Федеральное государственное автономное образовательное учреждение
высшего образования «Российский университет дружбы народов
имени Патриса Лумумбы» «РУДН»

На правах рукописи

Мамонов Антон Алексеевич

Инволютивные алгоритмы вычисления базисов Грёбнера в приложении к исследованию модели Лотки–Вольтерры

Специальность 1.2.2
Математическое моделирование, численные методы и комплексы программ

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук, доцент
Салпагаров Солтан Исмаилович

Москва — 2026

Оглавление

Стр.

Стр.

Стр.

Введение

Характерной чертой современных исследований в области математического моделирования является внимание к явлениям, обусловленным их нелинейностью. Одним из самых красивых примеров нелинейных осцилляторов является многовидовая модель конкуренции и миграции типа Вольтерры-Лотки, описывающая эволюцию нескольких взаимодействующих популяций с учётом их миграции между ареалами. Анализ режимов такой системы – поиск точек равновесия, исследование их устойчивости, отыскание периодических траекторий – сводится к решению систем нелинейных алгебраических уравнений. Эти исследования естественным образом упираются в необходимость быстро, точно и без потери корней решать системы нелинейных алгебраических уравнений. Возникающие при решении нелинейных систем трудности зачастую закрывают весьма перспективные с теоретической точки зрения подходы, вносят значительные погрешности, которые не дают возможности понять, какие явления обусловлены нелинейностью модели, а какие – погрешностями при решении систем нелинейных уравнений.

Одним из характерных примеров таких систем служат многовидовые модели конкуренции и миграции типа Вольтерра. В частности, в работах [1] рассматриваются модели вида «k конкурентов - k областей миграции», описываемые нелинейными системами обыкновенных дифференциальных уравнений. Подобные системы допускают несколько стационарных режимов – сосуществование популяций либо вытеснение части из них; вблизи точек равновесия наблюдаются как затухающие переходные процессы, так и периодические колебания. Отыскание стационарных режимов и исследование их устойчивости сводится к решению систем нелинейных алгебраических уравнений.

Фундаментальная и, вероятно, неразрешимая проблема состоит в том, что не существует универсального численного метода решения систем нелинейных уравнений с несколькими неизвестными, которые бы не теряли корни, не добавляли новых, правильно учитывали кратности, разрешали близкие корни и т.п. Именно так сформулировали ее авторы знаменитых «Численных рецептов». Такое положение дел требует создания по возможности универсального метода для решения систем нелинейных уравнений без описанных недостатков. Предпосылка к этому решению уже существует — это сочетание численных и

символьных алгоритмов. По существу в современных системах компьютерной алгебры, напр., Sage, Maple, Mathematica, реализована схема решения систем нелинейных уравнений, имеющих конечное число решений, которую, несколько огрубляя, можно описать так:

1. от системы уравнений переходят к порожденному ей идеалу и в символьном виде находят его базис Гребнера в лексикографическом порядке,
2. по последнему элементу базиса Гребнера численно находят все значения для самой младшей переменной, подставляют найденные значения в базис Гребнера, затем обычно из предпоследнего уравнения обычно удается найти предпоследнюю переменную и т.д.

Нетрудно заметить, что эти шаги представляют собой обобщения прямого и обратного прохода метода Гаусса решения системы линейных уравнений с той существенной поправкой, что прямой ход делается аналитически, а обратный — численно. Следует заметить, что численно решается уравнение с одной неизвестной, а эта задача хорошо проработана.

Объектом исследования данной диссертационной работы являются популяционная динамика, нелинейные математические модели, описывающие взаимодействие популяций, их стационарные и периодические режимы.

Предметом исследования этой работы являются методы, алгоритмы и программные средства для исследования нелинейных моделей популяционной динамики – нахождения их стационарных и периодических режимов.

Степень разработанности темы

Со стороны математического моделирования, задача моделирования популяций рассматривалась еще восемьсот лет назад. В работах Леонардо Фибоначчи, "Трактате о счете задача о размножении кроликов положила начало знаменитым числам Фибоначчи. В работах эпохи возрождения, прирост населения рассматривали как геометрическую прогрессию, и стали предлагать различные решения по ограничению такого прироста. Вскоре были сформулированы две основные модели популяционной динамики – модель логистического роста, и модель взаимодействия видов Лотки-Вольтерра.

В дальнейшем на основе этих моделей была построена масса других, более сложных, обобщающих или же специфичных, как в рамках популяционной динамики, так и вне ее. Классическая модель хищник-жертва [2; 3], модели химических реакций [4], и другие.

