все обобщенные грани многоугольника (вершины и ребра). Суммируются мономы, которые соответствуют обобщенной грани, получают функцию укороченной суммы $\hat{f}_j^{(d)}(X) = \sum a_i(X)$ по $Q(a_i) \in S_j^{(d)}$.

Для каждого ребра строим внешние нормали, общий вид координат которых $\lambda\omega(1, r)$, где $\lambda > 0$. Если $x \to 0$, то $\omega < 0$, если $x \to \infty$, то $\omega > 0$. Координаты нормалей определяют степень первого члена асимптотического разложения решения уравнения, для которого вычисляется асимптотика $y(x)$: для ребра с внешней нормалью $\omega(1, r)$ общий вид асимптотики решения уравнения имеет вид $y = cx^r$. Для вершин также определяются нормальные конусы: как части плоскости, лежащие между лучами, натянутыми на векторы внешних нормалей к ребрам, примыкающим к вершинам.

3. ПОСТРОЕНИЕ МНОГОУГОЛЬНИКА НЬЮТОНА ДЛЯ ВТОРОГО ЧЛЕНА ЧЕТВЕРТОЙ ИЕРАРХИИ ПЕНЛЕВЕ

Запишем второй член четвертой иерархии Пенлеве [3]. Заметим, что это уравнение сразу представлено в виде дифференциальной суммы:

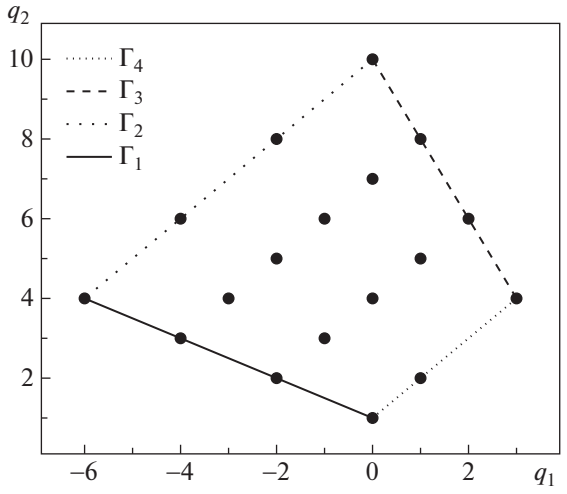


Рис. 1

$$(y_{xx} - 2xy - 2y^3 - \beta)y^2 y_{xxxx} - \frac{1}{2}y^2 y_{xxx}^2 +$$
$$+ (2y^2 + 8y^3 y_x + 4yy_x x - y_x y_{xx} + \beta y_x)yy_{xxx} -$$
$$- \frac{4}{3}yy_{xx}^3 + \left(3xy^2 + 3\beta y - \frac{3}{2}y^4 + \frac{3}{2}y_x^2\right)y_{xx}^2 +$$
$$+ (\beta y^4 - 2y_x y^2 - 12y_x^2 y^3 - 2\beta^2 y + 10xy^5 -$$
$$- 3\beta y_x^2 + 10y^7 - 4xyy_x^2 - 4\beta xy^2)y_{xx} + \qquad (3.1)$$
$$+ 2(\beta - 4y^3)y^2 y_x + \left(4\beta xy + 8xy^4 + \frac{3}{2}\beta^2 + 12\beta y^3\right)y_x^2 -$$
$$- \frac{10}{3}y^{10} - 8xy^8 - 2\beta y^7 - 6x^2 y^6 - 2x\beta y^5 +$$
$$+ \left(\frac{1}{2}\beta^2 - 2 + 9\delta - \frac{4}{3}x^3\right)y^4 + x\beta^2 y^2 + \frac{1}{3}\beta^3 y = 0.$$

Здесь x – независимая, y – зависимая комплексные переменные, β, δ – комплексные параметры. Далее считаем, что оба параметра уравнения ненулевые.

Дифференциальным мономам, участвующим в уравнении (3.1), соответствуют следующие 18 показателей степени: (–4; 3), (–6; 4), (–3; 4), (–4; 6), (–2; 5), (–2; 2), (–1; 6), (–2; 8), (–1; 3), (0; 10), (1; 8), (0; 7), (2; 6), (–1; 5), (0; 4), (3; 4), (1; 2), (0; 1).

Отображаем их на координатную плоскость (q_1, q_2) и строим многоугольник Ньютона как их выпуклую оболочку на рис. 1.

Данный многоугольник можем построить с помощью кода из Листинга 1 (в команду Expand вместо многоточия подставляется левая часть уравнения (3.1)).

```
(*Исходное уравнение *)
equation = Expand [ . . . ] ;
order = 4 ; v a r i a b l e = x ;
function = y [ x ] ;
(*Вычисление векторных степеней мономов *)
d = Coefficient Rules [equation,
    Join [{variable, function},
    Table [D[function, {variable, i}],
    {i, 1, order}]]]/.Rule => List
vectorDegree [degrees_ ] :=
    {degrees [[1]], degrees[[2]]} +
    Total [Table[{=( i = 2), 1}
    degrees [[i]], {i, 3,
    Length [degrees]}]]
points = Sort [DeleteDuplicates [Map[
vectorDegree, (#[[1]]) & /@d]]]
(*Построение многоугольника Ньютона *)
convexHull = ConvexHullMesh [points,
    PlotTheme => "Detailed"];
convexHull Points =
    MeshCoordinates [convexHull];
Show[convexHull, ListPlot [points,
    PlotStyle => {Red,
    PointSize [Large]}]]
```

Листинг 1: Построение многоугольника Ньютона

Каждому ребру и вершине многоугольника Ньютона, изображённого на рис. 1, сопоставим принадлежащие ему точки Q_i и составим укороченные суммы. Это можно сделать с помощью программы символьных вычислений, которая приведена в Листинге 2.

```
(*Точки вершин многоугольника Ньютона *)
gamma0 = (#[[1]]) & /@
    Rationalize [ MeshPrimitives[
Region 'Mesh' MergeCells [convex-
Hull], 0]]
lines = Rationalize [ MeshPrimitives [
Region 'Mesh' MergeCells [convex-
Hull], 1]];
(*Точки сторон многоугольника Ньютона *)
gamma1 = Table [Select [points,
    RegionMember[lines[[i]], #] &], {i,
    1, Length [lines]}]
(*Укороченные уравнения для вершин *)
f0 = Table [FromCoefficientRules [Map[
Rule [#[[1]], #[[2]]]&, Select [d,
vectorDegree [#[[1]]] == gamma0 [[i]]
&]], Join[{variable, function},
Table [D[function, {variable, i}],
{ i, 1, order}]]], {i, 1,
Length [gamma0]}]
```

```
(*Укороченные уравнения для сторон *)
f1 = Table [FromCoefficientRules [Map[
Rule [#[[1]], #[[2]]] &,
    Flatten [Table [Select[d,
    vectorDegree [#[[1]]] ==
    gamma1 [[i]] [[j]] &], {j, 1,
    Length [gamma1 [[i]]]}], 1]],
    Join [{variable, function},
    Table [D[function, {variable, i}],
    {i, 1, order}]]], {i, 1,
    Length [gamma1]}]
```

Листинг 2: Вычисление укороченных уравнений

Теперь последовательно рассмотрим ребра получившегося многоугольника, а затем его вершины и найдём отвечающие им асимптотики и асимптотические разложения.

4. РЕБРО Γ_1

Ребру Γ_1 отвечает внешняя нормаль $N_1 = (-1; -2)$ и укороченное уравнение

$$y_{xx}y^2y_{xxxx} - \beta y^2y_{xxxx} - \frac{1}{2}y^2y_{xxx}^2 - yy_xy_{xx}y_{xxx} + \\ + \beta yy_xy_{xxx} - \frac{4}{3}yy_{xx}^3 + 3\beta yy_{xx}^2 + \frac{3}{2}y_x^2y_{xx}^2 - \\ 2\beta^2yy_{xx} - 3\beta y_x^2y_{xx} + \frac{3}{2}\beta^2y_x^2 + c13\beta^3y = 0. \qquad (4.1)$$

Этому ребру соответствует асимптотика $y = cx^2$ при $x \to 0$. Значения c находим из уравнения

$$\frac{1}{3}\beta^3c + 2\beta^2c^2 - 12\beta c^3 + \frac{40}{3}c^4 = 0,$$

ненулевыми корнями которого являются $c_1 = -0.1\beta$ (некратный) и $c_2 = 0.5\beta$ (кратности два).

Проверим, есть ли критические числа, получим линейный оператор

$$y_{xx}y^2\frac{d^4}{dx^4} - \beta y^2\frac{d^4}{dx^4} - yy_xy_{xx}\frac{d^3}{dx^3} + \beta yy_x\frac{d^3}{dx^3} - \\ - \frac{4}{3}y_{xx}^3 - 4yy_{xx}^2\frac{d^2}{dx^2} + 3\beta y_{xx}^2 + 6\beta yy_{xx}\frac{d^2}{dx^2} + \\ + 3y_x^2y_{xx}\frac{d^2}{dx^2} + 3y_xy_{xx}^2\frac{d}{dx} - 2\beta^2y\frac{d^2}{dx^2} - 2\beta^2y_{xx} - \\ - 6\beta y_xy_{xx}\frac{d}{dx} - -3\beta y_x^2\frac{d^2}{dx^2} + 3\beta^2y_x\frac{d}{dx} + \frac{1}{3}\beta^3. \qquad (4.2)$$

При подстановке $y = -\frac{\beta}{10}x^2$ в выражение (4.2) получим оператор

$$\varkappa = \frac{108\beta^3}{125} - \frac{108\beta^3}{125}x\frac{d}{dx} + \frac{24\beta^3}{125}x^2\frac{d^2}{dx^2} + \\ + \frac{3\beta^3}{125}x^3\frac{d^3}{dx^3} - \frac{3\beta^3}{250}x^4\frac{d^4}{dx^4}.$$

Применим оператор $\varkappa$ к x^k и сократим результат на $x^k\beta^3$, получим характеристический многочлен

$$\nu(k) = \frac{108}{125} - \frac{117}{125}k - \frac{3}{250}k^2 + \frac{12}{125}k^3 - \frac{3}{250}k^4$$

с корнями $k_1 = -3$, $k_2 = 1$, $k_3 = 4$, $k_4 = 6$. Учитывая подходящие значения k, а именно, $k = 4$ и $k = 6$, выпишем множество степеней, участвующих в разложении, продолжающем рассматриваемую асимптотику, и сразу приведем соответствующее семейство степенных разложений решений при $x \to 0$:

$$W_1 = \left\{ y = -\frac{\beta x^2}{10} + c_4x^4 + \frac{\beta}{80}x^5 + c_6x^6 + \sum_{k=7}^{\infty} c_kx^k \right\},$$

где $c_4, c_6 \in \mathbb{C}$ – произвольные константы, остальные коэффициенты c_k, где $k > 6$, однозначно выражаются через них.

При подстановке $y = \frac{\beta}{2}x^2$ получим нулевой оператор, так как соответствующий корень был кратным. Делаем замену $y = \frac{\beta}{2}x^2 + w$ в уравнении (3.1) и проводим вычисления для уравнения, получившегося после замены. Получаем два варианта для следующего члена разложения: $w_{1,2} = \left(\frac{\beta}{20} \pm \frac{3\beta\sqrt{2\delta}}{250}\right)x^5$. Применяя первую вариацию укороченного уравнения, соответствующего ребру многоугольника Ньютона уравнения с зависимой переменной w (вычисленную на решениях $w_{1,2}$), к x^k, получаем характеристический многочлен, корнями которого являются $k_1 = 0$, $k_2 = 1$, $k_3 = 4$, $k_4 = 6$. Нам опять подходят только значения 4 и 6. Итак, продолжая асимптотику $y = \frac{\beta}{2}x^2$ в виде степенных асимптотических разложений, получаем 2 семейства разложений решений при $x \to 0$:

$$W_{2,3} = \left\{ y = \frac{\beta x^2}{2} + c_4x^4 + \right. \\ \left. + \left(\frac{\beta}{20} \pm \frac{3\beta\sqrt{2\delta}}{250}\right)x^5 + c_6x^6 + \sum_{k=7}^{\infty} c_kx^k \right\},$$

где $c_4, c_6 \in \mathbb{C}$ – произвольные константы, c_k, где $k > 6$, однозначно выражаются через них.

5. РЕБРО Γ_2

Данному ребру отвечает внешняя нормаль $N_2 = (-1; 1)$ и укороченное уравнение

$$y_{xx}y^2y_{xxxx} - 2y^5y_{xxxx} - \frac{1}{2}y^2y_{xxx}^2 + 8y^4y_xy_{xxx} - y_xy_{xx}yy_{xxx} - \frac{4}{3}yy_{xx}^3 - \frac{3}{2}y^4y_{xx}^2 + \frac{3}{2}y_x^2y_{xx}^2 - 12y_x^2y^3y_{xx} + 10y^7y_{xx} - \frac{10}{3}y^{10} = 0, \quad (5.1)$$

что дает соответствие асимптотики вида $y = \frac{c}{x}$ при $x \to 0$. Значения c находим из уравнения

$$c^4(c^6 - 6c^4 + 9c^2 - 4) = 0,$$

ненулевые корни которого – это $c_1 = 2$, $c_2 = -2$ (некратные), $c_3 = 1$, $c_4 = -1$ (каждый – кратности два).

Для корней $c = \pm 2$ получаем, что корни k характеристического многочлена таковы, что $k \in \{-3, -2, 3, 4\}$, что влияет на структруру степеней, участвующих в степенном разложении. Для продолжений этих асимптотик при $x \to 0$ имеем такие семейства разложения:

$$W_4 = \left\{ y = \frac{2}{x} + \left(\frac{\beta - 2}{20}\right)x^2 + c_3c^3 + c_4x^4 + \sum_{k=5}^{\infty} c_kx^k \right\};$$

$$W_5 = \left\{ y = -\frac{2}{x} + \left(\frac{\beta + 2}{20}\right)x^2 + c_3c^3 + c_4x^4 + \sum_{k=5}^{\infty} c_kx^k \right\},$$

здесь $c_3, c_4 \in \mathbb{C}$ – произвольные константы, остальные коэффициенты c_k, $k > 4$ однозначно определяются через них.

Для исследования разложений, соответствующих кратным корням $c = \pm 1$ делаем в уравнении (5.1) замену $y = \pm\frac{1}{x} + w$, проводя процесс поиска следующего члена асимптотики для нового уравнения и его многоугольника Ньютона. Так, следующее слагаемое асимптотики имеет вид $w_{6,7} = \pm\frac{\sqrt{9\delta - 8 - 8\beta - 2\beta^2}}{4\sqrt{2}}x^2$ для $y = \frac{1}{x}$ и $w_{8,9} = \pm\frac{\sqrt{-9\delta + 8 - 8\beta + 2\beta^2}}{4\sqrt{2}}x^2$ для $y = -\frac{1}{x}$ (здесь нумерацию приводим в соответствии с нумерацией соответствующих разложений). Далее находим характеристический многочлен. Его корни – это $k_1 = -2$, $k_2 = 0$, $k_3 = 3$, $k_4 = 4$. Критическими числами являются k_3 и k_4, так как для них $\operatorname{Re} k_i > 2$. Имеем следующие четыре семейства асимптотических разложений решений при $x \to 0$:

$$W_{6,7} = \left\{ y = \frac{1}{x} \pm \frac{\sqrt{9\delta - 8 - 8\beta - 2\beta^2}}{4\sqrt{2}}x^2 + c_3x^3 + c_4x^4 + \sum_{k=5}^{\infty} c_kx^k \right\};$$

$$W_{8,9} = \left\{ y = -\frac{1}{x} \pm \frac{\sqrt{-9\delta + 8 - 8\beta + 2\beta^2}}{4\sqrt{2}}x^2 + c_3x^3 + c_4x^4 + \sum_{k=5}^{\infty} c_kx^k \right\},$$

вновь $c_3, c_4 \in \mathbb{C}$ – произвольные константы, остальные постоянные c_k, $k > 4$ однозначно определяются через них.

6. РЕБРО Γ_3

Ребру Γ_3 соответствует внешняя нормаль $N_3 = (1; 1/2)$ и укороченное уравнение

$$-\frac{10}{3}y^{10} - 8xy^8 - 6x^2y^6 - \frac{4}{3}x^3y^4 = 0. \quad (6.1)$$

Сразу отметим, что укороченное уравнение (6.1) алгебраическое, критических чисел нет. Исходя из вида внешней нормали, ищем степенную асимптотику соответствующего решения в виде $y = cx^{1/2}$, подставляя это выражение в уравнение (6.1). Получаем соотношение

$$-\frac{10}{3}c^{10}x^5 - 8xc^8x^4 - 6x^2c^6x^3 - \frac{4}{3}x^3c^4x^2 = 0,$$

откуда выражаем ненулевые значения c: $c_{1,2} = \pm i$ кратности 2, $c_{3,4} = \pm i\sqrt{\frac{2}{5}}$ – некратные корни. Продолжая асимптотики вида $y = c_kx^{1/2}$ в виде степенных асимптотических разложений, получаем 6 разложений решений при $x \to \infty$:

$$W_{10,11} = \left\{ y = i\sqrt{x} + \frac{\beta \pm \sqrt{\beta - (\beta^2 - 18\delta)}}{4x} + \sum_{k=1}^{\infty} c_kx^{-1-\frac{3}{2}k} \right\};$$

$$W_{12,13} = \left\{ y = -i\sqrt{x} + \frac{\beta \pm \sqrt{\beta + (\beta^2 - 18\delta)}}{4x} + \sum_{k=1}^{\infty} c_kx^{-1-\frac{3}{2}k} \right\};$$

$$W_{14,15} = \left\{ y = \pm\sqrt{\frac{2}{5}}i\sqrt{x} - \frac{\beta}{2x} + \sum_{k=1}^{\infty} c_kx^{-1-\frac{3}{2}k} \right\}.$$

7. РЕБРО Γ_4

Укороченное уравнение для ребра Γ_4 имеет вид

$$-\frac{4}{3}x^3y^4 + x\beta^2y^2 + \frac{1}{3}\beta^3y = 0,$$

а внешняя нормаль – это $N_4 = (1; -1)$, ребру опять соответствуют асимптотики решений при $x \to \infty$, это асимптотики вида $y = cx^{-1}$. Укороченное уравнение вновь является алгебраическим, не дает критических чисел. Для нахождения значений c имеем уравнение

$$-\frac{4}{3}c^4 + \beta^2 c^2 + \frac{1}{3}\beta^3 c = 0,$$

откуда выражаем ненулевые значения c: $c_1 = \beta$ кратности 1, $c_2 = -\dfrac{\beta}{2}$ – кратности 2. Продолжаем асимптотики вида $y = c_k x^{1/2}$ в виде степенных асимптотических разложений, имеем ещё 3 разложения решений при $x \to \infty$:

$$W_{16} = \left\{ y = \frac{\beta}{x} + \frac{6\beta\delta - 7\beta - 5\beta^3}{2x^4} + \sum_{k=2}^{\infty} c_k x^{-1-3k} \right\};$$

$$W_{17,18} = \left\{ y = -\frac{\beta}{2x} \pm \frac{3\beta\sqrt{\delta}}{4x^4} + \sum_{k=2}^{\infty} c_k x^{-1-3k} \right\}.$$

8. АСИМПТОТИЧЕСКИЕ РАЗЛОЖЕНИЯ РЕШЕНИЙ ДЛЯ ВЕРШИН

Рассмотрим укороченные уравнения, соответствующие вершинам.

Вершинам (0, 1), (3, 4) и (0, 10) соответствуют алгебраические мономы $1/3\beta^3 y$, $-4/3x^3y^4$, $-10/3y^{10}$, они не дают асимптотик решений уравнений, согласно [1]. Вершине (–6, 4) соответствует нормальный конус $\{(p_1, p_2) : (p_1, p_2) = \lambda_1(-1; -2) + \lambda_2(-1; 1), \lambda_1, \lambda_2 \in \mathbb{R}, \lambda_1 > 0, \lambda_2 > 0\}$, что дает ограничения на показатель степени: $\{r : -1 < \mathrm{Re}\, r < 2, r \neq -1, r \neq 2\}$. Также данной вершине отвечает укороченное уравнение

$$y^2 y_{xx} y_{xxxx} - \frac{y^2 y_{xxx}^2}{2} - \frac{4}{3} y y_{xx}^3 + \frac{3}{2} y_x^2 y_{xx}^2 - y y_x y_{xx} y_{xxx} = 0.$$

Подставим в него $y = cx^r$, приравняем полученное выражение к нулю и решим уравнение относительно r; получим возможные значения r: $r = -3$, $r = 0$, $r = 1$, $r = 4$.

Только $r = 0$, $r = 1$ лежат в нормальном конусе, соответствующем данной вершине, откуда легко вычисляются дополнительные однопараметрические семейства асимптотик решений второго члена четвертой иерархии Пенлеве при $x \to 0$, а именно, имеются асимптотики вида $y = c$ и $y = cx$, где $c \in \mathbb{C}\backslash\{0\}$ – произвольная постоянная. В данной работе продолжим только вторую асимптотику как асимптотическое разложение.

Стандартный метод вычисления структуры разложения сразу не приводит нас к результату:

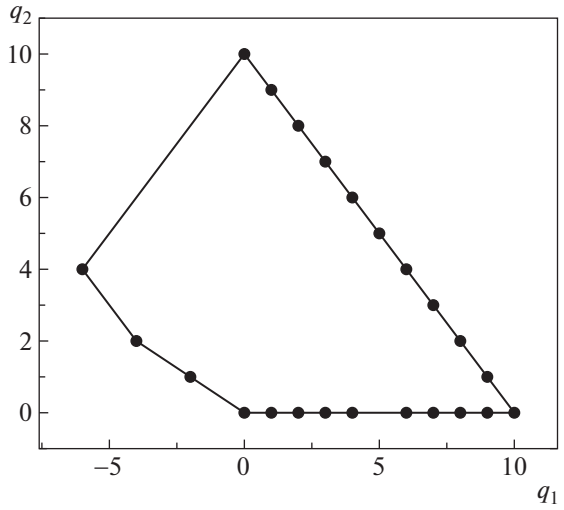


Рис. 2

линеаризация укороченного уравнения, соответствующего вершине (–6, 4), на решении $y = cx$, дает нулевой оператор. Поэтому мы продолжаем вычисления, делая сдвиг $y = cx + w$. Для нового уравнения получаем следующий многоугольник Ньютона (рис. 2).