В конце 20-го века моделирование популяций вышло на новый этап. Индустриализация и взрывной рост население человечества значимо сказались на природных экосистемах, что сделало задачу прогнозирования таких последствий как никогда актуальной. В то же время, развитие вычислительной техники расширило количественный и качественный инструментарий исследователей, что в том числе коснулось и моделирования популяций. Объединение этих факторов вылилось в возникновение и интенсивное развитие имитационного моделирования [5].

Со временем, однако, энтузиазм нового направления был исчерпан. Имитационная модель, по определению стремящаяся имитировать конкретный объект, при переносе на другой объект, даже родственной природы, часто требует внесения значительных изменений. Ограниченные возможности имитационного моделирования подтолкнули сообщество экологов к разумному скептицизму и возрождению интереса к классическому математическому моделированию.

Однако на текущий момент все хорошо изученные математические модели имеют одну общую черту – они являются либо линейными, либо нелинейны лишь в малой степени. Причины тому очевидны – отсутствие способа решать описывающие их системы уравнений. В то же время, масса научно- и практически значимых задач лежит именно в рамках многомерных нелинейных задач – реальные ареалы обитания насчитывают десятки видов, в то время как современные инструменты позволяют уверенно моделировать только однозначное число видов.

Задача об отыскании базиса Гребнера заданного идеала была сформулирована и решена алгоритмически Бухбергером в 1970-х годах. В настоящее время строго доказано, что алгоритм Бухбергера позволяет найти базис Гребнера за конечное число шагов, однако оценка его вычислительной сложности является экспоненциальной [6]. На практике на современном компьютере его удается применять только для решения систем не очень большой степени и с не более чем десятком неизвестных.

В 1990-х годах были разработаны различные улучшения алгоритма по нахождению базиса Гребнера. Многие из них являются коммерческими, и недоступны для большинства исследователей. Среди свободного программного обеспечения выгодно выделяется отечественная разработка — система GInv. В основе этой системы лежит оригинальная концепция инволютивного базиса

идеала, возникшая при сравнении алгоритма Бухбергера и методов, используемых при исследовании совместности систем дифференциальных уравнений в частных производных. Сама концепция была описана в работах Жаркова, Гредта и Блинкова [7—11]. Ей была посвящена докт. дисс. Блинкова [12], где приведены все необходимые теоремы и их доказательства, а самое современное изложение дано в [13].

Названная альтернатива алгоритму Бухбергера была реализована на компьютере Гердтом и Блинковым в ЛИТ ОИЯИ (Дубна) в 2000-х годах, что привело к созданию 1-ой версии системы GInv, написанной на C. В 2020-х годах Блинков продолжил работу над этой системой, уделяя особое внимание работе с длинной арифметикой и сборке мусора, так появилась 2-ая версия GInv, доступная в репозитории [14]. Совсем недавно появилась версия, написанная полностью на Python и доступная на том же репозитории. Параллельно с этим, в рамках этого диссертационного исследования, на базе GInv была разработана система GInvDist [50], поддерживающая дистанционное развертывание через репозиторий PyPI.

В настоящее время вычисление инволютивного базиса в реальных задачах может занимать несколько дней, требовать десятки ГБ памяти. При этом пользователь не видит никаких сообщений, системный монитор тоже ему не помогает: после быстрого занятия десятка ГБ памяти наступает долгое время затишья, когда работает 1 ядро и кажется, что все «зависло». Фактически вычисления, проводившиеся несколько дней, могут быть остановлены пользователем как бесперспективные за минуту до получения ответа. Поэтому прогнозирование времени, необходимого для завершения вычислений, является очень важной задачей. К ее решению нельзя привлекать экспоненциальные оценки, полученные в теории, но можно и нужно привлекать методы машинного обучения.

За последние два десятилетия было высказано множество гипотез о том, как ускорить вычисления инволютивного базиса и/или сэкономить ресурсы. Это привело к возникновению различных версий и форков, что требует создания удобного инструмента, позволяющего на репрезентативной базе сравнивать эти версии друг с другом. Такая база тестовых примеров была собрана ещё в 2000-х годах Гердтом и Блинковым, однако до сих пор не создано никакого инструмента для автоматического тестирования и сравнения этих версий. Поэтому совершенно необходимо разработать инструмент для автоматического

тестирования, сравнения и отбраковки таких идей. Создание такого программного обеспечения позволит в течение месяцев решить вопрос об удачности той или иной гипотезы, направленной на ускорение вычислений или экономию ресурсов.