Нас интересуют только те его вершины и ребра, нормальные конусы к которым могут дать асимптотики, соответствующие конусу задачи $\mathcal{K} = \{k : \mathrm{Re}\, k \geq 1, k \neq 1\}$. Этому условию соответствуют лишь вершины (0, 0) и (–4, 2) и соединяющее их ребро. Последовательно рассмотим их.

Вершине (0, 0) соответствует алгебраическое укороченное уравнение, которое не дает асимптотик.

Вершине (–4, 2) отвечает уравнение

$$\frac{3}{2}c^2 w_{xx}^2 - \frac{1}{2}c^2 x^2 w_{xxx}^2 + c^2 x^2 w_{xx} w_{xxxx} - c^2 x w_{xx} w_{xxx} = 0,$$

подставим в него $w = c_1 x^{r_1}$, видим, что $r_1 = 3$ или $r_1 = 5$. Ни одно из значений показателей степени не подходит, поскольку вектор $(-1, -r_1)$ не принадлежит нормальному конусу вершины.

Ребру, соединяющему указанные вершины, ставится в соответствие укороченное уравнение

$$\frac{3}{2}c^2 w_{xx}^2 - \frac{1}{2}c^2 x^2 w_{xxx}^2 + c^2 x^2 w_{xx} w_{xxxx} - c^2 xy w_{xx} w_{xxx} -$$
$$- \beta c^2 x^2 w_{xxxx} + \beta c^2 x w_{xxx} - 3\beta c^2 w_{xx} + \frac{3}{2}\beta^2 c^2 = 0,$$

а также внешняя нормаль (–1, –2), поэтому решение данного уравнения ищем в виде $w = Ax^2$. После подстановки получили:

$$6A^2 - 6\beta A + \frac{3}{2}\beta^2 = 0,$$

откуда $A = \frac{\beta}{2}$ – корень кратности 2. И поэтому мы продолжаем вычисления, делая сдвиг $y = cx + \frac{\beta}{2}x^2 + y_2$, добиваясь ненулевых значений линеаризации укороченного уравнения на решении. После подстановки получаем уравнение с многоугольником Ньютона, у которого подходящими являются вершина $(-4, 2)$, дающая следующий член асимптотики вида c_3x^3 и критическое число 4, и ребро, соединяющее вершины $(-4, 2)$ и $(0, 4)$ данного многоугольника (критическое число 5).

В итоге получаем следующие семейства степенных асимптотических разложений, продолжающих асимптотику решения $y = cx$ при $x \to 0$:

$$W_{19} = \left\{ y = cx + \frac{\beta}{2}x^2 + c_3x^3 + c_4x^4 + \sum_{k=5}^{\infty} a_k x^k \right\};$$

$$W_{20,21} = \left\{ y = cx + \frac{\beta}{2}x^2 + \frac{c(2 \pm 3\sqrt{2\delta})}{12}x^4 + c_5x^5 + \sum_{k=6}^{\infty} b_k x^k \right\},$$

где c, $c_3 \in \mathbb{C}\backslash\{0\}$, c_4, $c_5 \in \mathbb{C}$ – произвольные постоянные, коэффициенты a_k, b_k однозначно через них выражаются.

9. БЛАГОДАРНОСТИ

Работа выполнена при поддержке гранта Российского научного фонда 19-71-10003.

СПИСОК ЛИТЕРАТУРЫ

1. *Пикеринг А.* Иерархии Пенлеве и тест Пенлеве // УМН. 2003. Т. 137. С. 445–456.
2. *Кудряшов Н.А.* О четвертой иерархии Пенлеве // Теор. и матем. физика. 2003. Т. 134. С. 101–109.
3. *Брюно А.Д.* Асимптотики и разложения решений обыкновенного дифференциального уравнения // УМН. 2004. Т. 59. № 3. С. 31–80.

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 40–53

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 512.548.7

АЛГОРИТМЫ ПРОВЕРКИ НЕКОТОРЫХ СВОЙСТВ *N*-КВАЗИГРУПП

© 2022 г. А. В. Галатенко[a],*, А. Е. Панкратьев[a],, В. М. Староверов[a],*****

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

**E-mail: agalat@msu.ru*

***E-mail: apankrat@intsys.msu.ru*

****E-mail: staroverovvl@imscs.msu.ru*

Поступила в редакцию 23.07.2021 г.
После доработки 05.08.2021 г.
Принята к публикации 15.09.2021 г.

В работе описываются эффективные алгоритмы для проверки некоторых существенных с криптографической точки зрения свойств n-квазигрупп: полиномиальной полноты (которая сводится к проверке простоты и неаффинности) и существования n-подквазигрупп. Доказываются теоремы об оценках времени работы предложенных алгоритмов и их пространственной сложности, а также приводятся результаты численных экспериментов для оценки практической эффективности программной реализации.

DOI: 10.31857/S0132347422010046

1. ВВЕДЕНИЕ

Конечные квазигруппы являются популярной платформой для реализации различных криптографических примитивов. Шеннон доказал, что табличное гаммирование по латинскому квадрату (т.е. по таблице Кэли конечной квазигруппы) является совершенным шифром [1]. Широкий спектр различных криптографических примитивов на основе квазигрупп представлен в обзорных работах [2, 3]. В последних конкурсах NIST по выбору новых криптографических стандартов регулярно участвуют квазигрупповые алгоритмы-кандидаты (в качестве примера приведем хэш-функцию EDON-R' [4]). Растет интерес и к структурам более высокой размерности – латинским кубам (таблицам Кэли 3-квазигрупп) и латинским гиперкубам (таблицам Кэли n-квази групп при $n \geq 4$). Возникают новые алгоритмы (см., например, [5, 6]), ведутся исследования криптографически важных свойств [7–9].

Наша работа посвящена задаче алгоритмической проверки двух важных свойств n-квазигрупп: полиномиальной полноты и наличия собственных n-подквазигрупп. Важность полиномиальной полноты в криптографических приложениях была отмечена В.А. Артамоновым [10]; она обусловлена тем, что распознавание разрешимости уравнений над полиномиально полными алгебрами является NP-полной задачей [11], что обеспечивает защиту от атак методом решения уравнений на биты ключа. Наличие собственных n-подквазигрупп может привести к вырождению операции на n-подквази-группу, что облегчит, например, атаку методом “грубой силы”. При этом в ряде случаев n-подквази группы порядка 1 считаются допустимыми. Задача построения и обоснования эффективных алгоритмов для распознавания упомянутых свойств относится к числу классических проблем компьютерной алгебры.

В случае $n = 2$ для проверки обоих свойств предложены достаточно эффективные алгоритмы (время работы которых является кубическим от размера квазигруппы k для проверки полиномиальной полноты [12–14], имеет порядок $k^{7/3} \cdot (\log k)^{2/3}$ для проверки наличия собственных подквазигрупп и порядок $k^3 \log k$ для проверки наличия собственных подквазигрупп из не менее чем двух элементов [15]). В нашей работе приводится обобщение алгоритмов [14, 15] на случай n-квазигрупп для $n \geq 3$. Проверка полиномиальной полноты сводится к проверке одновременной простоты (сложность $O(k^{n+1})$) и неаффинности (сложность $O(k^n)$ при $n \geq 3$). Результаты были анонсированы в работе [8] и частично представлены на семинаре “Компьютерная алгебра” [16]. Алгоритмы проверки существования собственных n-подквази групп и собственных n-подквазигрупп порядка ≥ 2 были анонсированы в работе [9]; их сложность со-

ставляет $O\left(k^{\frac{n^2+n+1}{n+1}}\log^{n+1}k\right)$ и $O\left(k^{\frac{n^2+2n+4}{n+2}}\log^{n+2}k\right)$ соответственно. Заметим, что при $n = 2$ полученные оценки уточняют результат работы [15] для случая подквазигрупп порядка ≥ 2. Помимо доказательства корректности и оценки сложности мы приводим результаты практических экспериментов, полученные при применении распараллеленных реализаций алгоритмов к семейству тестовых примеров.

Дальнейшее изложение имеет следующую структуру. В разделе 2 приводятся необходимые определения. Раздел 3 посвящен проверке простоты, раздел 4 – проверке неаффинности. В разделе 5 суммируются результаты для задачи определения полиномиальной полноты. В разделе 6 изучаются задачи проверки существования собственных n-подквазигрупп и собственных n-подквазигрупп порядка ≥ 2. Наконец, раздел 6 является заключением.

При работе над статьей авторы имели счастливую возможность обсуждать материал с В.А. Артамоновым.

Работа выполнена при финансовой поддержке DRDO (Индия), проект “Quasigroup Based Cryptography: Security Analysis and Development of Crypto-Primitives and Algorithms (QGSEC)”, номер гранта SAG/4600/TCID/Prog/QGSEC.

2. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Определение 1. *Конечной n-квазигруппой называется пара (Q, f), где $Q = \{q_1, \ldots, q_k\}$ – конечное множество, а $f: Q^n \to Q$ такова, что при произвольной фиксации любых $n-1$ переменных результат является биекцией, то есть перестановкой на множестве Q. При этом операция f называется n-квазигрупповой.*

В дальнейшем все структуры будут конечными, поэтому для краткости слово “конечный” будет опускаться.

Таблица Кэли n-квазигрупповой операции является n-мерным латинским гиперкубом, то есть n-мерной матрицей размера $|Q| \times \ldots \times |Q|$, такой что каждая одномерная “строка” задает перестановку на множестве Q. Обратно, произвольный n-мерный латинский гиперкуб задает n-квазигрупповую операцию с соответствующим универсумом.

Без ограничения общности в дальнейшем мы будем считать, что $Q = E_k = \{0, 1, \ldots, k-1\}$, а n-квазигрупповая операция является n-арной функцией k-значной логики. Пусть P_k – множество всех функций k-значной логики, P_k^0 – множество всех констант. Пусть $F \subseteq P_k$. Через $[F]$ обозначим замыкание множества F относительно операции суперпозиции (см., например, [17]).

Определение 2. *n-квазигруппа (Q, f) называется полиномиально полной, если*

$$[\{f\} \cup P_k^0] = P_k.$$

При этом "функция (операция)” f также называется полиномиально полной.

Другими словами, полиномиальная полнота означает, что произвольная функция k-значной логики может быть получена суперпозициями из функции f и констант. Для описания характеристического свойства полиномиально полных n-квазигрупп нам потребуется ввести два вспомогательных понятия.

Пусть ~ является нетривиальным отношением эквивалентности на E_k, $g \in P_k$ – функция арности m.

Определение 3. *Функция g сохраняет отношение эквивалентности ~, если для любой пары входных наборов $a = (a_1, \ldots, a_m)$ и $b = (b_1, \ldots, b_m)$, таких что $a_i \sim b_i$, $i = 1, \ldots, m$, выполнено отношение $g(a) \sim g(b)$.*

Определение очевидным образом обобщается на случай множества функций.

Определение 4. *Множество функций $G \subseteq P_k$ сохраняет отношение эквивалентности ~, если каждая функция $g \in G$ сохраняет ~.*

Легко увидеть, что n-квазигрупповые операции могут сохранять только отношения эквивалентности, все классы которых являются равномощными.

Определение 5. *n-квазигруппа (Q, f) называется простой, если не существует нетривиального отношения эквивалентности на E_k, сохраняемого функцией f.*

Определение 6. *n-квазигруппа (Q, f) называется аффинной, если существует абелева группа $(Q, +)$, автоморфизмы $\alpha_1, \ldots, \alpha_n$ группы (Q, f) и константа $c \in Q$, такие что выполнено тождество*

$$f(x_1, \ldots, x_n) \equiv \alpha_1(x_1) + \ldots + \alpha_n(x_n) + c. \qquad (2.1)$$

В противном случае n-квазигруппа (Q, f) называется неаффинной.

Известно (см., например, [18]), что n-квази группа является полиномиально полной тогда и только тогда, когда она проста и неаффинна.

Пусть $Q' \subseteq Q$. Через $f(Q')$ обозначим замыкание множества Q' относительно операции f, то есть множество всех констант, выразимых через f и элементы Q'. Очевидно, что замыкание обладает свойством монотонности: из вложения $Q' \subseteq Q''$ следует вложение $f(Q') \subseteq f(Q'')$.

Определение 7. *Множество* $Q' \subseteq Q$ *называется полным, если* $f(Q') = Q$.

В силу монотонности из полноты Q' следует полнота всех его надмножеств.

Определение 8. *Пусть* $Q' \subset Q$, $1 \le |Q'| < |Q|$. *Если* $f(Q') = Q'$, *то говорим*, что *n-квазигруппа* (Q, f) *содержит собственную n-подквазигруппу* (Q', f'), *где* f' *– ограничение операции* f *на* $(Q')^n$.

Заметим, что очевидным образом для любого $Q' \subseteq Q$ выполнено равенство $f(f(Q')) = f(Q')$, поэтому если $Q' \neq \emptyset$ и $f(Q') \neq Q$, то $f(Q')$ является собственной n-подквазигруппой.

В дальнейшем для краткости мы будем отождествлять n-подквазигруппу (Q', f') с множеством Q'.

Определение 9. *Пусть* Q' *– собственная n-подквазигруппа n-квазигруппы* (Q, f). *Если дополнительно выполнено условие* $|Q'| > 1$, *то подквазигруппа называется нетривиальной*.

Для алгоритма проверки существования собственных n-подквазигрупп нам потребуется понятие системы представителей.

Определение 10. *Пусть* M, $|M| < \infty$ *– некоторое множество*, $t \in \mathbb{N}$, $M_1, \ldots, M_t$ *– подмножества* M. *Множество* $M_0 \subseteq M$ *называется системой представителей для* $M_1, \ldots, M_t$, *если для всех* i *от* 1 *до* t *найдется элемент* $m_i \in M_0$, *такой что* $m_i \in M_i$.

Элемент $m_i \in M_0$, такой что $m_i \in M_i$, называется представителем множества M_i. Заметим, что представители для различных i могут совпадать.

В дальнейшем мы будем считать, что n-квазигруппы заданы таблицами Кэли. Случай функционального задания является предметом будущих исследований. Элементарными операциями считаются вычисление n-квазигрупповой операции (в нашем случае это просто чтение из памяти), а также арифметические операции в $\mathbb{Z}_k$ (если элементы квазигруппы не помещаются в базовые типы данных, то таблица Кэли не помещается в оперативную память, поэтому допущение является разумным). Основание всех логарифмов в дальнейшем равно 2.

3. АЛГОРИТМ ПРОВЕРКИ ПРОСТОТЫ

Проверка простоты n-квазигруппы (Q, f) может быть сведена к поиску нетривиального отношения эквивалентности, сохраняемого множеством всех одноместных функций, порожденных f. Корректность перехода обосновывается следующим утверждением, являющимся аналогом леммы 1 из работы [13]. Обозначим через F_1 множество всех одноместных функций, полученных произвольной фиксацией произвольных $(n-1)$ переменных функции f. Мощность множества F_1 очевидным образом равна $n \cdot k^{n-1}$.

Лемма 1. *Пусть* (Q, f) *– n-квазигруппа. Отношение эквивалентности* $\sim$ *сохраняется функцией f тогда и только тогда, когда* $\sim$ *сохраняется множеством* F_1.

Доказательство. Сперва докажем необходимость. Пусть $g(x_i)$ – произвольная функция из F_1, полученная из f фиксацией

$$x_1 = a_1, \ldots, x_{i-1} = a_{i-1}, \quad x_{i+1} = a_{i+1}, \ldots, x_n = a_n.$$

Рассмотрим произвольную пару эквивалентных элементов a', a''. Заметим, что $g(a') = f(a_1, \ldots, a_{i-1}, a', a_{i+1}, \ldots, a_n)$, $g(a'') = f(a_1, \ldots, a_{i-1}, a'', a_{i+1}, \ldots, a_n)$. Так как наборы в правых частях являются покомпонентно эквивалентными, а функция f сохраняет отношение эквивалентности, имеем $g(a') \sim g(a'')$. В силу произвольности функции g и пары a', a'' необходимость доказана.

Докажем достаточность. Предположим, что отношение $\sim$ не сохраняется функцией f. По определению существует пара покомпонентно эквивалентных наборов $a = (a_1, \ldots, a_n)$ и $b = (b_1, \ldots, b_n)$, таких что $f(a) \not\sim f(b)$. Рассмотрим наборы $c^0 = a$, $c^1 = (b_1, a_2, \ldots, a_n)$, ..., $c^{n-1} = (b_1, \ldots, b_{n-1}, a_n)$, $c^n = (b_1, \ldots, b_n)$. Заметим, что для любого i от 1 до n наборы c^{i-1} и c^i отличаются ровно в одной компоненте. Кроме того, по предположению $f(c^0) \not\sim f(c^n)$. Следовательно, найдется значение индекса i, для которого $f(c^{i-1}) \not\sim f(c^i)$. Рассмотрим функцию $g(x_i)$, полученную из f фиксацией $x_1 = b_1$, ..., $x_{i-1} = b_{i-1}, x_{i+1} = a_i, \ldots, x_n = a_n$. Заметим, что $a_i \sim b_i$, но $g(a_i) = f(c^{i-1}) \not\sim f(c^i) = g(b_i)$, то есть множество F_1 не сохраняет отношение $\sim$. Достаточность доказана. □

Рассмотрим следующую процедуру Check(i), принимающую на вход таблицу Кэли n-квази группы и параметр i, $1 \le i \le k-1$, и вычисляющую транзитивное замыкание отношения $0 \sim i$ под действием множества F_1.

```
1. пометить пару {0, i} как эквивалентную
2. инициализировать очередь пар на проверку
3. добавить пару {0, i} в очередь
4. пока очередь непуста
5.   извлечь первую пару из очереди
          (пусть это пара {s, t})
6.   для каждой функции g из F1
вычислить g(s) и g(t)
7.   если пара {g(s), g(t)} не помечена
          как эквивалентная
```

```
8.   пометить пару как эквивалентную
9.   добавить пару в очередь
10.  объединить классы эквивалентности,
       соответствующие g(s) и g(t),
       и пометить новые
       эквивалентные пары
11.    добавить новые эквивалентные
       пары в очередь
12.   конец если
13.  конец цикла
14. конец цикла
15. если все элементы эквивалентны
16.  вернуть "отношение не найдено"
17. иначе
18.  вернуть "отношение найдено"
```

Следующее утверждение обобщает лемму 2 из работы [13].

Лемма 2. *Процедура Check*(i) *возвращает "отношение найдено" тогда и только тогда, когда существует нетривиальное отношение эквивалентности* $\sim$, *сохраняемое* F_1 *и такое, что* $0 \sim i$.

Доказательство. Предположим, что процедура вернула "отношение найдено". Значит, на выходе возникло разбиение множества E_k на непересекающиеся классы, то есть отношение эквивалентности. Обозначим это отношение через $\sim$. Заметим, что по построению для каждой функции $g \in F_1$ и для каждой пары элементов $a, b \in E_k$, лежащих в одном классе эквивалентности, в процессе выполнения процедуры установлено отношение $g(a) \sim g(b)$. Значит, отношение $\sim$ сохраняется множеством F_1.

Обратно, предположим, что процедура вернула "отношение не найдено", однако множество F_1 сохраняет некоторое отношение эквивалентности $\sim$, для которого $0 \sim i$. Рассмотрим пару $\{a, b\}$, такую что $a \not\sim b$, и для любой пары $\{a', b'\}$, эквивалентность которой была установлена процедурой до эквивалентности a и b, выполнено $a' \sim b'$. Такая пара очевидным образом существует по предположению. Возможны следующие случаи.

1. Эквивалентность a и b была установлена процедурой на строке 7, то есть для некоторых $g \in F_1$ и $a', b' \in E_k$, $a' \sim b'$, выполнено $g(a') = a$, $g(b') = b$. Так как $a \not\sim b$, получаем противоречие с предположением.

2. Эквивалентность a и b установлена на строке 10. В этом случае существуют $a', b' \in E_k$, $a' \sim b'$, такие что $a' \sim a$, $b' \sim b$, что противоречит транзитивности отношения эквивалентности $\sim$. □

Применим к процедуре Check(i) два оптимизирующих преобразования, аналогичные оптимизациям из [14]: не будем добавлять в очередь на проверку пары, возникающие при слиянии классов, а также прекратим работу, если размер хотя бы одного класса станет строго больше, чем $k/2$. Получим следующую процедуру OptimizedCheck(i), отличающуюся от Check(i) только в строке 11.

```
1. пометить пару {0, i} как эквива-
лентную
2. инициализировать очередь пар на
проверку
3. добавить пару {0, i} в очередь
4. пока очередь непуста
5.  извлечь первую пару из очереди
(пусть это пара {s, t})
6.  для каждой функции g из F1
вычислить g(s) и g(t)
7.  если пара {g(s), g(t)} не помечена
как эквивалентная
8.    пометить пару как эквивалентную
9.    добавить пару в очередь
10.   объединить классы эквивалентности,
          соответствующие g(s) и g(t)
          и пометить новые
          эквивалентные пары
11.   если размер класса больше k/2
вернуть "отношение не найдено"
12.    конец если
13.   конец цикла
14. конец цикла
15.если все элементы эквивалентны
16.  вернуть "отношение не найдено"
17. иначе
18.  вернуть "отношение найдено"
```

Лемма 3. *Процедура OptimizedCheck*(i) *возвращает "отношение не найдено" тогда и только тогда, когда процедура Check*(i) *возвращает "отношение не найдено".*

Доказательство. Достаточно установить справедливость следующих фактов: не существует нетривиальных отношений эквивалентности, сохраняемых множеством F_1, у которых найдется класс, размер которого превышает $k/2$; если пара была добавлена на проверку в процедуре Check(i) на строке 11, то при рассмотрении этой пары новых эквивалентностей не будет найдено. Первый факт следует из того, что n-квазигрупповые операции могут сохранять только отношения эквивалентности, для которых все классы эквивалентности равномощны (то есть отношение нетривиально, только если размер всех классов $\le k/2$). Для доказательства второго факта рассмотрим произвольную пару $\{a, b\}$, добавленную в процедуре Check(i) на строке 11. Заметим, что перед парой $\{a, b\}$ на строке 9 в очередь была помещена пара $\{a', b'\}$, такая что $a' \sim a$, $b' \sim b$. Значит, к момен-

ту начала обработки пары $\{a, b\}$ будут обработаны пары $\{a', a\}$, $\{b', b\}$ и $\{a', b'\}$. Рассмотрим произвольную функцию $g \in F_1$. Легко увидеть, что $g(a) \sim g(a') \sim g(b') \sim g(b)$. В силу транзитивности эквивалентность $g(a) \sim g(b)$ уже установлена.

Лемма 4. *Сложность процедуры Optimized-Check(i) составляет $O(k^n)$ для n-квазигрупп порядка k при фиксированном $n \geq 3$ и $k \to \infty$.* □

Доказательство. Сперва оценим сложность, связанную с объединением классов эквивалентности. Заметим, что в первый момент классов $k-1$, а в конце процедуры классов не меньше одного. Таким образом, процедура слияния происходит не более $k-2$ раз. Отсюда, в частности, следует, что число пар, попавших в очередь, не превосходит $k-1$. Слияние классов легко реализовать со сложностью, линейной по k (например, с помощью массива, сопоставляющего каждому элементу E_k номер класса; для слияния достаточно изменить метки меньшего по мощности из объединяемых классов). Кроме того, общее число эквивалентных пар не более, чем квадратично зависит от k. Таким образом, суммарная сложность шага 10 составляет $O(k^2)$, то есть $o(k^n)$ (с учетом неравенства $n \geq 3$).

Оставшиеся операции являются элементарными и оцениваются числом итераций цикла. Внешний цикл, как показано выше, выполнится не более $k-1$ раза. Внутренний цикл будет выполнен столько раз, сколько функций содержится в множестве F_1, то есть $n \cdot k^{n-1} = O(k^{n-1})$ при фиксированном n, и итоговая сложность составит $O(k^n)$. □

Рассмотрим следующий алгоритм проверки простоты.

```
1. цикл по i от 1 до k - 1
2.   выполнить OptimizedCheck(i)
3.   если результат "отношение найдено"
4.     вернуть "непростая"
5. конец цикла
6. вернуть "простая"
```

Теорема 1. *Предложенный алгоритм возвращает "простая" тогда и только тогда, когда n-квазигруппа является простой. Сложность алгоритма составляет $O(k^{n+1})$ при фиксированном n и порядке n-квазигруппы k, стремящемся к бесконечности.*

Доказательство. Корректность алгоритма следует из лемм 1, 2 и 3 с учетом равномощности классов эквивалентности (из равномощности вытекает, что в любом нетривиальном отношении эквивалентности $\sim$, сохраняемом n-квазигрупповой операцией, есть пара $0 \sim i$ для некоторого $i \in \{1, \ldots, k-1\}$); оценка сложности непосредственно вытекает из леммы 4. □

Таблица 1. Время проверки простоты 3-квазигрупп

k	1	2	4	8
32	0	0	0	0
64	0	0	0	0
128	2	1	1	1
256	75	47	30	17
512	1332	733	456	426

3.1. Оценка практической эффективности

Алгоритм проверки простоты для 3-квази групп был программно реализован с поддержкой OpenMP-распараллеливания. Программа была протестирована на рабочей станции с 8-ядерным процессором i7–3770 CPU @3.40GHz и 32 гигабайтами памяти на 3-квазигруппах, аналогичных тестовым примерам из [14]. Здесь и далее для контроля корректности написанного кода использовались инструменты *AddressSanitizer*, *LeakSanitizer*, *ThreadSanitizer* (см., например, [19]) и сравнение оптимизированной реализации с референсной. Время работы представлено в табл. 1. В левом столбце приведены порядки 3-квазигрупп, в столбцах со второго по пятый содержатся максимальные времена работы в секундах на соответствующем числе ядер.

Легко увидеть, что результаты численного эксперимента по проверке простоты согласуются с теоретической асимптотикой времени работы алгоритма в зависимости от порядка 3-квазигрупп. Полученные результаты подтверждают возможность проверки простоты 3-квазигрупп за приемлемое время до порядка 512 включительно.

4. АЛГОРИТМ ПРОВЕРКИ НЕАФФИННОСТИ

Алгоритм проверки неаффинности n-квази групп при $n \geq 3$ является естественным обобщением алгоритма проверки неаффинности квази групп, предложенного в работе [13]. Следующее утверждение является переносом леммы 3 из [13] на случай n-квазигрупп.

Лемма 5. *Пусть (Q, f) – n-квазигруппа, аффинная над абелевой группой $(Q,+)$. Тогда для любого $d \in Q$ существует абелева группа $(Q,+')$, для которой элемент d является нейтральным, и (Q, f) аффинна над $(Q,+')$.*

Доказательство. Определим операцию $+'$ по правилу $x +' y = x + y - d$. Пусть e – нейтральный элемент группы $(Q,+)$. Рассмотрим отображение $\varphi : Q \to Q$, заданное соотношением $\varphi(x) = x + d$. Заметим, что $\varphi(x + y) = x + y + d = (x + d) + (y + d) - d = \varphi(x) +' \varphi(y)$. Таким образом, $(Q,+')$ – абелева группа, изоморфная $(Q,+)$ и имеющая

нейтральный элемент $\varphi(e) = e + d = d$. По определению аффинной n-квазигруппы справедливо тождество (2.1):

$$f(x_1, \ldots, x_n) = \alpha_1(x_1) + \ldots + \alpha_n(x_n) + c$$

для некоторых автоморфизмов $\alpha_1, \ldots, \alpha_n$ группы $(Q,+)$ и $c \in Q$. Заметим, что $\alpha_i' = \varphi \circ \alpha_i \circ \varphi^{-1}$ является автоморфизмом группы $(Q,+')$ для $i = 1, \ldots, n$. Действительно, для любых $x, y \in Q$ выполнены равенства

$$\begin{gathered}\alpha_i'(x +' y) = \varphi \circ \alpha_i \circ \varphi^{-1}(x + y - d) = \\ = \varphi \circ \alpha_i((x - d) + (y - d)) = \\ = \varphi(\alpha_i(x - d) + \alpha_i(y - d)) = \\ = \varphi(\alpha_i \circ \varphi^{-1}(x) + \alpha_i \circ \varphi^{-1}(y)) = \\ = \varphi \circ \alpha_i \circ \varphi^{-1}(x) +' \varphi \circ \alpha_i \circ \varphi^{-1}(y) = \alpha_i'(x) +' \alpha_i'(y).\end{gathered}$$

Наконец, преобразуем тождество (2.1) следующим образом:

$$\begin{gathered}f(x_1, \ldots, x_n) = \alpha_1(x_1) + \ldots + \alpha_n(x_n) + c = \\ = \varphi \circ \varphi^{-1}(\alpha_1(x_1) + \ldots + \alpha_n(x_n) + c) = \\ = \varphi(\alpha_1(x_1) + \ldots + \alpha_n(x_n) + c - d) = \\ = \varphi(\alpha_1(x_1) - \alpha_1(d) + \alpha_1(d) + \ldots + \alpha_n(x_n) - \\ - \alpha_n(d) + \alpha_n(d) + c - d) = \varphi(\alpha_1 \circ \varphi^{-1}(x_1) + \ldots + \\ + \alpha_n \circ \varphi^{-1}(x_n) + \alpha_1(d) + \ldots + \alpha_n(d) + c - d) = \\ = \varphi \circ \alpha_1 \circ \varphi^{-1}(x_1) +' \ \ldots +' \varphi \circ \alpha_n \circ \varphi^{-1}(x_n) +' c' = \\ = \alpha_1'(x_1) +' \ \ldots +' \alpha_n'(x_n) +' c',\end{gathered}$$

где $c' = \alpha_1(d) + \ldots + \alpha_n(d) + c$ — элемент множества Q. □

В дальнейшем мы будем считать, что таблица Кэли операций индексирована естественным образом (то есть набором $(0, 1, \ldots, k-1)$).

Рассмотрим введенную в [13] процедуру GetGroup, принимающую на вход таблицу Кэли C квазигруппы (в нашем случае достаточно рассмотреть произвольную фиксацию переменных $x_3, \ldots, x_n$) и возвращающую таблицу Кэли абелевой группы, над которой эта квазигруппа может быть аффинной, или "неаффинна", если такой группы не существует.

```
1. рассмотреть первую строку C
                    как перестановку s
2. вычислить перестановку t, обратную s
3. рассмотреть строки C как перестанов-
ки и умножить каждую строку на t справа
4. обозначить результат через C1
5. переупорядочить строки C1 так,
   что первый столбец и первая строка
                              совпадают
6. обозначить результат через C2
7. проверить, что матрица C2 симмет-
рична
8. если C2 несимметрична
9. вернуть "неаффинна"
10.  проверить, что C2 задает
                ассоциативную операцию
11.если нет,
12. вернуть "неаффинна"
13.иначе
14. вернуть C2
```

Легко увидеть, что в случае, если процедура возвращает $C2$, нейтральным элементом операции является 0.

Лемма 6. *Сложность процедуры составляет* $O(k^3)$ *при* $k \to \infty$.

Доказательство. Первые две строки легко реализовать со сложностью порядка $k \log k$; третья, пятая и седьмая строки имеют квадратичную сложность. Наконец, строка 10 реализуется кубической по сложности проверкой тождества $C2[a, C2[b, c]] \equiv C2[C2[a, b], c]$. Оставшиеся строки имеют константную сложность. □

В работе [14] предлагается ряд методов оптимизации проверки ассоциативности.

Пусть процедура GetGroup вернула абелеву группу с таблицей Кэли $C2$. Для удобства обозначим соответствующую операцию через +. Заметим, что при произвольной фиксации любых $n-2$ переменных функции f будет получаться квазигрупповая операция, в таблице Кэли которой найдутся строка и столбец, начинающиеся с 0 (это следует из того, что первая строка и первый столбец являются перестановками и содержат все элементы, включая 0). Пусть найденная таким образом строка или столбец соответствуют вариации i-й переменной функции f. Тогда соответствующая перестановка обозначается через α_i. Поиск α_i несложно реализовать с линейной сложностью. Дополнительно проверим, что α_i является автоморфизмом относительно операции + (это можно сделать, проверив справедливость тождества $\alpha_i(a + b) \equiv \alpha_i(a) + \alpha_i(b)$; непосредственная проверка имеет сложность $O(k^2)$). Рассмотрим процедуру поиска автоморфизмов GetAut(i), вычисляющую описанным выше способом α_i и проверяющую, что это автоморфизм. В случае, если α_i автоморфизмом не является, процедура возвращает "неаффинна". Оформим приведенные рассуждения о сложности в виде утверждения.

Лемма 7. *Сложность процедуры GetAut*(i) *составляет* $O(k^2)$ *при* $k \to \infty$.

Рассмотрим следующий алгоритм проверки неаффинности.

```
1. C2 = GetGroup
2. если C2 равно "неаффинна"
3.   вернуть "неаффинна"
4. цикл по i от 1 до n
5.   alpha_i равно GetAut(i)
6.   если alpha_i равно "неаффинна"
7.     вернуть "неаффинна"
8. конец цикла
9. c = f (0, …, 0)
10. проверить тождество f(x1, …, xn) =
                = alpha_1 (x1) + |… +
                   + alpha_n (xn) + c
11. если тождество выполнено
12.   вернуть "аффинна"
13. иначе
14. вернуть "неаффинна"
```

Лемма 8. *Пусть алгоритм вернул "аффинна" для n-квазигруппы (Q, f). Тогда эта n-квазигруппа аффинна.*

Доказательство. Утверждение является следствием определения аффинности, так как тождество, справедливость которого установлено на строке 10, есть тождество (2.1), выписанное для абелевой операции + и автоморфизмов $\alpha_1, \ldots, \alpha_n$. □

Лемма 9. *Пусть n-квазигруппа (Q, f) аффинна. Тогда алгоритм вернет "аффинна".*

Доказательство. Пусть выполнено тождество (2.1). В силу леммы 5 без ограничения общности можно считать, что нейтральным элементом операции + является 0. Первая строка матрицы C в процедуре GetGroup, то есть перестановка s, имеет вид $\alpha(x) + v$, где α – автоморфизм группы $(Q, +)$, $v \in Q$, а оставшиеся строки с номерами $i = 2, \ldots, k$ имеют вид $\alpha(x) + v_i, v_i \in Q$. После умножения справа на $t = s^{-1}$ первая строка станет тождественной перестановкой, а оставшиеся примут вид $x + v_i'$. Следовательно, получившаяся после переупорядочения матрица $C2$ задает операцию $x + y$, являющуюся коммутативной и ассоциативной, и процедура GetGroup не вернет "неаффинна".

Перестановка α_i в процедуре GetAut(i) получается фиксацией оставшихся переменных, такой что $\alpha_i(0) = 0$. По предположению об аффинности, α_i совпадает с соответствующим автоморфизмом в представлении (2.1), и ни одна из процедур GetAut(i) не вернет "неаффинна".

Так как все автоморфизмы сохраняют 0, значение $f(0, \ldots, 0)$ в точности равно константе c в представлении (2.1), поэтому проверка справедливости тождества на строке 10 закончится успехом, и процедура вернет "аффинна".

Таблица 2. Время проверки неаффинности 3-квазигрупп

k	1	2	4	8
512	1/1	1/1	1/1	1/1
1024	4/5	4/4	3/3	2/3
2048	79/83	56/61	42/46	32/32

Теорема 2. *Предложенный алгоритм возвращает "неаффинна" тогда и только тогда, когда n-квазигруппа является неаффинной. Сложность проверки составляет $O(k^n)$ при фиксированном $n \geq 3$ и порядке n-квазигруппы k, стремящемся к бесконечности.*

Доказательство. Корректность алгоритма следует из лемм 8 и 9. Оценим сложность. По лемме 6 сложность восстановления операции + на строке 1 есть $O(k^3)$. По лемме 7 сложность цикла оценивается как $O(n \cdot k^2) = o(k^3)$. Сложность проверки справедливости тождества на строке 10 составляет $O(k^n)$ (с константной сложностью проверяется справедливость равенства для каждого из k^n входных наборов). Оставшиеся строки выполняются с константной сложностью. Так как $n \geq 3$, общая сложность составляет $O(k^n)$. □

Интересно, что при переходе от $n = 2$ к $n = 3$ порядок сложности не увеличивается.

4.1. Оценка практической эффективности

Алгоритм проверки неафинности для 3-квазигрупп был программно реализован с поддержкой OpenMP-распараллеливания. Программа была протестирована на рабочей станции с 8-ядерным процессором i7–3770 CPU @3.40GHz и 32 гигабайтами памяти на 3-квазигруппах, аналогичных тестовым примерам из [14]. Время работы представлено в табл. 2. В левом столбце приведен порядок 3-квазигрупп, в столбцах со второго по пятый содержатся максимальные времена работы в секундах на соответствующем числе ядер для неаффинных/аффинных 3-квазигрупп.

Полученные результаты подтверждают возможность проверки неафинности 3-квазигрупп за приемлемое время до порядка 2048 включительно. Заметим, что $k = 2048$ является максимально возможным порядком вида 2^m, $m \in \mathbb{N}$, при котором 3-квазигруппы допускают табличное задание при 32 гигабайтах оперативной памяти.

5. ПРОВЕРКА ПОЛИНОМИАЛЬНОЙ ПОЛНОТЫ

Так как полиномиальная полнота эквивалентна одновременной простоте и неаффинности, из теорем 1 и 2 вытекает следующий факт.

Следствие 1. *Предложенные алгоритмы осуществляют проверку полиномиальной полноты n-квазигруппы порядка k со сложностью $O(k^{n+1})$ при фиксированном n и $k \to \infty$.*

В ряде случаев одну из проверок можно не проводить. В силу равномощности классов эквивалентности все n-квазигруппы простого порядка простые, и в этом случае полиномиальная полнота эквивалентна неаффинности. Полностью аналогично [20, Предложение 3.2] можно показать, что аффинная n-квазигруппа может быть простой, только если $k = p^m$ для некоторого простого числа p и $m \in \mathbb{N}$, а абелева группа $(Q, +)$ изоморфна $\mathbb{Z}_p^m$. Поэтому в случае, когда порядок группы не является степенью простого числа, неаффинность непосредственно следует из простоты.

6. ПРОВЕРКА НАЛИЧИЯ N-ПОДКВАЗИГРУПП

В работе П.И. Собянина [21] был предложен алгоритм проверки наличия подквазигрупп, который может быть легко переформулирован для случая проверки наличия собственных n-подквазигрупп порядка $\geq d$ (наиболее интересными являются значения $d = 1, 2$) следующим образом. Для каждого d-элементного подмножества $G \subset Q$ вычисляется замыкание $f(G)$. Легко увидеть, что n-подквазигруппа порядка $\geq d$ существует тогда и только тогда, когда хотя бы одно из замыканий $f(G)$ отлично от Q. Вычисление одного замыкания несложно реализовать со сложностью $O(k^n)$, число d-элементных подмножеств составляет $O(k^d)$; таким образом, общая сложность есть $O(k^{n+d})$. Понизим сложность этого алгоритма, используя метод, введенный в работе [15] для случая проверки наличия подквазигрупп порядка $\geq d$. Основная идея заключается в следующем. Выбирается параметр $K = K(k)$, $d \leq K < k$. Для каждого d-элементного подмножества G вычисляется "частичное" замыкание, либо пока не будет найдена n-подквазигруппа (и тогда задача решена), либо пока размер частичного замыкания не станет равным K. Предположим, что на этом этапе n-подквазигруппа не найдена. Тогда для каждого из частичных замыканий G' рассматривается множество всех d-элементных подмножеств G' (обозначим его через G''). Для совокупности всех G'' строится система представителей. На последнем шаге для каждого представителя (это d-элементное подмножество Q) вычисляется полное замыкание; n-квазигруппа не содержит собственных подквазигрупп порядка $\geq d$ тогда и только тогда, когда все эти замыкания совпадают с Q. Дадим подробное описание алгоритма, докажем его корректность и оценим сложность.

Сперва рассмотрим процедуру GetClosure(G, K), принимающую на вход таблицу Кэли n-квазигруппы (Q, f), подмножество $G \subseteq Q$ и число K, $|G| \leq K \leq k$, и возвращающую "n-подквазигруппа найдена", если существует собственная n-подквазигруппа Q', такая что $G \subseteq Q'$ и $|Q'| \leq K$, и частичное замыкание $f(G)$, такое что $|f(G)| = K$, в противном случае. Заметим, что при $K = k$ алгоритм вернул бы полное замыкание.

```
1. добавить в очередь НаПроверку элементы G
2. создать множество Кандидаты = G
3. пока очередь НаПроверку непуста
4. извлечь элемент из очереди
                    (пусть это элемент x)
5. для каждого набора s (|s|=n) элементов
                        из Кандидаты и x,
              содержащего хотя бы один x
6. вычислить f(s)
7. если f(s) не лежит в Кандидаты
8.     добавить f(s) в очередь
9.     если |Кандидаты| = K
10.     выйти из циклов
11.    добавить f(s) в Кандидаты
12.  конец если
13. конец цикла для каждого…
14. конец цикла пока…
15. если очередь НаПроверку пуста
16.   вернуть "n-подквазигруппа найдена"
17. иначе
18. вернуть Кандидаты
```

По построению множество кандидатов является подмножеством $f(G)$. Выход из циклов на строке 10 означает, что размер полного замыкания $f(G) > K$, при этом число кандидатов в точности равно K. Наконец, штатное завершение циклов означает, что число кандидатов не превысило K, причем подстановка произвольного набора кандидатов (легко увидеть, что каждый такой набор возникнет на строке 5 ровно один раз) в функцию f дает на выходе элемент множества кандидатов, то есть в множестве кандидатов хранится n-подквазигруппа. Таким образом, процедура корректна. Оценим сложность. Все операции внутри циклов могут быть реализованы с

константной сложностью, поэтому общее время работы "циклической" части по порядку оценивается числом итераций. Общее число итераций оценивается сверху общим числом наборов из рассмотренных элементов очереди, то есть K^n. Оставшиеся операции могут быть реализованы с линейной сложностью. Таким образом, верно следующее утверждение.

Лемма 10. *Для произвольной n-квазигруппы (Q, f) процедура GetClosure(G, K) возвращает "n-подквазигруппа найдена" тогда и только тогда, когда существует n-подквазигруппа Q', такая что $G \subseteq Q'$ и $|Q'| \le K$. В противном случае возвращается множество $G' \subseteq f(G)$, такое что $|G'| = K$. Сложность процедуры составляет $O(K^n)$ при фиксированном n и $K \to \infty$.*

Заметим, что если Q' – нетривиальная n-подквазигруппа, то очевидным образом выполнено неравенство $|Q'| \le |Q|/2$, поэтому при практической реализации как только размер множества кандидатов превысит $k/2$, дальнейшие вычисления можно либо не проводить (если интересует только факт нетривиальности замыкания), либо произвольным образом дополнить множество кандидатов до нужного размера.

Пусть для каждого G процедура GetClosure(G, K) вернула множество G', то есть n-подквази группа порядка $\ge d$ пока не найдена. Перейдем от множеств G' к множествам G'', состоящим из всех d-элементных подмножеств соответствующего G'. В частности, при $d = 1$ G' и G'' совпадают, при $d = 2$ множество G'' состоит из всех неупорядоченных пар элементов соответствующего множества G'. Аналогично лемме 1 из [15] построим систему представителей для множеств G'' с помощью жадной процедуры GetRepresentatives. Процедура принимает на вход множества $M_1, \ldots, M_D$, $M_1, \ldots, M_D \subseteq M$, $|M_1| = \ldots = |M_D| = T$, $|M| = S$, и выдает на выход систему представителей M_0. На высоком уровне она устроена следующим образом.

```
1. сделать M0 пустым множеством
2. объявить все Mi непокрытыми
3. пока имеются непокрытые Mi
4.   выбрать m – наиболее частый элемент
                          в непокрытых Mi
5.   добавить m в M0
6.   объявить покрытыми все непокрытые Mi,
                              содержащие m
7. конец цикла
8. вернуть M0
```

То, что M_0 является системой представителей, непосредственно следует из построения. В лемме 1 работы [15] доказано, что число итераций цикла (равное размеру системы представителей) удовлетворяет соотношению

$$|M_0| \le \left(\frac{S}{T} + 1\right) \cdot (\log D + 2).$$

Рассмотрим реализацию процедуры в случаях $d = 1$ и $d = 2$ и оценим сложность.