Целью данной работы является разработка эффективного численного метода для изучения нелинейных математических моделей популяции, таких как модель Лотки-Вольтерры, на основе инволютивного алгоритма вычисления базиса Гребнера, и тестирование этого метода в виде реализации программного комплекса.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Разработать метод для исследования математических моделей популяционной динамики на основе числено-символьного решения системы нелинейных уравнений через нахождение базиса Гребнера.
2. Изучить обобщения понятия деления многочленов; построить оригинальное изложение теории инволютивного деления на основе понятия линейного деления.
3. Разработать программный инструмент для выполнения инволютивного алгоритма вычисления базиса, а также автоматического тестирования алгоритмов и сбора данных о полиномиальных идеалах..
4. Используя разработанный инструмент сравнить эффективность классического алгоритма Бухбергера (Sage) и инволютивного алгоритма (GInvDist).
5. Выполнить оригинальное исследование о нахождении стационарных решений для популяционных моделей, найденных через вычисление базиса Гребнера.
6. Выполнить оригинальное исследование о нахождении периодических решений для динамических систем, найденных по методу Кагана, также через вычисление базиса Гребнера.

Научная новизна:

1. Предложен оригинальный численно-символьный метод исследования моделей популяционной динамики.
2. На основе системы GInv создана оригинальная система GInvDist, с внесением правок в модули: poly, gb, janet; и поддержкой установки в виде библиотеки открытого кода.

3. С использованием разработанного инструментария получены оригинальные результаты для шестимерной модели "четыре конкурента, два ареала миграции".
4. Было выполнено оригинальное исследование зависимости времени вычисления для различных алгоритмов нахождения базисов Гребнера, в зависимости от различной конфигурации оборудования, программного обеспечения, и выбранных мономиальных порядков.

Практическая значимость

1. Практическая значимость проведенного исследования заключается в разработке и программной реализации алгоритма для нахождения стационарных и периодических режимов для моделей популяционной динамики. Для ряда моделей были получены результаты, соответствующие ранее полученным другими авторами, но с демонстрацией выигрыша по времени вычислений. Для шестимерной модели "четыре конкурента - два ареала миграции"получены ранее не доступные результаты.
2. Сформированная в рамках диссертации база тестов из систем нелинейных уравнений и специализированного скрипта представляет собой в достаточной степени универсальный инструмент для верификации и оценки эффективности различных программных реализаций алгоритмов нахождения базиса Гребнера. Созданные программные средства собраны в пакет GInvDist и опубликованы в открытом репозитории библиотек PyPI (Python Package Index). Подобная форма публикации позволила существенно снизить порог вхождения для использования системы и расширить круг потенциальных пользователей среди специалистов в области математического моделирования.
3. Разработанные методы предсказания времени вычислений на основе численных характеристик полиномиальных идеалов могут быть использованы для оптимизации распределения ресурсов в высокопроизводительных вычислительных системах.
4. Результаты диссертации могут быть использованы при создании учебных курсов по темам «Математическое моделирование» и «Компьютерная алгебра».

Методология и методы исследования.

В работе использовались методы математического моделирования, популяционной динамики, компьютерной алгебры. Также использовались технологии разработки программного обеспечения. Собрана и опубликована база примеров и набор данных об их решениях в разных конфигурациях, в формате JSON. Символьные и численные вычисления выполнялись в системах компьютерной алгебры GInv и Sage; созданные в рамках диссертационного исследования инструменты были объединены в пакет GInvDist.

Основные положения, выносимые на защиту:

1. Для математических моделей в популяционной динамики предложен оригинальный численно-символьный метод для изучения стационарных и периодических режимов, путем нахождения базиса Гребнера.
2. Дано обобщение понятия деления в полиномиальных идеалах в виде понятия линейного деления. На его основе представлено оригинальное изложение теории инволютивного деления в полиномиальных идеалах.
3. Реализован программный комплекс GInvDist, объединяющий систему компьютерный алгебры GInv с системой тестирования и модулем для прогнозирования времени вычисления и выбора оптимальных параметров на основе структурных характеристик полиномиального идеала.
4. На основе результатов компьютерных экспериментов выполнено сравнение эффективности классического алгоритма вычисления базиса Гребнера (алгоритм Бухбергера) в системе Sage и инволютивного алгоритма в системе GInv, показано влияние выбора мономиального порядка и типа деления на вычислительную сложность задачи. Обоснована необходимость предварительной классификации полиномиальных идеалов для оптимизации проводимых вычислений.
5. Проведено оригинальное исследование стационарных и периодических режимов многовидовой модели конкуренции и миграции типа Вольтерра с использованием разработанных инструментов. Показано, что система GInv позволяет более эффективно находить искомые режимы, по сравнению с стандартным алгоритмом Бухбергера.