При $d = 1$ имеем $D = S = k$, $T = K$. Сперва создадим массив V размера k, в котором для каждого i, $0 \le i \le k-1$, будем хранить число множеств G'', в которых встречается элемент i. Для заполнения этого массива достаточно пройти по всем множествам G'' ровно один раз, то есть сложность пропорциональна $K \cdot k$. Также создадим k списков L_j, хранящих идентификаторы множеств G'', содержащих элемент j. Для этого потребуется число операций и объем памяти, пропорциональные $k \cdot K$. На каждой итерации цикла будем выбирать максимальный элемент массива V (это потребует линейной по k сложности), добавлять этот элемент в множество M_0, затем уменьшать значения счетчиков в массиве V, соответствующих покрытым на этом шаге множествам. Так как для каждого элемента каждого множества G'' вычитание будет произведено ровно один раз, общая сложность этого шага пропорциональна $k \cdot K$, а итоговая сложность есть $O(k \cdot K) + O(k \cdot (k/K + 1) \cdot (\log k + 2))$, то есть $O(k \cdot K) + O(k^2 \cdot \log k/K)$. Заметим, что дополнительная память по порядку не превосходит объема таблицы Кэли исходной n-квазигруппы.

При $d = 2$ имеем $D = S = C_k^2 = k \cdot (k-1)/2$, $T = C_K^2 = K \cdot (K-1)/2$. Заметим, что в этом случае мы будем пользоваться частичными замыканиями пар элементов множества Q (на что потребуется $O(k^2 \cdot K)$ памяти) и не будем явно создавать массивы пар элементов. Создадим матрицу V размера $k \times k$, в которой для каждой пары (i, j), $0 \le i < j \le k-1$, будем хранить число множеств G'', в которых встречается пара элементов $\{i, j\}$. Для заполнения этого массива достаточно пройти по всем парам элементов множеств G'' ровно один раз, то есть сложность пропорциональна $K^2 \cdot k^2$. Также создадим $k \cdot (k-1)/2$ списков $L_{i,j}$, хранящих идентификаторы множеств G'', содержащих пары элементов i, j, $0 \le i < j \le k-1$. Для этого потребуется число операций, пропорциональное $k^2 \cdot K^2$, и объем памяти, пропорциональный $k^2 \cdot K$. На каждой итерации цикла будем выбирать максимальный элемент матрицы V (это потребует квадратичной по k сложности), добавлять индексы этого элемента в множество M_0 (это множество

состоит из пар элементов Q), затем уменьшать значения счетчиков в массиве V, соответствующих покрытым на этом шаге множествам. Так как для каждой пары элементов каждого множества G'' вычитание будет произведено ровно один раз, общая сложность этого шага пропорциональна $k^2 \cdot K^2$, а итоговая сложность есть $O(k^2 \cdot K^2) + O(k^2 \cdot (k^2/K^2) \cdot \log k)$. Объем дополнительной памяти равен $O(k^2 \cdot K)$. Заметим, что при $n \geq 3$ в силу неравенства $K \leq k$ дополнительная память по порядку заведомо не превосходит объема таблицы Кэли исходной квазигруппы.

Оформим результат в виде леммы.

Лемма 11. *Процедура GetRepresentatives корректно строит систему представителей множеств G''. В случае $d = 1$ мощность множества представителей не превосходит $(k/K + 1) \cdot (\log k + 2)$, временная сложность процедуры составляет $O(k \cdot K) + O(k^2 \cdot \log k/K)$, при этом используется дополнительная память, объем которой есть $O(k \cdot K)$. В случае $d = 2$ мощность множества представителей не превосходит $(k(k-1)/(K(K-1)) + 1) \cdot (\log(k(k-1)/2) + 2)$, временная сложность процедуры составляет $O(k^2 \cdot K^2) + O((k^4/K^2) \cdot \log k)$, при этом используется дополнительная память, объем которой есть $O(k^2 \cdot K)$.*

Наконец, сформулируем итоговый алгоритм.

```
1. для каждого
        d-элементного подмножества G
2.   вычислить GetClosure (G, K)
3.   если результат
          "n-подквазигруппа найдена"
4.   вернуть "есть n-подквазигруппы
                      порядка >= d"
5. конец цикла
6. GetRepresentatives
7. для каждого представителя m
8.   вычислить GetClosure (m, k)
9.   если результат
          "n-подквазигруппа найдена"
10.   вернуть "есть n-подквазигруппы
                      порядка >= d"
11.конец цикла
12.вернуть "n-подквазигруппы
            порядка >= d отсутствуют"
```

Для обоснования корректности нам потребуется два вспомогательных утверждения, аналогичных леммам 2 и 3 из работы [15].

Лемма 12. *Пусть (Q, f) – n-квазигруппа, $d \in \mathbb{N}$, $G_1, \ldots, G_t \subset Q$, $|G_1| = \ldots = |G_t| = d$, $M_i \subseteq f(G_i)$, $|M_i| \geq d$, $i = 1, \ldots, t$, M_i' – множество всех d-элементных подмножеств M_i, M_0 – система представителей множеств M_i'. Тогда из полноты всех $m_0 \in M_0$ следует полнота всех G_i.*

Доказательство. Предположим противное: найдется G_i, не являющееся полным. По условию существует $m_0 \in M_0$, такое что $m_0 \subseteq f(G_i)$. В силу монотонности замыкания

$$f(m_0) \subseteq f(f(G_i)) = f(G_i) \neq Q,$$

что противоречит условию. □

Лемма 13. *Пусть выполнены все условия леммы 12 и дополнительно множества S_i представляют собой все d-элементные подмножества Q. Тогда n-квазигруппа (Q, f) имеет собственные n-подквазигруппы порядка $\geq d$ если и только если существует* представитель $m_0 \in M_0$, *не являющийся полным.*

Доказательство. Достаточность очевидна (примером искомой n-подквазигруппы является $f(m_0)$). Необходимость является следствием леммы 12 и того, что все d-элементные подмножества входят в $\{G_i\}$, $i = 1, \ldots, t$. □

Наконец, сформулируем основное утверждение раздела. Для доказательства потребуется выбрать подходящее значение параметра $K = K(k)$.

Теорема 3. *Представленный алгоритм возвращает "n-подквазигруппы порядка $\geq d$ отсутствуют" тогда и только тогда, когда n-квазигруппа (Q, f) не имеет собственных n-подквазигрупп порядка $\geq d$. При этом в случае $d = 1$ параметр K может быть выбран таким образом, что временная сложность составляет $O\left(k^{\frac{n^2+n+1}{n+1}} \log^{\frac{n}{n+1}} k\right)$, а пространственная сложность есть $O(k^n)$ при фиксированном n и $k \to \infty$. В случае $d = 2$ параметр K может быть выбран таким образом, что временная сложность составляет $O\left(k^{\frac{n^2+2n+4}{n+2}} \log^{\frac{n}{n+2}} k\right)$, а пространственная сложность есть $O(k^n)$ при фиксированном $n \geq 3$ и $O(k^{5/2} \log^{1/3} k)$ при $n = 2$ и $k \to \infty$.*

Доказательство. Корректность алгоритма непосредственно вытекает из лемм 12 и 13.

Оценим сложность алгоритма в случае $d = 1$. Из леммы 10 следует, что сложность построения частичных замыканий всех одноэлементных подмножеств Q есть $O(k \cdot K^n)$. Из леммы 11 вытекает, что сложность построения системы представителей равна $O(k \cdot K) + O(k^2 \cdot \log k/K)$. Из лемм 10 и 11 следует, что сложность вычисления замыканий представителей есть $O(k^n \cdot (k/K + 1) \cdot (\log k + 2))$.

Положим $K(k) = \min(k-1, [c \cdot (k^n \cdot \log k)^{1/(n+1)}])$, где квадратные скобки означают взятие целой части, а c – некоторая положительная наперед заданная константа. Если $K(k) < 1$, положим $K = 1$. Очевидно, что для любых $n \geq 2$ и $c > 0$ найдется такое $k_0 \in \mathbb{N}$, что для любого $k \geq k_0$ выполнено равенство $K(k) = [c \cdot (k^n \cdot \log k)^{1/(n+1)}]$ и неравенства

$$2 \leq c \cdot (k^n \cdot \log k)^{1/(n+1)} - 1 \leq$$
$$\leq K \leq c \cdot (k^n \cdot \log k)^{1/(n+1)}.$$

При фиксированном $n \geq 2$ и $k \to \infty$ верны следующие оценки (без ограничения общности считаем, что $k \geq k_0$).

$$k \cdot K^n \leq c^n \cdot k \cdot k^{\frac{n^2}{n+1}} \cdot \log^{\frac{n}{n+1}} k =$$
$$= O\left(k^{\frac{n^2+n+1}{n+1}} \cdot \log^{\frac{n}{n+1}} k\right)$$

$$k \cdot K = o(k \cdot K^n)$$

$$\frac{k^2 \cdot \log k}{K} \leq \frac{k^2 \log k}{(c \cdot k^n \cdot \log k)^{1/(n+1)} - 1} \sim$$
$$\sim \frac{k^2 \log k}{(c \cdot k^n \cdot \log k)^{1/(n+1)}} = \frac{k^{\frac{n+2}{n+1}} \cdot \log^{\frac{n}{n+1}} k}{c^{\{1/(n+1)\}}} =$$
$$= o\left(k^{\frac{n^2+n+1}{n+1}} \cdot \log^{\frac{n}{n+1}} k\right)$$

$$k^n \cdot (k/K + 1) \cdot (\log k + 2) \leq k^n \cdot 2\frac{k}{K} \cdot 2\log k \leq$$
$$\leq \frac{4 \cdot k^{n+1} \cdot \log k}{c \cdot (k^n \cdot \log k)^{1/(n+1)} - 1} \sim$$
$$\sim \frac{4 \cdot k^{n+1} \cdot \log k}{c \cdot (k^n \cdot \log k)^{1/(n+1)}} = O\left(k^{\frac{n^2+n+1}{n+1}} \cdot \log^{\frac{n}{n+1}} k\right)$$

Из этих оценок непосредственно следует искомая оценка временной сложности.

Объем таблицы Кэли составляет $O(k^n)$. В силу лемм 10, 11 объем дополнительной памяти есть $O(k \cdot K)$. Так как $K = o(k)$, при $n \geq 2$ второе значение пренебрежимо мало в сравнении с первым, откуда вытекает оценка пространственной сложности.

Перейдем к рассмотрению случая $d = 2$. Из леммы 10 следует, что сложность построения частичных замыканий всех двухэлементных подмножеств есть $O(k^2 \cdot K^n)$. По лемме 11 сложность построения системы представителей равна $O(k^2 \cdot K^2) + O(k^4 \cdot \log k / K^2)$. Из лемм 10 и 11 вытекает, что общая сложность построения полных замыканий представителей есть

$$O(k^n \cdot (k(k-1)/(K(K-1)) + 1) \times$$
$$\times (\log(k(k-1)/2) + 2)).$$

Положим

$$K(k) = \min(k-1, [c \cdot (k^n \cdot \log k)^{1/(n+2)}]),$$

где c – некоторая положительная наперед заданная константа. Если $K(k) < 2$, положим $K = 2$. Очевидно, что для любых $n \geq 2$ и $c > 0$ найдется такое $k_0 \in \mathbb{N}$, что для любого $k \geq k_0$ выполнено равенство $K(k) = [c \cdot (k^n \cdot \log k)^{1/(n+2)}]$ и неравенства

$$2 \leq c \cdot (k^n \cdot \log k)^{1/(n+2)} - 1 \leq$$
$$\leq K \leq c \cdot (k^n \cdot \log k)^{1/(n+2)}.$$

При фиксированном $n \geq 2$ и $k \to \infty$ верны следующие оценки (без ограничения общности считаем, что $k \geq k_0$).

$$k^2 \cdot K^n \leq k^2 \cdot c^n \cdot k^{\frac{n^2}{n+2}} \cdot \log^{\frac{n}{n+2}} k =$$
$$= O\left(k^{\frac{n^2+2n+4}{n+2}} \cdot \log^{\frac{n}{n+2}} k\right)$$

$$k^2 \cdot K^2 \leq k^2 \cdot K^n = O\left(k^{\frac{n^2+2n+4}{n+2}} \cdot \log^{\frac{n}{n+2}} k\right)$$

$$\frac{k^4 \cdot \log k}{K^2} \leq \frac{k^4 \cdot \log k}{(c \cdot (k^n \cdot \log k)^{1/(n+2)} - 1)^2} \sim$$
$$\sim \frac{k^4 \cdot \log k}{c^2 (k^n \cdot \log k)^{2/(n+2)}} = \frac{k^{\frac{2n+8}{n+2}} \cdot \log^{\frac{n}{n+2}} k}{c^2} =$$
$$= O\left(k^{\frac{n^2+2n+4}{n+2}} \cdot \log^{\frac{n}{n+2}} k\right)$$

$$k^n \cdot \left(\frac{k(k-1)}{K(K-1)} + 1\right) \cdot (\log(k(k-1)/2) + 2) \lesssim$$
$$\lesssim k^n \cdot \frac{8k^2}{(K-1)^2} \cdot \log k \leq$$
$$\leq \frac{8k^{n+2} \cdot \log k}{(c \cdot (k^n \cdot \log k)^{1/(n+2)} - 2)^2} \sim$$
$$\sim \frac{8k^{n+2} \cdot \log k}{c^2 \cdot (k^n \cdot \log k)^{2/(n+2)}} = \frac{8k^{\frac{n^2+2n+4}{n+2}} \cdot \log^{\frac{n}{n+2}} k}{c^2} =$$
$$= O\left(k^{\frac{n^2+2n+4}{n+2}} \cdot \log^{\frac{n}{n+2}} k\right)$$

Таблица 3. Порядок сложности проверки наличия собственных n-подквазигрупп

$n = 2$	$n = 3$	$n = 4$
k^3	k^4	k^5
$k^{7/3}\log^{2/3} k$	$k^{13/4}\log^{3/4} k$	$k^{21/5}\log^{4/5} k$

Таблица 4. Порядок сложности проверки наличия нетривиальных n-подквазигрупп

$n = 2$	$n = 3$	$n = 4$
k^4	k^5	k^6
$k^3\log^{1/2} k$	$k^{19/5}\log^{3/5} k$	$k^{14/3}\log^{2/3} k$

Таблица 5. Время проверки существования собственных 3-подквазигрупп

k	256	512	1024	2048
Easy	0	6	236	
QC0.25	1	15	693	
QC0.5	0	14	147	
QC1	1	1	20	
QC2	0	0	4	
QC4	0	1	5	
QP2C0.25	0	11	515	
QP2C0.5	0	9	105	
QP2C1	0	1	13	
QP2C2	0	0	3	
QP2C4	0	1	4	
QP4C0.25	0	9	508	
QP4C0.5	0	8	101	
QP4C1	0	1	10	
QP4C2	0	0	2	
QP4C4	0	1	3	
QP8C0.25	0	9	419	1887
QP8C0.5	0	8	84	441
QP8C1	0	1	11	58
QP8C2	0	0	3	5
QP8C4	0	0	3	8

Отметим, что в случае $n = 2$ все четыре слагаемых имеют одинаковый порядок.

Из приведенных оценок непосредственно вытекает искомая оценка временной сложности.

Объем таблицы Кэли составляет $O(k^n)$. В силу лемм 10, 11 объем дополнительной памяти есть $O(k^2 \cdot K)$. При $n = 2$ выполнены соотношения

$$k^2 \cdot K \sim k^2 \cdot k^{1/2} \cdot \log^{1/3} k = O(k^{5/2} \cdot \log^{1/3} k),$$

$$k^2 = o(k^{5/2} \cdot \log^{1/3} k).$$

При $n \geq 3$ выполнены соотношения

$$k^2 \cdot K \sim k^2 \cdot k^{n/(n+2)} \cdot \log^{1/(n+2)} k = o(k^n).$$

Таким образом, оценка пространственной сложности доказана. □

Заметим, что в случае $n = 2$ выигрыш во временной сложности по сравнению с работой [15] получился за счет незначительного проигрыша в пространственной сложности.

Содержательно параметр c позволяет регулировать объем памяти – увеличение c приводит к уменьшению теоретической временной сложности за счет роста теоретической пространственной сложности. На практике, однако, при реализации на OpenMP-архитектуре повышение параметра c может привести к замедлению программы из-за ограниченности пропускной способности канала обмена с памятью. Этот эффект можно наблюдать в табл. 5, 6.

Для наглядности выпишем порядки сложности полученных алгоритмов для небольших значений n и отдельной строкой приведем оценки сложности, получаемые с помощью обобщения алгоритма из работы [21]. В табл. 3 сведены результаты для проверки существования произвольных собственных n-подквазигрупп, в табл. 4 – результаты для проверки существования нетривиальных n-подквазигрупп, то есть собственных n-подквазигрупп порядка ≥ 2. Верхняя строка таблиц соответствует обобщению [21], нижняя – оптимизированному алгоритму.

6.1. Оценка практической эффективности

Алгоритм проверки существования собственных n-подквазигрупп и нетривиальных n-подквазигрупп для 3-квазигрупп был программно реализован с поддержкой OpenMP-распараллеливания. Программа была протестирована на рабочей станции с 8-ядерным процессором i7–3770 CPU @3.40GHz и 32 гигабайтами памяти на 3-квази группах, аналогичных тестовым примерам из [14], с различными значениями параметра c.

В табл. 5 приведены максимальные времена (в секундах) проверки наличия собственных 3-подквазигрупп. Столбцы соответствуют различным порядкам 3-квазигрупп. В строке Easy приведены времена работы обобщения алгоритма из статьи [21] на одном вычислительном ядре (здесь и далее пустые клетки означают, что время счета превысило предустановленный порог, так что программа не закончила вычисления). Строки вида QPNCC соответствуют оптимизированному алгоритму, распараллеленному с помощью OpenMP на N ядер и ис-

Таблица 6. Время проверки существования нетривиальных 3-подквазигрупп

k	256	512	1024	2048
Easy	34	1340		
QC0.25	6	142	737	
QC0.5	1	12	91	
QC1	1	3	31	
QC2	0	1	15	
QC4	0	1	26	
QP2C0.25	4	93	551	
QP2C0.5	1	5	85	
QP2C1	1	1	26	
QP2C2	0	1	10	
QP2C4	0	1	23	
QP4C0.25	3	57	353	
QP4C0.5	1	5	52	
QP4C1	1	2	16	
QP4C2	1	1	7	
QP4C4	0	1	15	
QP8C0.25	3	55	324	17397
QP8C0.5	1	5	46	1585
QP8C1	0	1	15	196
QP8C2	1	1	6	42
QP8C4	0	1	11	84

пользующему параметр-константу $c = 1/C$. Если пара PN не указана, вычисление проводилось на одном ядре. Заметим, что $k = 2048$ является максимально возможным порядком вида 2^m, $m \in \mathbb{N}$, при котором 3-квазигруппы допускают табличное задание при 32 гигабайтах оперативной памяти.

В табл. 6 приведены аналогичные данные для проверки наличия нетривиальных 3-подквази групп.

Полученные результаты подтверждают возможность проверки за приемлемое время наличия собственных 3-подквазигрупп и нетривиальных 3-подквазигрупп в 3-квазигруппах до порядка 2048 включительно.

7. ЗАКЛЮЧЕНИЕ

В рамках работы рассматриваются n-квазигруппы, заданные таблицами Кэли. Представлены алгоритмы для проверки полиномиальной полноты n-квазигрупп и существования собственных n-подквазигрупп и нетривиальных n-подквази групп, то есть собственных n-подквазигрупп порядка ≥ 2. Полиномиальная полнота обеспечивает труднорешаемость задачи проверки разрешимости уравнений над заданной n-квазигруппой, то есть защиту от атак методом составления уравнений на биты ключа. Отсутствие собственных n-подквазигрупп обеспечивает невырожденность преобразования на меньшие подмножества, при этом в ряде случаев “неподвижные точки”, то есть n-подквазигруппы порядка 1, считаются допустимыми.

Известно, что полиномиальная полнота n-квазигруппы эквивалентна одновременной простоте и неаффинности. В работе представлен алгоритм проверки простоты, имеющий сложность $O(k^{n+1})$, и алгоритм проверки неаффинности, имеющий сложность $O(k^n)$, при фиксированном $n \geq 3$ и порядке n-квазигрупп $k \to \infty$.

Для решения задачи проверки наличия собственных n-подквазигрупп предложен алгоритм, сложность которого составляет $O\left(k^{\frac{n^2+n+1}{n+1}} \log^{\frac{n}{n+1}} k\right)$ при фиксированном $n \geq 3$ и $k \to \infty$. Также предложен алгоритм для проверки наличия нетривиальных n-подквазигрупп, имеющий сложность $O\left(k^{\frac{n^2+2n+4}{n+2}} \log^{\frac{n}{n+2}} k\right)$ при фиксированном $n \geq 2$ и $k \to \infty$ (включение $n = 2$ обусловлено тем, что в этом случае удалось уточнить оценку, полученную в работе [15]). Все алгоритмы были программно реализованы с использованием OpenMP-распараллеливания и протестированы на широком классе примеров.

В дальнейшем планируется изучить возможность дополнительной оптимизации алгоритмов (прежде всего проверки простоты), рассмотреть другие модели распараллеливания (в частности, MPI), а также и провести тестирование на машинах с большими объемами памяти. Отдельной задачей является исследование функционального подхода к заданию n-квазигрупповых операций. В частности, в случае полиномиального задания операций на первый план выходит проблема канонического представления полиномиального выражения (см., например, [22]).

СПИСОК ЛИТЕРАТУРЫ

1. *Shannon C.* Communication theory of secrecy systems // The Bell System Technical Journal, 1949. V. 28. № 4. P. 656–715.
2. *Глухов М.М.* О применениях квазигрупп в криптографии // Прикладная дискретная математика. 2008. № 2. С. 28–32.
3. *Shcherbacov V.A.* Quasigroups in cryptology // Computer Science Journal of Moldova. 2009. V. 17. № 2 (50). P. 193–228.
4. *Gligoroski D., Ødegård R.S., Mihova M., Knapskog S.J., Drapal A., Klíma V., Amundsen J., El-Hadedy M.* Cryptographic hash function EDON-R’ // Proceedings of

the 1st International Workshop on Security and Communication Networks, 2009. P. 1–9.

5. *Dömösi P., Horváth G.* A novel cryptosystem based on abstract automata and Latin cubes // Studia Scientiarum Mathematicarum Hungarica. 2015. V. 52. № 2. P. 221–232.
6. *Xu M., Tian Z.* An image cipher based on Latin cubes // Proceedings of the 3rd International Conference on Information and Computer Technologies, 2020. P. 160–168.
7. *Dimitrova V., Mihajloska H.* Classification of ternary quasigroups of order 4 applicable in cryptography // Proceedings of the 7th International Conference for Informatics and Information Technology (CIIT 2010), 2010. P. 145–148.
8. *Галатенко А.В., Панкратьев А.Е., Староверов В.М.* Проверка полиномиальной полноты n-квазигрупп // Материалы XVIII Международной конференции “Алгебра, теория чисел и дискретная геометрия: современные проблемы, приложения и проблемы истории”. Тула, 2020. С. 146–150.
9. *Галатенко А.В., Панкратьев А.Е., Староверов В.М.* Об одном алгоритме проверки существования нетривиальных n-подквазигрупп // Материалы XIX Международной конференции “Алгебра, теория чисел, дискретная геометрия и многомасштабное моделирование: современные проблемы, приложения и проблемы истории”. Тула, 2021. С. 100–103.
10. *Artamonov V.A., Chakrabarti S., Gangopadhyay S., Pal S.K.* On Latin squares of polynomially complete quasigroups and quasigroups generated by shifts // Quasigroups and Related Systems. 2013. V. 21. № 2. P. 117–130.
11. *Horváth G., Nehaniv C.L., Szabó Cs.* An assertion concerning functionally complete algebras and NP-completeness // Theoretical Computer Science. 2008. V. 407. № 1. P. 591–595.
12. *Галатенко А.В., Панкратьев А.Е., Родин С.Б.* О полиномиально полных квазигруппах простого порядка // Алгебра и логика. 2018. Т. 57. № 5. С. 509–521.
13. *Галатенко А.В., Панкратьев А.Е.* О сложности проверки полиномиальной полноты конечных квазигрупп // Дискретная математика. 2018. Т. 30. № 4. С. 3–11.
14. *Galatenko A.V., Pankratiev A.E., Staroverov V.M.* Efficient verification of polynomial completeness of quasigroups // Lobachevskii Journal of Mathematics. 2020. V. 41. № 8. P. 1444–1453.
15. *Галатенко А.В., Панкратьев А.Е., Староверов В.М.* Об одном алгоритме проверки существования подквазигрупп // Чебышевский сборник. 2021. Т. 22. № 2. С. 76–89.
16. *Абрамов С.А., Боголюбская А.А.* Семинар по компьютерной алгебре в 2019–2020 гг. // Программирование. 2021. № 2. С. 3–4.
17. *Яблонский С.В.* Введение в дискретную математику. М.: Высшая школа, 2008. 384 с.
18. *Hagemann J., Herrmann C.* Arithmetical locally equational classes and representation of partial functions // Universal Algebra, Esztergom (Hungary). 1982. V. 29. P. 345–360.
19. *Вьюкова Н.И., Галатенко В.А., Самборский С.В.* Средства динамического анализа программ в компиляторах gcc и clang // Программирование. 2020. № 4. С. 46–64.
20. *Artamonov V.A., Chakrabarti S., Pal S.K.* Characterizations of highly non-associative quasigroups and associative triples // Quasigroups and Related Systems. 2017. V. 25. № 1. P. 1–19.
21. *Собянин П.И.* Об алгоритме проверки наличия подквазигруппы в квазигруппе // Интеллектуальные системы. Теория и приложения. 2019. Т. 23. № 2. С. 79–84.
22. *Шпиз Г.Б., Крюков А.П.* Каноническое представление полиномиальных выражений с индексами // Программирование. 2019. № 2. С. 66–72.

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 54–64

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.422.8

АНАЛИТИКО-ЧИСЛЕННАЯ РЕАЛИЗАЦИЯ АЛГЕБРЫ ПОЛИВЕКТОРОВ НА ЯЗЫКЕ JULIA

© 2022 г. М. Н. Геворкян[a,*], А. В. Демидова[a,], Т. Р. Велиева[a,b,***], А. В. Королькова[a,****], Д. С. Кулябов[a,c,*****]**

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

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

[c] *Объединенный институт ядерных исследований, Лаборатория информационных технологий, ул. Жолио-Кюри, 6, Дубна, 141980 Московская область, Россия*

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

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

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

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

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

Поступила в редакцию 02.08.2021 г.
После доработки 02.09.2021 г.
Принята к публикации 19.09.2021 г.

Геометрическая алгебра основывается на трудах Грассмана и Клиффорда. Основными изучаемыми объектами являются p-векторы (поливекторы) и мультивекторы. Поливекторы вкупе с операцией внешнего умножения дают реализацию алгебры Грассмана, а мультивекторы с операцией геометрического умножения реализуют алгебру Клиффорда. Алгебра мультивекторов позволяет обобщить многие операции и объекты из аналитической и дифференциальной геометрий, такие как векторное и смешанное произведения, векторы нормали и бинормали и т.д. на многомерный случай и дать им наглядную геометрическую интерпретацию. Комплексные числа и кватернионы изоморфны мультивекторам специального вида. В работе рассмотрено применение идей геометрической алгебры для решения некоторых прикладных задач, которые встречаются в компьютерной геометрии. Для решения данных задач используется пакет Grassmann.jl для языка Julia.

DOI: 10.31857/S0132347422010058

1. ВВЕДЕНИЕ

Тензорный формализм является крайне общим. Он описывает произвольные пространства с группой симметрии $GL(n)$. Однако пространство, в котором мы живем, обладает более узкой группой симметрии. В трехмерном случае нам достаточно группы $O(3)$, а в четырехмерном случае – группы Лоренца Λ. Группа Лоренца является более узкой, чем общая линейная группа, но при этом имеет больше специфических свойств, удобных для описания нашего пространства. Поэтому представляется оправданным использовать более специфический инструментарий, чем тензоры, а именно спиноры. Спиноры являются элементами алгебры Клиффорда. Обычно используют матричное представление спиноров, которое, по историческим причинам, является крайне громоздким. Одним из формализмов, призванных упростить манипуляции со спинорами, является формализм геометрической (и пространственно-временной) алгебры.

В данной статье дается краткий обзор модуля Grassmann.jl [1], который реализует внешнюю алгебру (p-векторы и внешнее произведение) и алгебру мультивекторов (реализация абстрактной алгебры Клиффорда). Если внешняя алгебра широко известна и изучается в расширенном курсе алгебры и дифференциальной геометрии для физико-математических специальностей [2], то геометрическая алгебра гораздо менее распространена.

Геометрическая алгебра довольно прикладная область. Ее результаты нашли применение в области машинной графики и компьютерной геометрии [3–6]. Ее концепции обладают большой обобщающей силой. Так, например, для двумерного евклидова ортонормированного пространства, мультивекторы второго ранга изоморфны

комплексным числам, а в случае трехмерного пространства те же мультивекторы изоморфны уже телу кватернионов. Более того, к операциям геометрической алгебры сводится векторное и смешанное умножения, ориентированная площадь и ориентированный объем.

При выборе языка программирования, на котором должен быть реализован формализм геометрической алгебры мы опирались на идею *одного языка*. Под этим мы подразумеваем использование одного языка как для манипуляции с символьными данными, так и для численных расчетов[1]. Ранее мы экспериментировали с созданием рабочего конвейера с использованием разнообразных программных средств [8].

В качестве такого языка мы выбрали язык Julia [9]. Хотя основной парадигмой языка Julia являются высокопроизводительные вычисления [10], а этот язык является своего рада наследником языка FORTRAN [11], в данном случае для нас не является основной причиной выбора языка. Для нас важнее всего была встроенная поддержка создания на основе языка Julia новых предметно-ориентированных языков и поддержка оберток для сторонних программ и библиотек [12].

1.1. Структура статьи

В разделе 2 кратко перечислены некоторые специальные факты из теории поливекторов. В разделе 3 рассмотрены случаи двумерного и трех- мерного векторных пространств. Раздел 4 посвящен изложению базовых концепций геометрической алгебры – геометрического умножения и понятия мультивектора. В разделе 5 подробно разобран пример описания трехмерных вращений с помощью мультивектора особого вида, называемого *ротором*. В ходе изложения подчеркивается ряд преимуществ, по сравнению с кватернионами, которые на данный момент стали стандартом де-факто для описания поворотов в трехмерном евклидовом пространстве. В разделе 6 на базе изложенного материала дается обзор основных возможностей модуля Grassmann.jl. Данный модуль, как и сам язык Julia, для которого он написан, находится еще в начальной стадии развития, поэтому страдает от явных недоделок. По ходу работы над примерами встречаются несколько таких моментов.

1.2. Обозначения и соглашения

1. Контравариантные векторы (векторы) будем обозначать строчными латинскими буквами и выделять с помощью полужирного шрифта. Например: $\mathbf{v}$, $\mathbf{x}$ и т.д. Над греческими буквами будем ставить значок стрелки, например: $\vec{\boldsymbol{\omega}}$.

2. Линейное пространство контравариантных векторов будем обозначать буквой L и полагать наличие базиса $\langle \mathbf{e}_1, \ldots, \mathbf{e}_n \rangle$. Нумерацию векторов базиса будем вести нижними индексами от 1 до n. Буквой n всегда будем обозначать размерность L т.е. $\dim L = n$.

3. Под полем скаляров будем подразумевать поле действительных чисел $\mathbb{R}$, хотя многие формулы и утверждения остаются справедливыми для любого произвольного числового поля R. Скаляры будем обозначать строчными греческими буквами α, β и т.д.

4. Будем полагать, что пространство L наделено структурой скалярного произведения. Кроме того, будем полагать, что в пространстве L можно задать ортонормированный базис $\langle \mathbf{e}_1, \ldots, \mathbf{e}_n \rangle$.

5. Скалярное произведение векторов $\mathbf{u}$ и $\mathbf{v}$ будем обозначать как $(\mathbf{u}, \mathbf{v})$, векторное произведение как $[\mathbf{u}, \mathbf{v}]$ или $\mathbf{u} \times \mathbf{v}$ и смешанное произведение трех векторов как $(\mathbf{u}, \mathbf{v}, \mathbf{w})$.

2. ОСНОВНЫЕ СВЕДЕНИЯ ИЗ ГЕОМЕТРИЧЕСКОЙ АЛГЕБРЫ

При изложении материала мы предполагаем, что читатель знаком с основами тензорной алгебры, по крайней мере в объеме, изложенном в книге [2].

2.1. Внешнее умножение и внешняя алгебра

Контравариантные кососимметричные тензоры валентности (ранга) $(0, p)$ называют *p-векторами* или *поливекторами*. Обозначать p-вектор произвольного ранга будем также, как и обычные векторы – полужирным латинскими буквами. Когда ранг непонятен из контекста, будем указывать его внизу буквы, например: $\underset{p}{\mathbf{u}}$.

Относительно операции сложения и умножения на скаляр, множество p-векторов образует линейные пространства, обозначаемые как $\Lambda^p(L)$, где пространство L – линейное пространство контравариантных векторов с базисом $\langle \mathbf{e}_1, \ldots, \mathbf{e}_n \rangle$. Полагают, что $\Lambda^0(L) = \mathbf{R}$ и $\Lambda^1(L) = L$, то есть 1-вектор является контравариантным вектором.

Рассмотрим пространство $\Lambda(L)$, которое представляет собой прямую сумму пространств p-векторов

$$\Lambda(L) = \Lambda^0(L) \oplus \Lambda^1(L) \oplus \Lambda^2(L) \oplus \Lambda^3(L) \oplus \ldots$$

Введем на этом пространстве операцию $\wedge\colon \Lambda(L) \times \Lambda(L) \to \Lambda(L)$ называемую *внешним про-*

[1] Впрочем, следует заметить, что первоначально в Julia принцип одного языка означал использование одного языка как для прототипирования, так и для разработки высокопроизводительной программы [7].

изведением. Для любых поливекторов $\mathbf{u} \in \Lambda^p(L)$, $\mathbf{v} \in \Lambda^q(L)$, $\mathbf{w} \in \Lambda^r(L)$ внешнее умножение определяется следующими свойствами:

• $\mathrm{rank}(\underset{p}{\mathbf{u}} \wedge \underset{q}{\mathbf{v}}) = p + q$.

• $\mathbf{u} \wedge (\mathbf{v} \wedge \mathbf{w}) = (\mathbf{u} \wedge \mathbf{v}) \wedge \mathbf{w}$ – ассоциативность.

• $1 \wedge \mathbf{u} = \mathbf{u} \wedge 1 = \mathbf{u}$, где 1 – скалярная единица (единичный элемент).

• $\alpha \wedge \beta = \alpha \cdot \beta$ – для скаляров $\wedge$ эквивалентно простому умножению.

• $\underset{p}{\mathbf{u}} \wedge \underset{q}{\mathbf{v}} = (-1)^{pq} \underset{q}{\mathbf{v}} \wedge \underset{p}{\mathbf{u}}$ – антикоммутативность для нечетных валентностей.

• $(\mathbf{u} + \mathbf{v}) \wedge \mathbf{w} = \mathbf{u} \wedge \mathbf{w} + \mathbf{v} \wedge \mathbf{w}$ – дистрибутивность (правая).

• $\mathbf{w} \wedge (\mathbf{u} + \mathbf{v}) = \mathbf{w} \wedge \mathbf{u} + \mathbf{w} \wedge \mathbf{v}$ – дистрибутивность (левая).

• $\mathbf{u} \wedge (\alpha\mathbf{v}) = (\alpha\mathbf{u}) \wedge \mathbf{v} = \alpha(\mathbf{u} \wedge \mathbf{v})$ это свойство в купе с дистрибутивностью дает билинейность операции $\wedge$.

Операция внешнего произведения наделяет пространство $\Lambda(L)$ структурой ассоциативной алгебры над полем скаляров. Такая алгебра называется *внешней алгеброй* пространства L (или *алгеброй Грассмана*) [2].

Все возможные p-векторы $\mathbf{e}_{i_1} \wedge \mathbf{e}_{i_2} \wedge \ldots \wedge \mathbf{e}_{i_p}$, составленные из базисных векторов пространства L, образуют базис пространства p-векторов $\Lambda^p(L)$. Так как между собой поливекторы $\mathbf{e}_{i_1} \wedge \mathbf{e}_{i_2} \wedge \ldots \wedge \mathbf{e}_{i_p}$ отличаются лишь знаком, то можно выбрать один из них, расположив индексы по возрастанию: $1 \le i_1 < i_2 < \ldots < i_p \le n$. Такой порядок индексов называется *естественным*, а выбранный базисный p-вектор *значимым*.

2.2. Размерность пространства и ранг p-вектора

При работе с p-векторами следует учитывать следующие размерности и ранги.

• Размерность пространства $L = \langle \mathbf{e}_1, \ldots \mathbf{e}_n \rangle$ обозначается как $\dim L = n$.

• Ранг p-вектора равен p. Указывается при обозначении пространства p-векторов: $\Lambda^p(L)$. Вместо термина ранг также часто используют термин *порядок* (grade) и обозначение *gr* [4, 13] и иногда термин *step* [3].

• Размерность пространства $\Lambda^p(L)$ равна наибольшему рангу, который может быть у ненулевого p-вектора в данном пространстве L. Она также равна количеству значимых базисных p-векторов и обозначается как $\dim \Lambda^p(L) = C_n^p$.

• Размерность пространства $\Lambda(L) = \Lambda^0(L) \oplus \Lambda^1(L) \oplus \Lambda^2(L) \oplus \Lambda^3(L) \oplus \ldots$ равна сумме размерностей всех подпространств, составляющих прямую сумму: $\sum_{i=0}^{n} C_n^i = 2^n$.

• Формально можно полагать ранг нулевого p-вектора любым (удобным для манипуляций).

Базис пространства $\Lambda^p(L)$ зависит как от ранга p, так и от n (размерности L). p-вектор будет иметь разное количество компонент в зависимости от размерности пространства L на котором мы его рассматриваем.

• На $\Lambda^2(L)$ базис задается 2-векторами $\mathbf{e}_{ij} = \mathbf{e}_i \wedge \mathbf{e}_j$, где $i < j$ и $i, j = 1, \ldots, n$.

• На $\Lambda^p(L)$ базис задается p-векторами $\mathbf{e}_{i_1 \ldots i_p} = \mathbf{e}_{i_1} \wedge \ldots \wedge e_{i_p}$, где $i_1 < \ldots < i_p$ и $i_1, \ldots, i_p = 1, \ldots, n$.

• На $\Lambda^n(L)$ базис задается одним n-вектором $\mathbf{e}_{12\ldots n} = \mathbf{e}_1 \wedge \mathbf{e}_2 \ldots \wedge \mathbf{e}_n$.

Обозначения $\mathbf{e}_{ij}$, $\mathbf{e}_{i_1 i_2 i_3}$ и т.д. мы ввели для краткости.

В литературе [4, 13] 2-вектор называют *бивектором*, 3-вектор – *тривектором* и 4-вектор – *квадривектором*. Для p-форм малой размерности таких специальных названий авторы не встречали.

2.3. Базисные n-векторы в n-мерном пространстве

Особый случай возникает, когда ранг рассматриваемого объекта совпадает с размерностью пространства L то есть $p = n = \dim L$. Тогда размерность пространства $\Lambda^n(L)$ равна $C_n^n = 1$ и существует только один значимый базисный n-вектор:

$$\mathbf{e}_{12\ldots n} = \mathbf{e}_1 \wedge \mathbf{e}_2 \ldots \wedge \mathbf{e}_n.$$

Можно считать, что n-вектор $\mathbf{e}_{12\ldots n}$ имеют одну единичную компоненту.

Базисный n-вектор $\mathbf{e}_{12\ldots n}$ задает *элемент единичного объема*. В геометрической алгебре базисный n-вектор обозначают как $\mathbf{E}_n$ или $\mathbf{I}_n$.

2.4. Разложимость

Разложимым называют p-вектор $\mathbf{V}$, который выражается через внешнее произведение p различных векторов из L:

$$\mathbf{V} = \mathbf{v}_1 \wedge \mathbf{v}_2 \wedge \ldots \wedge \mathbf{v}_p$$

В литературе на английском языке для обозначения разложимого вектора используют термин *blade* [3, 4]. Реже встречается термин *простой* (simple) вектор.

Формулировка необходимых и достаточных условий для разложимости p-вектора не является тривиальной задачей. Она была решена в работе [14]. В пространстве L размерности n разложимыми являются любые p-векторы, при $p \in \{0, 1, n-1, n\}$. В трехмерном пространстве все p-векторы являются разложимыми, но уже для четырехмерного пространства это неверно. Например, бивектор следующего вида

$$\mathbf{V} = \mathbf{e}_{12} + \mathbf{e}_{34} = \mathbf{e}_1 \wedge \mathbf{e}_2 + \mathbf{e}_3 \wedge \mathbf{e}_4$$

в четырехмерном пространстве нельзя записать в виде $\mathbf{V} = \mathbf{v}_1 \wedge \mathbf{v}_2$ ни для каких $\mathbf{v}_1$ и $\mathbf{v}_2$ из L [3, стр. 46].

Отдельно стоит заметить, что бивектор $\mathbf{V} = \mathbf{V}^{ij}\mathbf{e}_i \wedge \mathbf{e}_j$ разложим тогда и только тогда, когда $V \wedge V = 0$ [2, § 6.3.5].

2.5. Операция дополнения и ее свойства

Правым дополнением (right complement) базисного p-вектора $\mathbf{B} \in \Lambda^p(L)$, где $\dim L = n$ называется такой $(n-p)$-вектор $\overline{\mathbf{B}} \in \Lambda^{n-p}(L)$, что

$$\mathbf{B} \wedge \overline{\mathbf{B}} = \mathbf{E}_n.$$

Соответственно, *левым дополнением* (left complement) базисного p-вектора $\mathbf{B} \in \Lambda^p(L)$ называется такой $(n-p)$-вектор $\underline{\mathbf{B}} \in \Lambda^{n-p}(L)$, что

$$\underline{\mathbf{B}} \wedge \mathbf{B} = \mathbf{E}_n.$$

Иногда, базис состоящий из дополнений называют *кобазисом* [13].

Для любого разложимого p-вектора, в частности, для базисного p-вектора $\mathbf{B}$ выполняется следующее соотношение

$$\underline{\mathbf{B}} = (-1)^{p(n-p)}\overline{\mathbf{B}}$$

Данное соотношение показывает, что правое и левое дополнение совпадают в случае, если p – четное число и различаются знаком, если p – нечетное.

Также для разложимого p-вектора $\mathbf{B}$, где p – нечетное, выполняется равенство:

$$\overline{\overline{\mathbf{B}}} = \underline{\underline{\mathbf{B}}} = (-1)^{p(n-p)}\mathbf{B}$$

Для любого разложимого p-вектора:

$$\underline{\overline{\mathbf{A}}} = \mathbf{A}$$

Определив операцию дополнения для базисных p-векторов, можно распространить ее на произвольный p-вектор, потребовав линейности:

- $\overline{\alpha\mathbf{V}} = \alpha\overline{\mathbf{A}}$, где α – скаляр.
- $\overline{\mathbf{A} + \mathbf{B}} = \overline{\mathbf{A}} + \overline{\mathbf{B}}$.