Достоверность результатов диссертации опирается на проверку выдвигаемых гипотез компьютерными экспериментами. Полученные результаты были опубликованы в рецензируемых журналах. Везде, где это возможно, проводилось сравнение теоретических изысканий с результатами компьютерных

экспериментов. Результаты находятся в соответствии с результатами, полученными другими авторами.

Апробация работы. Основные результаты работы докладывались на: международной конференции по полиномиальной компьютерной алгебре, PCA, Санкт-Петербург, 2024-2025 гг.; международной конференции “Алгебра и математическая логика: теория и приложения”, Казань, 27 июня – 1 июля 2024 г.; конференции имени Трифонова «Программирование и вычислительная математика», посвященная 100-летию со дня рождения Николая Павловича Трифонова.

Личный вклад. Автор диссертации, работая в коллективе соавторов, выполнил оригинальное расширение исследования шестимерной модели "четыре конкурента - два ареала миграции"на основе работы [1], реализовал численно-символьный метод исследования моделей популяционной динамики, доказал ряд теорем о свойствах инволютивного деления, самостоятельно разработал и реализовал систему GInvDist, провел серию компьютерных экспериментов, разработал метод для предсказания времени нахождения базисов Гребнера.

Публикации. Основные результаты по теме диссертации изложены в 7 печатных изданиях, Зарегистрированы 2 программы для ЭВМ.

Объем и структура работы. Диссертация состоит из введения, 4 глав, и заключения. Полный объём диссертации составляет 131 страницу, включая 26 рисунков и 13 таблиц. Список литературы содержит 54 наименования.

Глава 1. Математическая модель популяционной динамики Лотки–Вольтерры

1.1 Моделирование популяций

1.1.1 Первые модели

Научные исследования в области моделирования и анализа популяционных систем в последние годы привлекают внимание не только биологов, но и математиков [15]. Такие модели часто описываются нелинейными динамическими системами, и в силу этого интересны и с математической точки зрения. Поиск стационарных и прочих режимов, описание бифуркаций в моделях – в общеми случае это требует проведения нетривиальных вычислений.

При этом с практической точки зрения актуальность изучения этих моделей легко обосновать множеством реальных примеров. Полученные модели используются при установлении максимальной границы добычи для рыболовных промыслов, для изучения и мониторинга эндемических и инвазивных видов, предсказания и предотвращения биологических экологических катастроф. Также подобные модели могут быть использованы для моделирования распространения микроорганизмов – паразитов, вирусов, бактерий; – а следовательно, и болезней.

С древности человечество стремилось решать задачи популяционной динамики, в том числе в математических терминах. [16] Человеку свойственно говорить о предметах, которые ему жизненно близки, а что может быть ближе, чем законы воспроизводства населения – будь то население людей, животных или растений.

Первая дошедшая до нас математическая модель динамики численности населения приведена в книге "Фибоначчи"("Liber abaci"), датированной 1202 годом, написанной крупнейшим итальянским ученым Леонардо Фибоначчи - Леонардо Пизанским (предположительно, 1170-1240)[16]. В этой книге, представляющей собой сборник арифметических и алгебраических сведений того

времени и впоследствии распространенной в виде списков по всей Европе, рассматривается следующая задача:

"Кто-то разводит кроликов в помещении, окруженном со всех сторон высокой стеной. Сколько пар крольчат рождается за один год от одной пары, если через месяц пара крольчат рожает другую пару, а крольчата рожают, начиная со второго месяца после своего рождения."

Решением задачи является ряд чисел:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 \ldots$$

(Сам Леонардо опустил первый член ряда). Два первых числа соответствуют первому и второму месяцу размножения. 12 последующих - месячному приросту поголовья кроликов. Каждый последующий ряд равен сумме двух предыдущих. Этот ряд вошел в историю как ряд Фибоначчи, а его члены - числа Фибоначчи.

Второй всемирно известной математической моделью, в основу которой положена задача о динамике численности популяции, является классическая модель неограниченного роста - геометрическая прогрессия в дискретном представлении,

$$A_{n+1} = qA_n$$

или экспонента, - в непрерывном

$$\frac{dx}{dt} = rx$$

Модель эта была предложена Мальтусом ещё в 1798 г. в его классическом труде "О росте народонаселения". Он обратил внимание на то, что численность популяции растет в геометрической прогрессии, в то время как производство продуктов питания растет в прогрессии арифметической. Из этого следовал вывод, достаточно справедливый, что со временем экспоненциальный рост популяций обгонит линейный рост продовольствия, и наступит голод.

В отношении этого важного и тревожного вывода Дарвин, изучавший популяции на практике, отметил следующее: ни одна популяция не размножается в природе до бесконечности, а следовательно, существуют факторы, препятствующие ее неограниченному размножению. Эти факторы могут быть как натуральные, так и антропогенные – нехватка продовольствия, ограниченная площадь ареала обитания, хищничество, внутривидовая и межвидовая конкуренция. Суммируясь, эти факторы замедляют скорость роста популяции, и

делают это тем сильнее, чем больше она разрастается. В конечном итоге, большинство популяций выходят на стационарный уровень.

Впервые описал модель с учетом таких факторов, ограничивающих рост популяции, Ферхюльст, в уравнении логистического роста (1848):

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right) \tag{1.1}$$

Представленное уравнение обладает двумя важными свойствами. Во-первых, при малых численность возрастает экспоненциально, что соответствует взрывоподобному росту популяций в "неограниченных"условиях, например, ряду Фибоначчи. Во-вторых, при больших x, стремящихся к K, прирост стремится к нулю. Таким образом популяция подходит к своему естественному пределу , величине, называемой емкостью популяции. Это единая величина объединяет в себе влияние ограниченности пищевых ресурсов, мест для гнездования, и многих других факторов.

Уравнение 1.1 можно решить аналитически. Решение имеет вид:

$$x(t) = \frac{Kx_0e^{rt}}{K + x_0\left(e^{rt} - 1\right)} \tag{1.2}$$

Формула описывает зависимость численности популяции от времени, и называется логистической кривой.

1.1.2 Модели взаимодействия двух популяций

Предыдущее уравнение можно также переписать в виде:

$$\frac{dx}{dt} = rx - \delta x^2 \tag{1.3}$$

Здесь δ – это коэффициент внутривидовой конкуренции (например, за пищевой ресурс, убежища, и т.п.). С одной стороны, это дает ограничение на рост популяции, подобный уже полученному в предыдущем варианте. С другой – это несколько изменяет трактовку модели и делает более лаконичным переход к моделированию сразу нескольких популяций.

Для любой популяции модель ее динамики можно описать через набор взаимодействий с окружающей средой. Для любого биологического вида, взаимодействие начинается внутри самого этого вида, между отдельными особями и группами особей (подвидами, носителями позитивных или негативных мутаций, и т.д.). Представители одного вида вынуждены как минимум делить одни и те же ресурсы, из-за чего между ними и возникает отображенная в 1.3 внутривидовая конкуренция.

Однако, очевидно, существует взаимодействие и с другими видами. Биология классифицирует разные виды таких взаимодействий. Из них наиболее распространены и интересны с точки зрения моделирования популяций следующие:

1. конкуренция – популяция обоих видов растет медленнее в присутствии друг друга;
2. симбиоз – популяция обоих видов растет быстрее в присутствии друг друга;
3. хищничество/паразитизм – популяция "хищников/паразитов"растет быстрее, а популяция "жертв/хозяев"медленнее;
4. нейтрализм – обе популяции не оказывают влияния друг на друга;
5. комменсализм – одна из популяций растет быстрее, без влияния на другую;
6. аменализм – одна из популяций растет медленнее, без влияния на другую;

Вито Вольтерра – крупнейший итальянский математик, основатель математической биологии, выполнил первое глубокое исследование закономерностей в динамике взаимодействующих популяций. В своей работе он предложил описывать взаимодействие между видами по аналогии с моделями из статистической физики и химической кинетики – в виде мультипликативных членов в уравнениях. С этим подходом, и учитывая самостоятельное ограничение из 1.1 и 1.3, для взаимодействия двух видов можно получить следующую схему:

$$\begin{aligned} \frac{dx_1}{dt} &= a_1 x_1 + b_{12} x_1 x_2 - c_1 x_1^2 \\ \frac{dx_2}{dt} &= a_2 x_2 + b_{21} x_1 x_2 - c_2 x_2^2 \end{aligned} \tag{1.4}$$

Здесь a_i – есть коэффициенты естественного прироста видов, c_i – коэффициенты внутривидовой конкуренции, b_{ij} – коэффициенты межвидового взаимодействия, для $(i,j = 1,2)$.