Правое дополнение должно подчиняться тем же свойствам.

После этого не составляет труда вычислять дополнительные (комплиментарные) p-векторы. Так для $\mathbf{v} \in L$, $\dim L = 3$ получим:

$$\overline{\mathbf{v}} = \overline{v^i\mathbf{e}_i} = v^i\overline{\mathbf{e}_i} = v^1\overline{\mathbf{e}}_1 + v^2\overline{\mathbf{e}}_2 + v^3\overline{\mathbf{e}}_3 = \\ = v^1\mathbf{e}_{23} - v^2\mathbf{e}_{13} + v^3\mathbf{e}_{12}$$

Вектор $\mathbf{v} \in \Lambda^1(L) = L$, а бивектор $\overline{\mathbf{v}} \in \Lambda^2(L)$, но благодаря одинаковым размерностям L и $\Lambda^2(L)$ (при условии $n = \dim L = 3$) возможно взаимно однозначное соответствие, устанавливаемое операцией дополнения.

3. ДВУМЕРНЫЕ И ТРЕХМЕРНЫЕ ВЕКТОРНЫЕ ПРОСТРАНСТВА

3.1. Векторное произведение

Пусть L есть трехмерное пространство с базисом $\langle \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \rangle$. Найдем внешнее произведение двух векторов $\mathbf{u}$ и $\mathbf{v}$

$$\mathbf{u} \wedge \mathbf{v} = (u^1v^2 - u^2v^1)\mathbf{e}_{12} + \\ + (u^1v^3 - u^3v^1)\mathbf{e}_{13} + (u^2v^3 - u^3v^2)\mathbf{e}_{23}.$$

Найдем дополнение к бивектору $\mathbf{u} \wedge \mathbf{v}$

$$\overline{\mathbf{u} \wedge \mathbf{v}} = (u^1v^2 - u^2v^1)\mathbf{e}_3 - (u^1v^3 - u^3v^1)\mathbf{e}_2 + \\ + (u^2v^3 - u^3v^2)\mathbf{e}_1 = \\ = \begin{pmatrix} u^2v^3 - u^3v^2 \\ u^3v^1 - u^1v^3 \\ u^1v^2 - u^2v^1 \end{pmatrix} \in \Lambda^1(L) = L$$

Мы получили векторное произведение $\mathbf{u} \times \mathbf{v}$

$$\mathbf{u} \times \mathbf{v} = \overline{\mathbf{u} \wedge \mathbf{v}}$$

Операция дополнения позволяет найти 1-вектор, ортогональный бивектору $\mathbf{u} \times \mathbf{v}$. Если же ее применить к 1-вектору, то, наоборот, получаем бивектор, <<ортогональный>> к исходному вектору. Для больших размерностей данная интерпретация также сохраняется, но теряет наглядность.

3.2. Смешанное произведение

На том же трехмерном пространстве L найдем внешнее произведение трех векторов $\mathbf{u}$, $\mathbf{v}$ и $\mathbf{w}$.

$$\mathbf{u} \wedge \mathbf{v} \wedge \mathbf{w} = u^iv^jw^k\mathbf{e}_i \wedge \mathbf{e}_j \wedge \mathbf{e}_k = \\ = u^iv^jw^k\varepsilon_{ijk}\mathbf{E}_3 = \begin{vmatrix} u^1 & v^1 & w^1 \\ u^2 & v^2 & w^2 \\ u^3 & v^3 & w^3 \end{vmatrix}\mathbf{E}_3,$$

где ε_{ijk} – символ Леви-Чивиты.

Найдем дополнение к тривектору $\mathbf{u} \wedge \mathbf{v} \wedge \mathbf{w}$

$$\overline{\mathbf{u}\wedge\mathbf{v}\wedge\mathbf{w}} = u^i v^j w^k \varepsilon_{ijk}\overline{\mathbf{E}}_3 = u^i v^j w^k \varepsilon_{ijk}$$

Это не что иное, как смешанное произведение трех векторов $\mathbf{u}$, $\mathbf{v}$ и $\mathbf{w}$:

$$(\mathbf{u},\mathbf{v},\mathbf{w}u,v,w) = \overline{\mathbf{u}\wedge\mathbf{v}\wedge\mathbf{w}}$$

3.3. Комплексная структура

Рассмотрим теперь $L = \langle \mathbf{e}_1, \mathbf{e}_2 \rangle$ и некоторый вектор $\mathbf{u} \in L$. Найдем правое дополнение к $\mathbf{e}_1$, $\mathbf{e}_2$

$$\overline{\mathbf{e}}_1 = \mathbf{e}_2 \quad \text{так как} \quad \mathbf{e}_1 \wedge \mathbf{e}_2 = \mathbf{E}_2,$$
$$\overline{\mathbf{e}}_2 = -\mathbf{e}_1 \quad \text{так как} \quad \mathbf{e}_2 \wedge (-\mathbf{e}_1) = \mathbf{e}_1 \wedge \mathbf{e}_2 = \mathbf{E}_2.$$

Найдем теперь дополнение к $\mathbf{u}$:

$$\overline{\mathbf{u}} = u^1\overline{\mathbf{e}}_1 + u^2\overline{\mathbf{e}}_2 = u^1\mathbf{e}_2 - u^2\mathbf{e}_1 = \begin{pmatrix} -u^2 \\ u^1 \end{pmatrix}$$

Это не что иное, как комплексная структура на двумерном декартовом пространстве:

- поворот на угол $\frac{\pi}{2}$:

$$J\mathbf{u} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} u^1 \\ u^2 \end{pmatrix}\begin{pmatrix} -u^2 \\ u^1 \end{pmatrix},$$

- поворот на угол $-\frac{\pi}{2}$:

$$\underline{\mathbf{u}} = u^1\underline{\mathbf{e}}_1 + u^2\underline{\mathbf{e}}_2 = -u^1\mathbf{e}_2 + u^2\mathbf{e}_1 = \begin{pmatrix} u^2 \\ -u^1 \end{pmatrix}.$$

Вновь рассмотрим двумерное пространство L. Найдем смешанное произведение двух векторов $\mathbf{u}$ и $\mathbf{v}$:

$$\mathbf{u}\wedge\mathbf{v} = u^i v^j \mathbf{e}_i \wedge \mathbf{e}_j = u^i v^j \varepsilon_{ij}\mathbf{E}_2 =$$
$$= \begin{vmatrix} u^1 & v^1 \\ u^2 & v^2 \end{vmatrix}\mathbf{E}_2 = (u^1 v^2 - u^2 v^1)\mathbf{E}_2.$$

Найдем дополнение от бивектора $\mathbf{u}\wedge\mathbf{v}$

$$\overline{\mathbf{u}\wedge\mathbf{v}} = (u^1 v^2 - u^2 v^1)\overline{\mathbf{E}}_2 = (u^1 v^2 - u^2 v^1).$$

Мы получили ориентированную площадь параллелограмма, построенного на векторах $\mathbf{u}$ и $\mathbf{v}$. Ориентированная площадь может служить геометрической интерпретацией бивектора.

4. ГЕОМЕТРИЧЕСКОЕ ПРОИЗВЕДЕНИЕ И ГЕОМЕТРИЧЕСКАЯ АЛГЕБРА

4.1. Мультивекторы и геометрическое произведение

Если допустить сложение элементов $\Lambda(L)$ различного ранга, то мы получим более общую структуру, задаваемую операцией сложения и внешнего умножения. Объект из $\Lambda(L)$, состоящий из элементов различного ранга, называется *мультивектором* или *числом Клиффорда* [15]:

$$\mathbf{M} = \alpha + \mathbf{u} + \underset{2}{\mathbf{V}} + \underset{3}{\mathbf{W}} + \ldots$$

Поскольку $\Lambda(L)$ является градуированной алгеброй, то в этом выражении знак суммы надо воспринимать также как и знак суммы в комплексных числах и кватернионах. Этот знак связывает скаляры, векторы, бивекторы, тривекторы и т.д. в единую сущность.

Операция геометрического умножения обладает большей общностью, чем операции скалярного и внешнего произведений. Поэтому разумно определить ее аксиоматически, а не выражать через скалярное и внешнее произведения. Тем не менее, мы будем предполагать, что в пространстве L задан метрический тензор $\mathbf{g}(\mathbf{u},\mathbf{v})$.

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, определенному в L.

- Геометрическое произведение вектора $\mathbf{u}$ самого на себя равно скаляру, значения которого определяется метрическим тензором:

$$\mathbf{u}^2 \equiv \mathbf{u}\mathbf{u} = \mathbf{g}(\mathbf{u},\mathbf{u}) = \|\mathbf{u}\|^2.$$

- Дистрибутивность:

$$\mathbf{u}(\mathbf{v}+\mathbf{w}) = \mathbf{u}\mathbf{v} + \mathbf{u}\mathbf{w},$$
$$\mathbf{u}(\mathbf{v}+\mathbf{w}) = \mathbf{u}\mathbf{v} + \mathbf{u}\mathbf{w}.$$

- 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.

Множество $\Lambda(L)$ с ассоциативной операцией геометрического произведения называется *алгеброй мультивекторов* и является реализацией абстрактной алгебры Клиффорда, обобщающей алгебру Грассмана и алгебру кватернионов Гамильтона.

Из аксиом следует формула

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

которую можно использовать для определения геометрического произведения векторов.

4.2. Обратный элемент и деление

Из свойства $\mathbf{u}\mathbf{u} = \|\mathbf{u}\|^2 \equiv u^2$ следует, что для любого ненулевого вектора $\mathbf{u}$ существует обратный элемент $\frac{\mathbf{u}}{\|\mathbf{u}\|^2}$:

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

Можно определить операцию (правого) деления, как произведение *справа* на обратный элемент:

$$\frac{\mathbf{u}}{\mathbf{v}} = \mathbf{u}\mathbf{v}^{-1} = \frac{\mathbf{u}\mathbf{v}}{\|\mathbf{u}\|^2}.$$

Если умножение произвести *слева*, то получим иное выражение: $\mathbf{v}^{-1}\mathbf{u} = \frac{\mathbf{v}\mathbf{u}}{\|\mathbf{v}\|^2}$ так как $\mathbf{u}\mathbf{v} \neq \mathbf{v}\mathbf{u}$.

Используя операцию деления можно записать произвольный вектор $\mathbf{u}$ относительно некоторого вектора $\mathbf{v}$ как:

$$\mathbf{u} = \frac{\mathbf{u}}{\mathbf{v}}\mathbf{v} = \mathbf{u}\frac{\mathbf{v}}{\|\mathbf{v}\|^2}\mathbf{v} = \mathbf{u}\mathbf{v}\frac{\mathbf{v}}{\|\mathbf{u}\|^2} =$$
$$= \frac{\mathbf{u}\mathbf{v}}{\mathbf{v}} = \frac{(\mathbf{u},\mathbf{v})}{\mathbf{v}} + \frac{\mathbf{u}\wedge\mathbf{v}}{\mathbf{v}}.$$

Первое слагаемое $\mathbf{u}_{\|\mathbf{v}}$ является проекцией вектора $\mathbf{u}$ на $\mathbf{v}$, а второе слагаемое $\mathbf{u}_{\perp\mathbf{v}}$ является ортогональной компонентой вектора $\mathbf{u}$ относительно вектора $\mathbf{v}$.

4.3. Геометрическое произведение произвольных мультивекторов

Обобщение геометрического произведения на произвольные p-векторы, требует обощения скалярного произведения. Однако, в случае если введен ортонормированный базис, можно обойтись без такого обобщения.

Рассмотрим евклидово пространство $L = \langle \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n \rangle$ и найдем геометрические произведения базисных векторов $\mathbf{e}_i$.

Произведение базисного вектора самого на себя находится из определения:

$$\mathbf{e}_i\mathbf{e}_i = (\mathbf{e}_i, \mathbf{e}_i) = \mathbf{e}_i^2.$$

Для произведения $\mathbf{e}_i\mathbf{e}_j$, где $i \neq j$ можно доказать следующее равенство [3]

$$\mathbf{e}_i\mathbf{e}_j = -\mathbf{e}_j\mathbf{e}_i, \quad i \neq j,$$

Где $\mathbf{e}_i\mathbf{e}_j = \mathbf{e}_i \wedge \mathbf{e}_j$. Вообще, для любого ортонормированного базиса справедливо тождество $\mathbf{e}_{i_1} \ldots \mathbf{e}_{i_p}\mathbf{e}_1 \wedge \ldots \wedge \mathbf{e}_p$ то есть базисные p-векторы могут быть записаны через геометрическое произведение.

Пользуясь указанными выше свойствами, можно находить геометрическое произведение любых мультивекторам, достаточно знать их разложение по базисным p-векторам. Например:

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

Это справедливо, так как

$$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 = -\mathbf{e}_1\underbrace{\mathbf{e}_2\mathbf{e}_2}_{1}\mathbf{e}_1\mathbf{e}_3 = -\underbrace{\mathbf{e}_1\mathbf{e}_1}_{=1}\mathbf{e}_3 = -\mathbf{e}_3,$$
$$\mathbf{e}_1\mathbf{e}_3\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 = \mathbf{e}_1\underbrace{\mathbf{e}_3\mathbf{e}_3}_{=1}\mathbf{e}_1\mathbf{e}_2 = \underbrace{\mathbf{e}_1\mathbf{e}_1}_{=1}\mathbf{e}_2 = \mathbf{e}_2.$$

4.4. Случай трехмерного пространства

В трехмерном декартовом пространстве мультивектор в общем случае будет иметь следующий вид:

$$\mathbf{V} = v^0 + v^1\mathbf{e}_1 + v^2\mathbf{e}_2 + v^3\mathbf{e}_3 +$$
$$+ v^{12}\mathbf{e}_1\mathbf{e}_2 + v^{23}\mathbf{e}_2\mathbf{e}_3 + v^{13}\mathbf{e}_1\mathbf{e}_3 + v^{123}\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$$

Обозначим $\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$ как $\mathbf{I}$. Тогда верно равенство

$$\mathbf{I}\mathbf{I} = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 = \mathbf{e}_1\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3\mathbf{e}_2\mathbf{e}_3 =$$
$$= -\mathbf{e}_1\mathbf{e}_1\mathbf{e}_2\mathbf{e}_2\mathbf{e}_3\mathbf{e}_3 = -1 \Rightarrow \boxed{\mathbf{I}^2 = -1}.$$

Также, переставляя сомножители и правильно меняя знак, можно показать, что

$$\mathbf{e}_1\mathbf{I} = +\mathbf{e}_2\mathbf{e}_3, \quad \mathbf{I}\mathbf{e}_1 = +\mathbf{e}_2\mathbf{e}_3,$$
$$\mathbf{e}_2\mathbf{I} = -\mathbf{e}_1\mathbf{e}_3, \quad \mathbf{I}\mathbf{e}_2 = -\mathbf{e}_1\mathbf{e}_3,$$
$$\mathbf{e}_3\mathbf{I} = +\mathbf{e}_1\mathbf{e}_2, \quad \mathbf{I}\mathbf{e}_3 = +\mathbf{e}_1\mathbf{e}_2,$$
$$\mathbf{e}_1\mathbf{e}_2\mathbf{I} = -\mathbf{e}_3, \quad \mathbf{I}\mathbf{e}_1\mathbf{e}_2 = -\mathbf{e}_3,$$
$$\mathbf{e}_1\mathbf{e}_3\mathbf{I} = +\mathbf{e}_2, \quad \mathbf{I}\mathbf{e}_1\mathbf{e}_3 = +\mathbf{e}_2,$$
$$\mathbf{e}_2\mathbf{e}_3\mathbf{I} = -\mathbf{e}_1, \quad \mathbf{I}\mathbf{e}_2\mathbf{e}_3 = -\mathbf{e}_1.$$

Видно, что $\mathbf{I}$ коммутирует с базисными векторами и бивекторами: $\mathbf{I}\mathbf{e}_i = \mathbf{e}_i\mathbf{I}$ и $\mathbf{I}\mathbf{e}_i\mathbf{e}_j = \mathbf{e}_i\mathbf{e}_j\mathbf{I}$ из чего следует, что произвольный мультивектор $\mathbf{U}$ в трехмерном пространстве коммутирует с $\mathbf{I}$:

$$\mathbf{I}\mathbf{U} = \mathbf{U}\mathbf{I}.$$

Также для произвольного бивектора справедливы равенства

$$\mathbf{I}\underset{2}{\mathbf{U}} = \underset{1}{\mathbf{v}} \quad \text{и} \quad \underset{2}{\mathbf{U}}\mathbf{I} = -\underset{1}{\mathbf{v}},$$

что дает возможность записать произвольный мультивектор в виде

$$\mathbf{V} = v^0 + \mathbf{u} + \mathbf{v}\mathbf{I} + v^{123}\mathbf{I}$$

Кроме того, с помощью $\mathbf{I}$ можно связать внешнее и векторное произведения:

$$\mathbf{I}\mathbf{u} \times \mathbf{v} = \mathbf{u} \wedge \mathbf{v}, \quad \mathbf{u} \times \mathbf{v} = -\mathbf{I}\mathbf{u} \wedge \mathbf{v}$$

Обозначим

$$\mathbf{i} = \mathbf{e}_1\mathbf{e}_2, \quad \mathbf{j} = \mathbf{e}_2\mathbf{e}_3, \quad \mathbf{k} = \mathbf{e}_1\mathbf{e}_3.$$

Данные бивекторы ведут себя точно также как и мнимые единицы кватернионов. Можно составить таблицу умножения:

	1	**i**	**j**	**k**
1	1	**i**	**j**	**k**
i	**i**	−1	**k**	−**j**
j	**j**	−**k**	−1	−**i**
k	**k**	**j**	−**i**	−1

Кроме того $\mathbf{ijk} = -1$.

Таким образом в трехмерном пространстве множество мультивекторов вида

$$\mathbf{V} = v^0 + \underset{2}{\mathbf{v}}$$

изоморфно множеству кватернионов.

5. ИСПОЛЬЗОВАНИЕ КВАТЕРНИОНОВ ДЛЯ ОПИСАНИЯ ПОВОРОТОВ В ТРЕХМЕРНОМ ПРОСТРАНСТВЕ

Кватернионами [16] называют гиперкомплексные числа вид $q = q_0 + iq_1 + jq_2 + kq_3$, где i, j и k – мнимые единицы, обладающие следующими свойствами: $i^2 = j^2 = k^2 = ijk = 1$, $ij = -ji = k$, $jk = -kj = i$ и $ki = -ik = j$.

В терминах общей алгебры говорят, что множество кватернионов с операцией сложения и умножения образуют *тело*, то есть ассоциативное, унитарное, некоммутативное по умножению кольцо. Обратный элемент в таком кольце существует для любого ненулевого элемента.

Мнимые единицы i, j, k ассоциируют с тремя ортами **i**, **j** и **k**. Число q_0 называют скалярной частью кватерниона, а элемент $q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ – векторной частью, которой дают простую геометрическую интерпретацию в виде вектора $\mathbf{q} = (q_1, q_2, q_3)^{\mathrm{T}}$. Кватернион, состоящий только из векторной части, называют *чистым*. Кватернион в общем случае записывают как

$$q = q_0 + \mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}.$$

Для кватерниона определяют норму и комплексное сопряжение:

$$\|q\|^2 = q_0^2 + (\mathbf{q}, \mathbf{q}) = q_0^2 + q_1^2 + q_2^2 + q_3^2$$
$$\text{и} \quad q^* = q_0 - \mathbf{q}.$$

Кватернион с нормой, равной единице, называют *нормированным*. Обратный элемент для произвольного кватерниона вычисляется как $q^{-1} = \dfrac{q^*}{\|q\|^2}$, а для нормированного обратным является сопряженный к нему кватернион: $q^{-1} = q^*$.

Нормированный кватернион можно представить в особенно простом тригонометрическом виде:

$$q = q_0 + \mathbf{q} = \cos\theta + \mathbf{u}\sin\theta, \quad \mathbf{u} = \frac{\mathbf{q}}{|\mathbf{q}|},$$

где угол θ лежит в полуинтервале $(-\pi, \pi]$. При умножении двух кватернионов с углами α и β соответственно, получаем кватернион с углом $\alpha + \beta$.

Кватернионы нашли свое применение в теории трехмерных вращений. Если поставить в соответствие некоторому вектору в трехмерном евклидовом пространстве чистый кватернион $v = \mathbf{v} = \mathbf{i}v^1 + \mathbf{j}v^2 + \mathbf{k}v^3$, то следующая операция

$$\mathbf{w} = q\mathbf{v}q^* = (2q_0^2 - 1)\mathbf{v} + 2(\mathbf{q}, \mathbf{v})\mathbf{q} + 2q_0\mathbf{q} \times \mathbf{v}$$

эквивалентна повороту вектора **v** на угол 2θ вокруг оси **q**, где q – нормированный кватернион $q = \cos\theta + \mathbf{u}\sin\theta$. Вращение в противоположную сторону задается похожей операцией:

$$\mathbf{w} = q^*\mathbf{v}q = (2q_0^2 - 1)\mathbf{v} + 2(\mathbf{v}, \mathbf{q})\mathbf{q} + 2q_0\mathbf{v} \times \mathbf{q}.$$

Вращение с помощью кватернионов стало стандартом де-факто в области компьютерной графики, вытеснив альтернативное описание с помощью углов Эйлера. С чисто алгоритмической точки зрения, кватернионное описание вращений очень эффективно, однако менее наглядно из-за сложностей в геометрической интерпретации угла θ. Далее рассмотрим подход к описанию поворотов с помощью мультивекторов особого вида, который столь же алгоритмически прост, но при этом имеет гораздо большую общность и геометрическую наглядность.