Для взаимодействия типа конкуренция межвидовые коэффициенты будут отрицательны $(b_{12},b_{21} < 0)$. Если же взять коэффициенты $b_{12} < 0,b_{21} > 0$, получается взаимодействие "хищничество и 1.4 превращается в классическую модель "хищник-жертва". Вынесем x_i в каждом уравнении, и сразу расставим знаки перед b_{ij}, чтобы придать ему более знакомый, канонический вид:

$$\begin{aligned} \frac{dx_1}{dt} &= x_1(a_1 - b_{12}x_2 - c_1x_1) \\ \frac{dx_2}{dt} &= x_2(a_2 + b_{21}x_1 - c_2x_2) \end{aligned} \tag{1.5}$$

Различные значения коэффициентов порождают различные поведения модели. Возможно выживание только жертвы, только хищника (если допускается другой источник питания), вымирания обоих видов, либо же сосуществование обоих видов. В последнем случае популяции совершают колебания, при этом колебания популяции хищников запаздывает по отношению к колебаниям жертв.

Адекватность такой модели, то есть ее достаточное соответствие реальному объекту исследования, многократно наблюдалась как в природе, так и на натурных экспериментах. Однако, круг задач, описываемых взаимодействием лишь двух видов, велик, но ограничен.

Более того, множество важных аспектов экологии не находят отражения в данной модели. Основная предсказательная сила модели – наличие регулярных колебаний, также не служит обязательным подтверждением, ни для одной из перечисленных моделей 1.1, 1.3. Колебания численности популяции может быть связано с природными циклами, оказывающими влияние на возможность пропитания. Хищники могут повторять эти циклы даже в том случае, если само взаимодействие их не вызывает, например, в силу эволюционной адаптации.

1.1.3 Применение моделей популяции за пределами моделирования популяций

Для любой конкретно поставленной задачи потребуется более подробно описанная модель, учитывающая специфику. Необходимо определить важные для поставленной задачи параметры, найти способ учесть их в построении модели. Тогда, при качественном подборе значений, такая модель может претендовать на адекватность.

Тем не менее, одна из высших ценностей математического моделирования, в том числе относящаяся к знаменитой формулировке о "Непостижимой эффективности математики в естественных науках в том, что математические модели, разработанные для объектов и явлений в одной предметной области, часто описывают поведения объектов и явлений в других областях.

Рассмотрим, например, реакцию Белоусова-Жаботинского[4]. Б. П. Белоусов в опытах обнаружил продолжительные автоколебания при окислении лимонной кислоты броматом в присутствии определенного катализатора – иона церия. Он заметил, что отношение концентраций трех- и четырехвалентных ионов церия колебалось во времени, что, в силу их различной цветовой окраски, давало картину пульсирующих концентрических кругов.

Сама же реакция Белоусова-Жаботинского, может быть описана в рамках дальнейшего развития модели 1.3. Несмотря на химическую сложность реакции (представляет собой набор из 80 элементарных реакций), математически модель довольно проста, особенно в сравнении с современными биологическими осцилляторами.

Зададим следующие обозначения: X_1 – концентрация автокатализатора; X_2 – концентрация $[Ce4+]$, X_3 – концентрация $[Br-]$. C – общая концентрация трех- и четырехвалентных ионов церия $[Ce4+]+[Ce3+]$. Тогда для $k_i, i = 1,2,3$ – коэффициентов реагирования, модель реакции будет следующей:

$$\frac{dX_1}{dt} = k_1 X_1 (C - X_2) - k_0 X_1 X_3$$

$$\frac{dX_2}{dt} = k_1 X_1 (C - X_2) - k_2 X_2$$

$$\frac{dX_3}{dt} = k_2 X_2 - k_3 X_3$$

В итоге мы получаем систему нелинейных уравнений, а значит – мы вновь возвращаемся к необходимости использования методов нелинейного анализа.

1.1.4 О математическом моделировании популяций

Математические модели – не только средство для описания объекта. Модель – это также и математический образ, инструмент, помогающий формализовать описание, обобщить теоретическое представление. Создание математической модели позволяет также создать язык простых терминов для описания сложной системы.

Представление нового образа мышления не в меньшей, а иногда и в большей степени представляет собой ценный результат научной деятельности, нежели количественные расчеты для отдельно взятого случая. Так и "популяционная динамика"несет с собой особый образ мышления.

Динамика популяций – это область математической биологии, которая использует модели для описания типов динамического поведения развивающихся систем, представляющих одну или несколько взаимодействующих популяций или внутрипопуляционных групп. Особенностью биологических популяций, как и всех живых систем, является их выведение из термодинамического равновесия, использование внешних источников энергии для их роста и развития. Это обусловливает необходимость описания таких систем с помощью нелинейных моделей, отражающих основные характерные особенности популяционной динамики лабораторных и природных популяций. Это ограничение роста, вызванное рядом факторов. Возможность множественных стационарных исходов зависит от начальных условий роста популяции. "Зависание"системы вблизи критического предела и ее чувствительность в этой области к небольшим колебаниям и индивидуальным усилиям. Замедленная реакция системы на изменения внешних факторов. Математические результаты, полученные при изучении моделей динамики популяций, служат практической цели управления биотехнологическими и природными системами и дают пищу для разработки собственных математических теорий.