5.1. Повороты в трехмерном пространстве

Рассмотрим вектор **v** и умножим его *слева* на некоторый вектор **a**, а *справа* на обратный вектор $\mathbf{a}^{-1}$. Так как операция геометрического произведения не коммутативна, то мы получим вектор **v'** отличный от **v**. Пользуясь тем, что $\mathbf{u}_{\|\mathbf{v}} = \dfrac{(\mathbf{u}, \mathbf{v})}{\mathbf{v}}$ и $\mathbf{u}_{\perp\mathbf{v}} = \dfrac{\mathbf{u} \wedge \mathbf{v}}{\mathbf{v}}$ получим:

$$\mathbf{v}' = \mathbf{a}\mathbf{v}\mathbf{a}^{-1}\frac{\mathbf{a}\mathbf{v}}{\mathbf{a}} = \frac{(\mathbf{a}, \mathbf{v})}{\mathbf{a}} + \frac{\mathbf{a} \wedge \mathbf{v}}{\mathbf{a}} =$$
$$= \frac{(\mathbf{v}, \mathbf{a})}{\mathbf{a}} - \frac{\mathbf{v} \wedge \mathbf{a}}{\mathbf{a}} = \mathbf{v}_{\|\mathbf{a}} - \mathbf{v}_{\perp\mathbf{a}}.$$

Таким образом, вектор **v'** является отражением вектора **v** относительно вектора **a**. Для двумерного случая данная процедура проиллюстрирована на рис. 1.

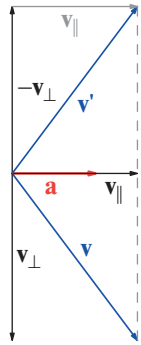


Рис. 1. Отражение вектора **v** относительно вектора **a**

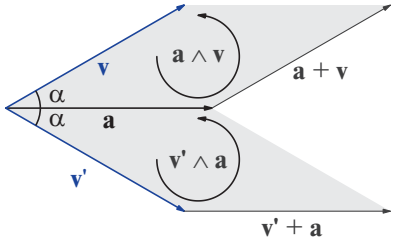


Рис. 2. Равенство бивекторных частей мультивекторов **av** и **v'a**

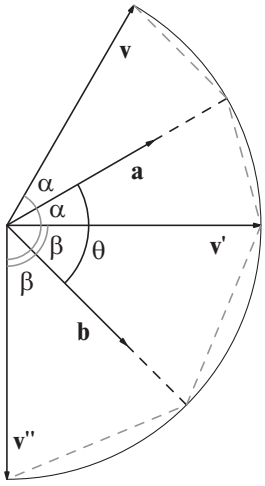


Рис. 3. Поворот

Заметим, что длина (норма) вектора **v** сохраняется. Так как по определению геометрического произведения $\mathbf{aa} = \|a\|^2$, то

$$\|\mathbf{ava}^{-1}\|^2 = \mathbf{ava}^{-1}\mathbf{ava}^{-1} = \frac{1}{\|a\|^4}\mathbf{avaava} =$$

$$= \frac{1}{\|a\|^4}\mathbf{av}\|\mathbf{a}\|^2\mathbf{va} = \frac{1}{\|a\|^2}\mathbf{a}\|\mathbf{v}\|^2\mathbf{a} =$$

$$= \|\mathbf{v}\|^2\frac{1}{\|a\|^2}\mathbf{aa} = \|\mathbf{v}\|^2.$$

В случае единичного вектора **a** обратный вектор $\mathbf{a}^{-1}$ совпадает с самим **a** так как норма **a** = 1 и операция отражения сводится к умножению на вектор **a** слева и справа: **v'** = **ava**.

Вектор **v** – отражение вектора **v** относительно вектора **a**. Длины векторов **v'** и **v** совпадают и значения углов $\angle(\mathbf{v},\mathbf{a})$ и $\angle(\mathbf{a},\mathbf{v}')$. Следовательно, геометрические произведения **av** и **v'a** совпадают:

$$\mathbf{v}'\mathbf{a} = \mathbf{ava}^{-1}\mathbf{a} = \mathbf{av}.$$

На рис. 2 показаны бивекторы $\mathbf{a}\wedge\mathbf{v}$ и $\mathbf{v}'\wedge\mathbf{a}$.

Применим к вектору **v** последовательно два отражения: относительно вектора **a**, а затем относительно вектора **b**:

$$\mathbf{v}' = \mathbf{ava}^{-1},$$

$$\mathbf{v}'' = \mathbf{bv}'\mathbf{b}^{-1} = \mathbf{bava}^{-1}\mathbf{b}^{-1} = \mathbf{bav}(\mathbf{ba})^{-1}.$$

Мультивектор **ba** обозначают как **R** и называют *ротором* (вращателем).

$$\mathbf{v}'' = \mathbf{RvR}^{-1}$$

R вращает **v** на угол 2θ от **a** к **ab**, где $\theta = \angle(\mathbf{a},\mathbf{b})$ (см. рис. справа). Вращение происходит от **a** к **a** параллельно плоскости соответствующей бивектору $\mathbf{a}\wedge\mathbf{b}$, а не **ba**, поэтому **R** записывают как

$$\mathbf{R} = (\mathbf{b},\mathbf{a}) + \mathbf{b}\wedge\mathbf{a} = (\mathbf{a},\mathbf{b}) - \mathbf{a}\wedge\mathbf{b}$$

Формула для вращения $\mathbf{v}' = \mathbf{RvR}^{-1}$ остается справедливой для любой размерности, а не только для двумерного или трехмерного случая.

Ротор **R** можно применять к объектам более высокого порядка, чем векторы, например:

$$\mathbf{RuvR}^{-1} = \mathbf{RuR}^{-1}\mathbf{RvR}^{-1} = (\mathbf{RuR}^{-1})(\mathbf{RvR}^{-1}).$$

Рассмотрим трехмерный случай для нормированных векторов **a** и **b**. Так как

$$\mathbf{a}\wedge\mathbf{b} = \overline{\mathbf{a}\times\mathbf{b}} = \overline{\underbrace{\|\mathbf{a}\|\|\mathbf{b}\|}_{=1}\sin\theta\mathbf{n}} =$$

$$= \sin\theta\overline{\mathbf{n}}, \quad \overline{\mathbf{n}} = \frac{\mathbf{a}\times\mathbf{b}}{\|\mathbf{a}\times\mathbf{b}\|},$$

то

$$\mathbf{R} = \cos\theta - \sin\theta\overline{\mathbf{n}}.$$

Такой ротор повернет вектор на угол 2θ. Если нужен поворот на угол θ, то нужно взять половинный угол:

$$\mathbf{R} = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}\overline{\mathbf{n}}.$$

5.2. Сравнение с кватернионами

Применение мультивекторов не дает очевидного вычислительного выигрыша по сравнению с

кватернионами. Однако можно выделить ряд концептуальных преимуществ

• Ротор $\mathbf{R}$ и операция $\mathbf{RvR}^{-1}$ задают вращение в пространстве любой размерности.

• Операция $\mathbf{RvR}^{-1}$ находит геометрическую интерпретацию в виде двойного отражения. В случае кватернионов операция $q\mathbf{v}q^*$ вводится ad hoc.

• Угол θ имеет геометрический смысл угла между $\mathbf{a}$ и $\mathbf{b}$. В случае кватернионов геометрическая интерпретация данного угла затруднительна.

6. МОДУЛЬ GRASSMANN.JL ДЛЯ ЯЗЫКА JULIA

6.1. Базовые возможности модуля Grassmann.jl

Модуль Grassmann.jl [17, 18] предназначен для языка программирования Julia. В нем реализованы все базовые операции геометрической алгебры. Одной из особенностей является поддержка смешанных численно-символьных вычислений: основные вычисления выполняются численно, но при подключении модуля Reduce появляется возможность задавать символьные компоненты и параметры.

Рассмотрим вначале базовый функционал данного модуля на примерах, повторяющих примеры изложенные нами выше. Укажем также на ошибки с которыми мы столкнулись в ходе изучения данного пакета. Наши замечания справедливы для версии библиотеки |0.7.5|. Рассмотрим следующий фрагмент кода.

```
1 G2 = L(ℝ^2) # DirectSum.Basis{⟨++⟩,4}(v, v₁,
  ↪ v₂, v₁₂)
2 @basis ℝ^2
3 complementright(e1), complementright(e2) #
  ↪ (1v₂, -1v₁)
4 complementleft(e1), complementleft(e2) #
  ↪ (-1v₂, 1v₁)
5 using Reduce
6 u = :u1 * e1 + :u2 * e2
7 complementright(u) # Error
8 u = 1*e1 + 2*e2
9 complementright(u) # -2v₁ + 1v₂
10 complementleft(u) # 2v₁ - 1v₂
```

Работа начинается с задания размерности пространства L, после чего генерируются все возможные p-векторы для пространства $\Lambda(L)$. В первой строке представлен вариант задания двумерного пространства $\Lambda(L) = \mathbb{R} \oplus L \oplus \Lambda^2(L)$ без экспорта сгенерированных базисных векторов в общее пространство имен, а во второй строке с экспортом.

Отметим, что для исходного кода на языке Julia общепринятым является использование различных специфических символов в кодировке Unicode. Для ввода тех символов, которые обычно используются в математике можно применять макросы $\LaTeX$. Так, например, для ввода буквы Λ в Jupyter Notebook или во встроенной оболочке, следует набрать `\Lambda` и нажать клавишу `Tab`. Некоторые макросы определены непосредственно в подключаемых пакетах. В нашем примере букву $\mathbb{R}$ можно ввести с помощью макроса `/bbR`, о чем, между прочим, ничего не сказано в официальной документации пакета Grassmann.

Далее (строки 3 и 4) мы применяем операции правого и левого дополнения. Вначале используется смешанный численно-символьный вариант, а затем делается попытка использовать чисто символьный вариант, однако она приводит к неудаче. Опытным путем авторами было установлено, что кроме операции внешнего произведения все остальные операции символьные вычисления не поддерживают. Ниже приведем пример вычисления ориентированной площади $S = \overline{\mathbf{u} \wedge \mathbf{w}}$ в символьном виде.

```
1 u = :u1 * e1 + :u2 * e2
2 w = :w1 * e1 + :w2 * e2
3 u ∧ w # (u1 * w2 - u2 * w1)v₁₂
4 # u = (1, 2) w = (2, 1)
5 complementright(u∧w), complementleft(u∧w) #
  ↪ -3v
6 complementright(w∧u), complementleft(w∧u) #
  ↪ +3v
```

Ниже показаны примеры вычисления векторного и смешанного произведений с помощью внешнего произведения и правого дополнения: $\mathbf{u} \times \mathbf{w} = \overline{\mathbf{u} \wedge \mathbf{w}}$ и $(u, v, w) = \overline{\mathbf{u} \wedge \mathbf{v} \wedge \mathbf{w}}$.

```
1 @basis ℝ^3
2 u∧w # (u1 * w2 - u2 * w1)v₁₂ + (u1 * w3 - u3
  ↪ * w1)v₁₃ + (u2 * w3 - u3 * w2)v₂₃
3 complementright(u∧w) # Error!
4 complementright.((e12, e13, e23)) # (1v₃,
  ↪ -1v₂, 1v₁)
```

Операция * переопределена для геометрического умножения. Можно перемножать мультивекторы, разложенные по базисным p-векторам, как это показано в фрагменте кода ниже.

```
1 (3 + 5*e1*e2 - e1*e3) * (4*e1*e2*e3) # - 4v₂
  ↪ - 20v₃ + 12v₁₂₃
```

```
2 e1*e2*e1*e2*e3 # -1v3
3 e1*e3*e1*e2*e3 # 1v2
```

Относительно операции геометрического произведения множество мультивекторов вида $\mathbf{V} = a + b\mathbf{e}_{12}$ изоморфно множеству комплексных чисел в случае двумерного пространства L. Данные мультивекторы состоят только из скалярной и бивекторной части, а векторная часть равна нулю. Продемонстрируем этот факт на следующем примере, сравнив действия с мультивекторами с встроенным в язык Julia типом данных комплексных чисел.

```
1 (1e + 2*e12) * (2e + e12) # 0 + 5v12
2 (1 + 2im)*(2 + im) # 0+5im
3 (1.0e + 2*e12) / (2.0e + e12) # 0.8
+ 0.6v12
4 (1 + 2im) / (2 + im) # 0.8 + 0.6im
5     sqrt(1.0e     +     2*e12)     #
1.2720196542002786 +
  ↪ 0.7861513560627762v12
6 sqrt(1 + 2im) # 1.272019649514069 +
  ↪ 0.7861513777574233im
```

Отметим, что в выражении `sqrt(1.0e + 2*e12)` нам пришлось указать 1.0 вместо 1, так как иначе возникает ошибка извлечение квадратного корня из целого числа, появляющаяся из-за отсутствия явного приведения типов.

Стоит повысить размерность пространства L до 3, как мультивекторы того же вида $\mathbf{V} = a + b\mathbf{e}_{12} + c\mathbf{e}_{23} + d\mathbf{e}_{13}$ становятся изоморфными относительно операции геометрического произведения множеству кватернионов. Бивекторная часть в трехмерном пространстве имеет уже три компонента. Следующий код позволяет распечатать таблицу умножения мнимых единиц кватернионов.

```
1 using Printf
2 @basis ℝ^3 L e
3 i = e12; j = e23; k = e13
4 for x in (1e, i, j, k)
5  @printf "%6s|%6s|%6s|%6s\n" x*1e
x*i x*j
   ↪ x*k
6 end
7 i*j*k # -1v
```

6.2. Реализация роторов

Рассмотрим конкретный пример вращения вектора [16]. При задании вращения в трехмерном пространстве вначале следует определить ось вращения. Для примера возьмем в качестве такой оси нормированный вектор $\mathbf{u} = \frac{1}{\sqrt{3}}(1, 1, 1)$ вокруг которого на угол $\frac{2\pi}{3}$ поворачивается базисный вектор (орт) $\mathbf{e}_1$. Из геометрических соображений понятно, что результатом такого вращения будет другой орт $\mathbf{e}_2$.

Если необходимо задать вращение в терминах мультивекторов, то следует задавать не ось вращения, а плоскость, параллельно которой данное вращение происходит. Данная плоскость определяется некоторым бивектором вида $\mathbf{U} = \mathbf{a} \wedge \mathbf{b}$. Чтобы вычислить $\mathbf{U}$ следует взять правое дополнение от оси вращения $\mathbf{u}$:

$$\mathbf{U} = \bar{\mathbf{u}}.$$

После чего можно вычислить ротор, используя его тригонометрическую форму:

$$\mathbf{R} = \cos\frac{\pi}{3} - \sin\frac{\pi}{3}\mathbf{U}$$

Далее с помощью этого ротора производится вращение:

$$\mathbf{ReR}^{-1} = \mathbf{ReR}^{*}.$$

Продемонстрируем вычисление данного примера с помощью пакета Grassmann. Оператор ~ является синонимом функции **reverse** взятия обратного элемента.

```
1 u = (v1 + v2 + v3)/sqrt(3)
2 # 0.5773502691896258v1 +
  ↪ 0.5773502691896258v2 +
  ↪ 0.5773502691896258v3
3 U = complementright(u)
4 # 0.5773502691896258v12 -
  ↪ 0.5773502691896258v13 +
  ↪ 0.5773502691896258v23
5 R = cos(π/3) - sin(π/3)*U
6 # 0.5000000000000001 - 0.5v12 + 0.5v13 -
  ↪ 0.5v23
7 R*v1*(~R)
8 # 0.0 + 1.1102230246251565e-16v1 +
1.0v2 -
  ↪ 1.1102230246251565e-16v3 -
  ↪5.551115123125783e-17v123
```

7. ЗАКЛЮЧЕНИЕ

Формализм геометрической алгебры является элегантным способом представления алгебры Клиффорда. При всей своей простоте она еще не стала привычным инструментом научного исследования. И нам представляется, что именно использование аналитико-численного подхода позволит внедрить данный формализм в практику научных исследований.

8. БЛАГОДАРНОСТИ

Публикация выполнена при поддержке Программы стратегического академического лидерства РУДН (Геворкян М.Н., Демидова А.В.), при поддержке гранта Российского научного фонда № 19-71-30008 (Велиева Т.Р.), при финансовой поддержке РФФИ в рамках научного проекта № 19-01-00645 (Королькова А.В., Кулябов Д.С.).

СПИСОК ЛИТЕРАТУРЫ

1. *Reed M.E.* Differential geometric algebra with leibniz and grassmann // Proceedings of JuliaCon, 2019. P. 1–6.
2. *Кострикин А.И.* Линейная алгебра. М.: МЦНМО, 2009. Т. 2. 368 с.
3. *Dorst L., Fontijne D., Mann S.* Geometric algebra for computer science (with errata). The Morgan Kaufmann Series in Computer Graphics. 1 ed. Morgan Kaufmann, 2007.
4. *Lengyel E.* Mathematics. Lincoln, California: Terathon Software LLC. 2016. V. 1. 195 p.
5. *Chisolm E.* Geometric algebra, 2012, arXiv: 1205.5935.
6. *Hitzer E.* Introduction to clifford's geometric algebra. 2012. V. 51. № 4. P. 338–350.
7. *Bezanson J., Edelman A., Karpinski S., Shah V.B.* Julia: A fresh approach to numerical computing // SIAM Review. 2017. V. 59. № 1. P. 65–98. arXiv: 1411.1607.
8. *Геворкян М.Н., Королькова А.В., Кулябов Д.С.* Использование шаблонизатора как инструментария компьютерной алгебры // Программирование. 2021. № 1. С. 25–34.
9. *Bezanson J., Karpinski S., Shah V.B., Edelman A.* Julia: A fast dynamic language for technical computing. 2012. P. 1–27. arXiv: 1209.5145.
10. *Gevorkyan M.N., Demidova A.V., Korolkova A.V., Kulyabov D.S.* Statistically significant performance testing of julia scientific programming language // Journal of Physics: Conference Series. 2019. V. 1205. № 1. P. 012017.
11. *Gevorkyan M.N., Korolkova A.V., Kulyabov D.S., Lovetskiy K.P.* Statistically significant comparative performance testing of julia and fortran languages in case of runge–kutta methods // Numerical Methods and Applications. NMA 2018 / Ed. by Geno Nikolov, Natalia Kolkovska, Krassimir Georgiev. Cham: Springer International Publishing, 2019. V. 11189 of Lecture Notes in Computer Science. P. 400–407.
12. *Кулябов Д.С., Королькова А.В.* Компьютерная алгебра на julia // Программирование. 2021. № 2. С. 44–50.
13. *Browne J.* Grassmann Algebra. CreateSpace Independent Publishing Platform, 2012. V. 1. 588 p.
14. *Eastwood M.G., Michor P.* Some remarks on the pluecker relations. 2000. V. 63. P. 85–88.
15. *Казанова Г.* Векторная алгебра / Под ред. М.К. Поливанов. Современная математика. М.: Мир, 1979. 120 с.
16. *Kuipers J.B.* Quaternions And Rotation Sequences. Princeton, New Jersey: Princeton University Press, 2002. 391 p.
17. *Reed M.E.* Grassmann.jl documentation site. 2021. https://grassmann.crucialflow.com/stable/.
18. *Reed M.E.* Grassmann.jl code. 2021. https://github.com/chakravala/Grassmann.jl.

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 65–70

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 519.85

О ВЫЧИСЛЕНИИ РЕЗУЛЬТАНТА МНОГОЧЛЕНА И ЦЕЛОЙ ФУНКЦИИ

© 2022 г. В. И. Кузоватов[a,*], А. А. Кытманов[a,], Е. К. Мышкина[b,***]**

[a] *Сибирский федеральный университет, пр. Свободный, 79, 660041 Красноярск, Россия*

[b] *Институт вычислительного моделирования СО РАН, Академгородок, 50/44, 660036 Красноярск, Россия*

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

***E-mail: aakytm@gmail.com*

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

Поступила в редакцию 02.08.2021 г.
После доработки 02.09.2021 г.
Принята к публикации 16.09.2021 г.

На основе рекуррентных формул Ньютона на комплексной плоскости найдены суммы значений одной целой функции в нулях другой целой функции. Это позволяет ответить на вопрос, имеют ли эти функции общие нули или нет. Приведен алгоритм вычисления приближения к результанту многочлена (или целой функции с конечным числом нулей) и целой функции. Алгоритм реализован в системе компьютерной алгебры Maple. Приведены примеры, демонстрирующие работу данного алгоритма.

DOI: 10.31857/S013234742201006X

1. ВВЕДЕНИЕ

Одним из наиболее значимых типов задач, решаемых при помощи универсальных систем компьютерной алгебры, являются задачи по вычислениям с полиномами от нескольких переменных над различными полями и кольцами (разложения на множители, вычисление дискриминантов и результантов систем уравнений, базисов Гребнера и т.п.), а также решения систем нелинейных алгебраических уравнений и других систем, сводящихся к ним подстановками элементарных функций.

В то же время, аппарат исследования систем неалгебраических уравнений находится еще на начальной стадии развития. Одним из инструментов для исследования таких систем является формула многомерного логарифмического вычета, на основе которой был создан модифицированный метод исключения неизвестных из алгебраических систем, предложенный в [1] и развитый в [2]. Однако, для систем неалгебраических уравнений (содержащих, например, голоморфные функции) такие разработки отсутствовали.

Вместе с тем, неалгебраические системы уравнений возникают в различных областях знания. В частности, в процессах, описываемых системами дифференциальных уравнений с правыми частями, разложимыми в ряд Тейлора, актуален вопрос об определении числа стационарных состояний в множествах определенного вида (и их локализации). Эта проблема приводит к задачам построения алгоритмов для определения числа корней заданной системы уравнений в различных множествах, определения самих корней, исключения части неизвестных из системы. В частности, в монографиях [2, 3] приведены многочисленные примеры из химической кинетики, где требуются алгоритмы исключения неизвестных.

Одним из методов исключения неизвестных служит построение результанта двух целых функций. Хорошо известен классический результант Сильвестра для двух многочленов и метод исключения неизвестных, на нем основанный. Для неалгебраических функций такое понятие не было изучено ранее. Лишь в последние годы в работах [4, 5] обсуждается один подход к нахождению результанта двух целых функций, основанный на рекуррентных формулах Ньютона.

Разработанный в данной работе алгоритм может быть также использован при исследовании дзета-функции корней некоторых классов целых функций, которые являются важным инструментом в создании методов исключения неизвестных из систем нелинейных уравнений, как было показано в работах [6, 7].