Новый этап развития моделирования популяций начался в конце 20-го века, что было связано с двумя факторами. Во-первых, промышленная революция

и экстенсивный рост производства вызвали катастрофические последствия для природных экосистем, что сделало задачу прогнозирования таких последствий как никогда актуальной. Во-вторых, бурное развитие вычислительной техники существенно расширило как количественный, так и качественный инструментарий исследователей, что в том числе коснулось и моделирования популяций. Объединение этих факторов вылилось в возникновение и интенсивное развитие имитационного моделирования [5].

Энтузиазм нового направления, однако, со временем был исчерпан. Имитационная модель, по определению стремящаяся имитировать конкретный объект, при переносе на другой объект, даже родственной природы, часто требует внесения значительных изменений. Ограниченные возможности имитационного моделирования подтолкнули сообщество экологов к разумному скептицизму и возрождению интереса к классическому математическому моделированию.

Говоря о математическом моделировании в экологии, стоит отметить многообразие названий. Популяционная динамика, математическая биофизика популяций и сообществ, математическая экология – такое множество обобщающих наименований происходит от того, что в настоящее время это направление исследований еще не стало отдельной научной дисциплиной, с официальным и общепризнанным названием.

В некотором противоречии с необходимостью подробно учитывать особенности объекта моделирования лежит потребность принятия некоторой идеализации этого объекта при построении модели. При этом уменьшение числа независимых переменных, часто ведет к выявлению и пониманию более общих свойств моделируемого объекта. Так один коэффициент может объединять сразу множество факторов, (например, как K в 1.1), главное правильно передавать характер их зависимости – прямо- или обратно- пропорциональной; линейной или нелинейной, зависимость от одной или нескольких переменных.

Далее следует рассмотреть несколько общих упрощений, принимаемых для возможности свободно оперировать над моделью. В подавляющем большинстве работ по моделированию популяций используются таковые или схожие упрощения, или же выводятся новые модели, пытающиеся заполнить как раз одно из этих упрощений.

1. **Постоянство внешних условий.** С математической точки зрения это упрощение – условие постоянства используемых коэффициентов,

(например, a_i в 1.5). Это облегчает и поиск значений самих коэффициентов, как для определенного состояния абстрактной моделируемой ситуации, так и замер в реальной природной среде. Тем не менее, очевидно, что в природе оставаться постоянными внешние условия могут лишь либо в замкнутой системе, либо на коротких временных отрезках.

2. **Непрерывность численности популяции.** Очевидно, что число реальных особей в популяции – дискретно. Более того, взаимодействие между особями, с целью размножения или же внутривидовой или межвидовой конкуренции, тоже часто подразумевает дискретное взаимодействие конкретных особей. Тем не менее, так как модели обычно используются для моделирования популяций от сотен особей и более, с всевозрастающими порядками, можно воспользоваться законом больших чисел и пренебречь дискретной природой отдельных особей.
3. **Локальная сосредоточенность популяции.** В классических моделях рассматривается в первую очередь динамика численности популяции во времени. При этом игнорируется пространственное распределение этой популяции. Семейство моделей пространственно распределенных сообществ анализирует как изменение численности, так и пространственной организации со временем, но является менее изученным, по сравнению с локальными моделями. Математическим аппаратом в этих моделей служат, как правило, построенные по аналогии с химическими задачами уравнения диффузии с нелинейной правой частью, так называемые системы типа "диффузия-кинетика"[17].

Тем не менее, сколь много упрощений не было бы принято для построения модели, если мы хотим сохранить ее адекватность – она должна отражать реальные зависимости моделируемых объектов и явлений. Большинство простых – линейных случаев, уже исчерпали себя: были давно рассмотрены и сведены к тривиальным задачам. Более того, простейшие случаи нелинейных моделей, решаемые с помощью особых, изолированных алгоритмов, также уже практически не встречаются в современных задачах. Вследствие этого, в новых задачах, поставленных перед математическим моделированием, в том числе моделированием популяций, практически всегда оказывается сложная нелинейная система.

На практике, любое внесение нового компонента в уже изученную модель, вводит новую независимую переменную. Это, в свою очередь, увеличивает размерность модели, что усложняет попытки как аналитического, так и численного

решения. Кроме того, новые компоненты часто перемножаются между собой и уже существующими, что существенно усугубляет нелинейных характер модели.