2. КЛАССИЧЕСКИЙ РЕЗУЛЬТАНТ СИЛЬВЕСТРА И ЕГО ОБОБЩЕНИЯ

Напомним, что для данных многочленов f и g классический результант $R(f,g)$ может быть определен различными способами (см. [8], [9]) с использованием определителя Сильвестра, способа Безу–Кэли или формулы для произведения

$$R(f,g)=\prod_{\{x:f(x)=0\}} g(x). \tag{1}$$

В данной работе мы рассматриваем построение результанта двух целых функций $f(z)$ и $g(z)$, $z\in\mathbb{C}$, с помощью (1). Выбор формулы произведения объясняется тем, что целые функции являются естественным обобщением многочленов в комплексном анализе.

В работах [10]– [14] были предложены обобщения понятия результанта для аналитических функций в кольце матричнозначных функций, мероморфных функций на римановой поверхности, для систем алгебраических уравнений. Во всех этих исследованиях предполагалось, что число нулей или полюсов конечно. Наш случай существенно отличается тем, что целые функции могут иметь бесконечное число нулей. Поэтому для нахождения результанта необходим предельный переход (по степени n многочлена $g(z)$ при фиксированном количестве нулей многочлена $f(z)$).

Первым шагом в нахождении результанта двух целых функций явилась работа [4], в которой рассмотрен случай, когда одна из функций – целая, а вторая является многочленом (или целой функцией с конечным числом нулей). В работе [15] обобщены результаты из [4] в случае, когда одна из целых функций удовлетворяет некоторым жестким ограничениям, но все-таки может иметь бесконечное число нулей.

Преимущество нашего подхода состоит в том, что он позволяет ответить на вопрос, имеют ли целые функции общие нули или нет, не вычисляя самих нулей. Итоговая формула для результанта содержит степенные суммы корней, которые можно вычислить по формулам Ньютона, не прибегая к нахождению самих нулей.

3. РЕЗУЛЬТАНТ ДВУХ ЦЕЛЫХ ФУНКЦИЙ

Рассмотрим систему уравнений, состоящую из двух многочленов $f(z)$ и $g(z)$ степеней m и n соответственно (случай $m=2$ рассмотрен в [5]):

$$\begin{cases} f(z)=a_0+a_1z+\ldots+a_{m-1}z^{m-1}+z^m,\\ g(z)=b_0+b_1z+\ldots+b_nz^n. \end{cases} \tag{2}$$

Напомним, что классические рекуррентные формулы Ньютона связывают между собой коэффициенты многочлена и степенные суммы его корней. А именно, пусть

$$P(z)=z^m+c_1z^{m-1}+\ldots+c_{m-1}z+c_m.$$

Тогда если $z_1,\ldots,z_m$ – его корни (в том числе и кратные), то степенные суммы S_k (определяемые как $S_0=m$, $S_k=z_1^k+\ldots+z_m^k$, $k\in\mathbb{N}$) и коэффициенты c_j связаны соотношениями:

$$\begin{cases} S_j+\sum\limits_{i=1}^{j-1} S_{j-i}c_i+jc_j=0, \quad 1\le j\le m,\\ S_j+\sum\limits_{i=1}^{m} S_{j-i}c_i=0, \quad j>m. \end{cases} \tag{3}$$

Обозначим нули многочлена $f(z)$ через z_1, ..., z_m. Везде далее под S_j понимаются степенные суммы корней многочлена $f(z)$, которые определяются формулами (3). Используя определение результанта в виде формулы произведения, в [16] был получен следующий результат.

Теорема 1 ([16]). *Результант $R(f,g)$ системы многочленов вида* (2), *где $m=3$, вычисляется по формуле*

$$\begin{aligned} R(f,g)=&\sum_{k=0}^{n} b_k^3(-a_0)^k+\sum_{s=0}^{n}\sum_{t=s+1}^{n}(-a_0)^s\times\\ &\times\left(\frac{1}{2}b_sb_t^2(S_{t-s}^2-S_{2t-2s})+b_s^2b_tS_{t-s}\right)+\\ &+\sum_{s=0}^{n}\sum_{t=s+1}^{n}\sum_{p=t+1}^{n} b_sb_tb_p(-a_0)^s\times\\ &\times[S_{t-s}\cdot S_{p-s}-S_{t+p-2s}]. \end{aligned} \tag{4}$$

В работе [16] приводится также общий результат для произвольных степеней m и n многочленов из (2). Однако, следует отметить, что приведенная в этой работе формула для результанта двух многочленов [16, Теорема 3, формула 7] не является полностью конструктивной. Алгоритмизация данного результата является открытой задачей и, по всей видимости, может быть решена с помощью описания рекуррентного перехода по степени m многочлена f. Тем не менее, используя данный результат, как частный случай может быть получена формула для степени $m=4$, которую мы приводим ниже.

Теорема 2. *Результант $R(f,g)$ системы многочленов вида* (2), *где $m=4$, вычисляется по формуле*

$$\begin{aligned} R(f,g)=&\sum_{k=0}^{n} b_k^4a_0^k+\sum_{s=0}^{n}\sum_{t=s+1}^{n} b_s^3b_ta_0^sS_{t-s}+\\ &+\sum_{s=0}^{n}\sum_{t=s+1}^{n} b_sb_t^3a_0^s\frac{S_{t-s}^3-3S_{2t-2s}\cdot S_{t-s}+2S_{3t-3s}}{4}+ \end{aligned}$$

$$+\sum_{s=0}^{n}\sum_{t=s+1}^{n} b_s^2 b_t^2 a_0^s \frac{1}{2}(S_{t-s}^2 - S_{2t-2s}) +$$

$$+\sum_{s=0}^{n}\sum_{t=s+1}^{n}\sum_{p=t+1}^{n} b_s^2 b_t b_p a_0^s \left(S_{p-s}\cdot S_{t-s} - S_{p+t-2s}\right) +$$

$$+\sum_{s=0}^{n}\pm\sum_{t=s+1}^{n}\sum_{p=t+1}^{n} b_s b_t^2 b_p a_0^s \frac{1}{2}(S_{t-s}^2\cdot S_{p-s} -$$

$$- S_{2t-2s}\cdot S_{p-s} - 2S_{t+p-2s}\cdot S_{t-s} + 2S_{2t+p-3s}) +$$

$$+\sum_{s=0}^{n}\sum_{t=s+1}^{n}\sum_{p=t+1}^{n} b_s b_t b_p^2 a_0^s \frac{1}{2}(S_{p-s}^2\cdot S_{t-s} -$$

$$- S_{2p-2s}\cdot S_{t-s} - 2S_{t+p-2s}\cdot S_{p-s} + 2S_{2p+t-3s}) +$$

$$+\sum_{s=0}^{n}\sum_{t=s+1}^{n}\sum_{p=t+1}^{n}\sum_{r=p+1}^{n} b_s b_t b_p b_r a_0^s \times$$

$$\times\left(S_{t-s}\cdot S_{p-s}\cdot S_{r-s} - S_{t+p-2s}\cdot S_{r-s} -\right.$$

$$\left. - S_{t+r-2s}\cdot S_{p-s} - S_{p+r-2s}\cdot S_{t-s} + 2S_{t+p+r-3s}\right). \qquad (5)$$

Осуществив в (4) (соответственно, в (5)) предельный переход по n, получаем утверждение о результанте относительно многочлена (или целой функции с конечным числом нулей) и целой функции.

Теорема 3. *Пусть $g(z)$ – целая функция на комплексной плоскости $\mathbb{C}$ вида*

$$g(z) = b_0 + b_1 z + b_2 z^2 + \ldots + b_n z^n + \ldots,$$

а $f(z)$ – многочлен вида (2) *при $m = 3$ (или, соответственно, при $m = 4$). Тогда результантом $R(f, g)$ является выражение* (4) (*соответственно, выражение* (5)), *в котором необходимо выполнить предельный переход по n.*

Замечание 1 ([16]). *Теорема 3 имеет смысл при условии, что ряды, получающиеся при предельном переходе по n в правых частях формул* (4) *и* (5), *абсолютно сходятся.*

4. ОПИСАНИЕ АЛГОРИТМА

Алгоритм состоит из функции, реализующей вычисление степенных сумм с помощью классических формул Ньютона (3) и основной функции, вычисляющей результант системы (2) по формулам (4) и (5). Таким образом, для заданного многочлена $f(z)$ 3-й или 4-й степени и заданной целой функции $g(z)$, алгоритм вычисляет результант $f(z)$ и частичной суммы порядка n (разложения в ряд) функции $g(z)$ для произвольного n. Таким образом, условие остановки алгоритма при предельном переходе в формулах (4) и (5) определяется тем, что в вычислениях целая функция заменяется многочленом степени n, при этом степень n является произвольной и задается пользователем (см. пример 2).

5. ПРИМЕРЫ

Алгоритм был реализован в среде Maple 2016 64bit. Полный код программы доступен по адресу `https://github.com/aakytmanov/Resultant34n`. Вычисления производились на машине Intel Core i7-4790 (3.6 GHz) с 32 Gb RAM под управлением Windows 10 Pro x64 21H1. Время счета для приведенных примеров составило менее 0.2 секунды.

Пример 1. Рассмотрим систему уравнений ($m = 4$, $n = 5$)

$$\begin{cases} f(z) = z^2(z-1)(z+1) = z^2(z^2-1) = z^4 - z^2, \\ g(z) = z^5 + 2. \end{cases}$$

Алгоритм 1: Алгоритм вычисления классических рекуррентных формул Ньютона

Function ***RecurrNewton***$(\{c_1, \ldots, c_m\}, k)$:

Input: Список коэффициентов $c_1, \ldots, c_m$; натуральное число k.

Output: Список $S_1, \ldots, S_k$ степенных сумм, вычисляющихся по формулам (3)

begin:

$S_1 := -c_1$

for $j = 2 \ldots \min(m, k)$ **do**

$S_j := -\sum_{i=1}^{j-1} S_{j-i} c_i - j c_j$

if $k > m$ then

for $j = m+1 \ldots k$ do

$S_j = -\sum_{i=1}^{m} S_{j-i} c_i$

return $\{S_1, \ldots, S_k\}$

В данном случае

$$a_0 = a_1 = a_3 = 0, \quad a_2 = -1,$$
$$b_0 = 2, \quad b_1 = b_2 = b_3 = b_4 = 0, \quad b_5 = 1.$$

Поскольку $a_0 = 0$, то в выражении (5) останутся лишь слагаемые, соответствующие $s = 0$. Также отметим, что поскольку только b_0 и b_5 отличны от нуля, то в соответствующих суммах останутся лишь слагаемые при $t = 5$. Таким образом, получим

$$\sum_{s=0}^{5} b_s^4 a_0^s = b_0^4 = 16,$$

$$\sum_{s=0}^{5}\sum_{t=s+1}^{5} b_s^3 b_t a_0^s S_{t-s} = \sum_{t=1}^{5} b_0^3 b_t S_t = b_0^3 b_5 S_5 = 8S_5,$$

$$\sum_{s=0}^{5}\sum_{t=s+1}^{5} b_s b_t^3 a_0^s \frac{1}{4}\times$$
$$\times (S_{t-s}^3 - 3S_{2t-2s}\cdot S_{t-s} + 2S_{3t-3s}) =$$
$$= \sum_{t=1}^{5} b_0 b_t^3 \frac{1}{4}(S_t^3 - 3S_{2t}\cdot S_t + 2S_{3t}) =$$
$$= \frac{1}{4} b_0 b_5^3 (S_5^3 - 3S_{10}\cdot S_5 + 2S_{15}) =$$
$$= \frac{1}{2}(S_5^3 - 3S_{10}\cdot S_5 + 2S_{15}),$$

$$\sum_{s=0}^{5}\sum_{t=s+1}^{5} b_s^2 b_t^2 a_0^s \frac{1}{2}(S_{t-s}^2 - S_{2t-2s}) =$$
$$= \sum_{t=1}^{5} \frac{1}{2} b_0^2 b_t^2 (S_t^2 - S_{2t}) = \frac{1}{2} b_0^2 b_5^2 (S_5^2 - S_{10}) =$$
$$= 2(S_5^2 - S_{10}).$$

Алгоритм 2: Алгоритм вычисления приближения к результанту многочлена 3-й степени и целой функции

Input: Список $\{a_0, a_1, a_2\}$ коэффициентов многочлена f; список $\{b_0, b_1, \ldots, b_n\}$ первых $n+1$ коэффициентов разложения целой функции g в ряд

Output: Приближение к результанту системы (2), вычисленное по формуле (4).

begin

$S :=$ **RecurrNewton**$(\{a_0, a_1, a_2\}, n)$

$R := \sum_{k=0}^{n} b_k^3 (-a_0)^k$

for $s = 0 \ldots n$ do

$$R := R + \sum_{t=s+1}^{n} (-a_0)^s \left(\frac{1}{2} b_s b_t^2 \times \right.$$
$$\left. \times (S^2[t-s] - S[2t-2s]) + b_s^2 b_t S[t-s]\right)$$

for $s = 0 \ldots n$ do

for $t = s+1 \ldots n$ do

$$R := R + \sum_{p=t+1}^{n} b_s b_t b_p (-a_0^s) \times$$
$$\times \left(S[t-s]\cdot S[p-s] - S[t+p-2s]\right)$$

return R

Все остальные тройные и четверные суммы в выражении (5) обращаются в ноль из-за того, что предыдущие суммы отличны от нуля только в случае значения индексов $s = 0$, $t = 5$.

Итак,

$$R(f,g) = 16 + 8S_5 + \frac{1}{2}S_5^3 -$$
$$- \frac{3}{2} S_{10}\cdot S_5 + S_{15} + 2S_5^2 - 2S_{10}.$$

Найдем степенные суммы корней S_5, S_{10}, S_{15}, не используя значения самих корней. А именно, по рекуррентным формулам Ньютона вида (3), в которых нужно положить

$$m = 4, \quad c_1 = a_3 = 0, \quad c_2 = a_2 = -1,$$
$$c_3 = a_1 = 0, \quad c_4 = a_0 = 0.$$

Алгоритм 3: Алгоритм вычисления приближения к результанту многочлена 4-й степени и целой функции

Input: Список $\{a_0, a_1, a_2, a_3\}$ коэффициентов многочлена f; список $\{b_0, b_1, \ldots, b_n\}$ первых $n+1$ коэффициентов разложения целой функции g в ряд

Output: Приближение к результанту системы (2), вычисленное по формуле (5).

begin

$S :=$ **RecurrNewton**$(\{a_0, a_1, a_2, a_3\}, n)$

$R := \sum_{k=0}^{n} b_k^4 a_0^k$

for $s = 0 \ldots n$ **do**

$$R := R + \sum_{t=s+1}^{n} \left(b_s^3 b_t a_0^s S[t-s] + \right.$$
$$+ \frac{b_s b_t^3 a_0^s}{4}\cdot (S^3[t-s] - 3S[2t-2s]\times$$
$$\left. \times S[t-s] + 2S[3t-3s]) + \frac{b_s^2 b_t^2 a_0^s}{2}(S^2[t-s] - S[2t-2s])\right)$$

for $s = 0 \ldots n$ **do**

for $t = s+1 \ldots n$ **do**

$$R := R + \sum_{p=t+1}^{n} b_s b_t b_p a_0^s \left(b_s\,(S[p-s]\cdot S[t-s] - \right.$$
$$- S[p+t-2s]) + \frac{b_t}{2}(S^2[t-s]\cdot S[p-s] -$$
$$- S[2t-2s]S[p-s] - 2S[t+p-2s]\cdot S[t-s] +$$
$$+ 2S[2t+p-3s]) + \frac{b_p}{2}(S^2[p-s]S[t-s] -$$
$$- S[2p-2s]\cdot S[t-s] -$$
$$\left. - 2S[t+p-2s]\cdot S[p-s] + 2S[2p+t-3s])\right)$$

for $s = 0 \ldots n$ **do**

  for $t = s + 1 \ldots n$ **do**

    for $p = t + 1 \ldots n$ **do**

$$R := R + \sum_{r=p+1}^{n} b_s b_t b_p b_r a_0^s (S[t-s] \cdot S[p-s] \cdot \cdot S[r-s] - S[t+p-2s] \times \times S[r-s] - S[t+r-2s] \cdot S[p-s] - - S[p+r-2s] \cdot S[t-s] + 2S[t+p+r-3s])$$

return R

Так, для нашего случая

$$S_5 = 0, \quad S_{10} = 2, \quad S_{15} = 0.$$

Таким образом, $R(f,g) = 12$.

Входными данными алгоритма в этом случае будут списки коэффициентов a_i и b_j:

```
> Res4d([0, 0, -1, 0], [2, 0, 0, 0, 0, 1]);
```

Пример 2. Данные методы приводят к вычислению сумм некоторых типов кратных числовых рядов, ранее отсутствовавших в известных справочниках. Эти результаты представляют самостоятельный интерес. Рассмотрим систему уравнений

$$\begin{cases} f(z) = z^3 - a^3, \\ g(z) = e^{bz} = 1 + bz + \dfrac{(bz)^2}{2!} + \ldots + \dfrac{(bz)^n}{n!} + \ldots. \end{cases}$$

Вычисляя результант $R(f,g)$, используя теорему 3 с одной стороны, и используя определение результанта в виде формулы для произведения с другой стороны, получим соотношение (см. 16])

$$\sum_{k=0}^{\infty} \frac{(ab)^{3k}}{(k!)^3} + \sum_{s=0}^{\infty}\sum_{j=1}^{\infty} a^{3s} \times \times \left(3\frac{b^{3s+6j}}{s!(s+3j)!^2} a^{6j} + 3\frac{b^{3s+3j}}{s!^2(s+3j)!} a^{3j} \right) + + \sum_{s=0}^{\infty}\sum_{t=s+1}^{\infty}\sum_{p=t+1}^{\infty} b_s b_t b_p a^{3s} \times \times [S_{t-s} \cdot S_{p-s} - S_{t+p-2s}] = 1.$$

Частичные суммы выражения в левой части могут быть получены с помощью предложенного алгоритма, например, используя входные данные

```
> simplify(Res3d([-a^3, 0, 0], [1, b, b^2/2!, b^3/3!, b^4/4!, b^5/5!, b^6/6!]));
```

получим приближение к результанту (для заданной системы функций) в виде

$$R(f,g) = 1 - \frac{a^9 b^9}{4320} + \frac{a^{12} b^{12}}{345600} + + \frac{a^{15} b^{15}}{10368000} + \frac{a^{18} b^{18}}{373248000}.$$

6. БЛАГОДАРНОСТИ

Исследования, представленные в работе, были выполнены при поддержке следующих фондов: первый автор использовал финансовую поддержку РФФИ, Правительства Красноярского края и Красноярского краевого фонда науки, грант 20-41-243002 (разработка алгоритма, тестирование); второй автор использовал поддержку гранта РНФ 18-71-10007 (программная реализация алгоритма); третий автор поддержан грантом РФФИ 19-31-60012 (постановка технического задания для разработки алгоритма, вычисление примеров).

СПИСОК ЛИТЕРАТУРЫ

1. *Айзенберг Л. А.* Об одной формуле обобщенного многомерного логарифмического вычета и решении систем нелинейных уравнений // Докл. АН СССР. 1977. Т. 234. № 3. С. 505–508.
2. *Bykov V.I., Kytmanov A.M., Lazman M.Z.* Elimination methods in polynomial computer algebra, Kluwer Academic Publishers, Dodrecht-Boston-Basel, 1998.
3. *Быков В.И., Цыбенова С.Б.* Нелинейные модели химической кинетики. М.: КРАСАНД, 2011.
4. *Kytmanov A.M., Naprienko Ya.M.* An approach to define the resultant of two entire functions // Journal Complex Variables and Elliptic Equations. 2017. V. 62. № 2. P. 269–286.
5. *Kytmanov A.M., Myshkina E.K.* On Some Approach for Finding the Resultant of Two Entire Functions // J. Siberian Federal Univ. Math. Phys. 2019. V. 12. № 4. P. 434–438.
6. *Kytmanov A.M., Myslivets S.G.* On the Zeta-Function of Systems of Nonlinear Equations // Siberian Math. J. 2007. V. 48. № 5. P. 863–870.
7. *Kuzovatov V.I., Kytmanov A.A.* On the Zeta-Function of Zeros of Some Class of Entire Functions // J. Siberian Federal Univ. Math. Phys. 2014. V. 7. № 4. P. 489–499.
8. *Бурбаки Н.* Алгебра. Многочлены и поля, упорядоченные группы. М.: Наука, 1965.
9. *Krein M.G., Naimark M.A.* The Method of Symmetric and Hermitian Forms in the Theory of the Separation of the Roots of Algebraic Equation // Linear and Multilinear Algebra. 1981. V. 10. № 4. P. 265–308.
10. *Gohberg I.C., Heinig G.* Resultant Matrix and its Generalization. I. Resultant Operator of Matrix Polynomial // Acta Sci. Math. 1975. V. 72. P. 41–61.

11. *Gohberg I.C., Heinig G.* Resultant Matrix and its Generalization. II. Continual Analog of Resultant Matrix // Acta Math. Acad. Sci. Hungar. 1976. V. 28. P. 189–209.
12. *Gohberg I.C., Lerer L.E.* Resultant Operators of a Pair of Analytic Functions // Proc. Amer. Math. Soc. 1978. V. 72. № 1. P. 65–73.
13. *Gustafsson B., Tkachev V.G.* The Resultant on Compact Riemann Surfaces // Comm. Math. Physics. 2009. V. 10. P. 265–308.
14. *Morozov A.Yu., Shakirov Sh.R.* New and Old Results in Resultant Theory // Theor. Math. Phys. 2010. V. 163. № 2. P. 587–617.
15. *Kytmanov A.M., Khodos O.V.* An Approach to the Determination of the Resultant of Two Entire Functions // Russian Mathematics. 2018. V. 62. № 4. P. 42–51.
16. *Kytmanov A.M., Myshkina E.K.* On Finding the Resultant of Two Entire Functions // Probl. Anal. Issues Anal. 2020. V. 9. № 3. P. 119–130.

ПРОГРАММИРОВАНИЕ, 2022, № 1, с. 71–72

Авторский указатель статей, опубликованных в 2021 году

DOI: 10.31857/S0132347422010083