Такое положение дел требует разработки более универсального метода работы с нелинейными моделями, решения системы нелинейных уравнений от многих неизвестных.

В рамках этого исследования рассматривается хорошо изученный подкласс динамических систем (Модель k конкурентов - k областей миграции), для которых уже доказано существование стационарных режимов и замкнутых орбит.

1.2 Модель k конкурентов - k областей миграции

1.2.1 Дальнейшее развитие модели

Математическое моделирование популяционных систем с учетом миграционных потоков и конкуренции видов представляет значительный научный и практический интерес. Например, они могут быть использованы для сохранения биоразнообразия, что является одной из ключевых задач современной экологии. Чтобы эффективно решить эту проблему, необходимо уметь прогнозировать изменение популяций с учетом целого ряда факторов, включая миграцию и конкуренцию между видами.

Уже упомянутая модель Лотки-Вольтерры была создана в середине 1920-х годов двумя выдающимися учеными. Альфред Лотка разработал свою собственную версию этой модели в 1925 году [2], и Вито Вольтерра независимо друг от друга создали аналогичную модель в 1926 году [3]. С тех пор эта модель стала основным инструментом для изучения различных популяций и прогнозирования их изменений. Базовая модель, как и ее последующие расширения, описывается системами нелинейных дифференциальных уравнений. Сложность решения таких систем значительно возрастает при увеличении размерности модели. При этом, как уже говорилось ранее, не существует универсального эффективного способа решения систем нелинейных уравнений.

Хотя базовая модель Лотки-Вольтерры "хищник-жертва"хорошо описана и изучена, для решения реальных задач часто используются более сложные модели. Так, например, в диссертационной работе Васильевой И.И. [1] исследуется семейство многомерных моделей типа "k конкурентов - k областей миграции". Для простого случая рассмотрим $k = 2$: два конкурирующих вида обитают в основном ареале и могут мигрировать в соответствующие миграционные убежища. Упрощенная четырехмерная модель (два конкурента - два убежища) задается системой:

$$\begin{cases} \dot{x}_1 = ax_1 - px_1^2 - rx_1x_3 + \beta x_2 - \beta x_1, \\ \dot{x}_2 = ax_2 - px_2^2 + \beta x_1 - \beta x_2, \\ \dot{x}_3 = ax_3 - px_3^2 - rx_1x_3 + \delta x_4 - \delta x_3, \\ \dot{x}_4 = ax_4 - px_4^2 + \delta x_3 - \delta x_4 \end{cases} \tag{1.6}$$

где x_1, x_3 плотности популяций конкурирующих видов в общем ареале обитания; x_2, x_4 плотности популяций этих видов в миграционных убежищах; a – коэффициент естественного прироста, в данном случае одинаковый для всех популяций; p – коэффициент внутривидовой конкуренции, также одинаковый для обоих видов; r – коэффициент межвидовой конкуренции; β, δ коэффициенты миграции. В работе [1] эта система идет под номером (2.3).

В [1], методом дифференциальной эволюции получают следующие значения для параметров модели: $a = 4.656449$, $p = 0.582056$, $r = 2.561047$, $\beta = 4.365421$, $\delta = 2.037196$. После этого, численные методы из библиотеки *sympy* используются для поиска состояний равновесия. Решение успешно находится, но имеет малую точность – до второго знака после запятой. Рассмотрим, как можно использовать построение базиса Гребнера для поиска состояний равновесия популяционной модели с алгебраической точностью.

1.2.2 Нахождение базиса Гребнера для нахождения равновесных состояний для четырехмерной модели

Состояния равновесия системы 1.6 являются корнями полиномиальной алгебраической системы, полученной путем приравнивания левых частей к ну-

лю – нулевые производные популяций означают отсутствие изменений, что и является состоянием равновесия для модели. Классические численные итерационные методы (методы Ньютона, эволюционные алгоритмы) решают эту систему приближенно и не гарантируют нахождения всех корней. В частности, Васильева И.И. указывает [1], что "размерность модели и возможности используемого прикладного программного обеспечения позволяют получить только часть решений".

Альтернативой является использование метода с построением базиса Гребнера [6; 18]. Зададим идеал $I = [f_1, \ldots, f_m]$ в кольце многочленов $k[x_1, \ldots, x_n]$. Тогда базис Гребнера такого идеала, с использованием лексикографического порядка, будет иметь вид треугольной (ступенчатой) декомпозиции системы: последний элемент базиса будет зависеть только от одного неизвестного - x_n. Одно из предшествующих значений в свою очередь будет зависит только от x_n и x_{n-1}, и так далее. Это позволяет последовательно находить все корни с помощью процедуры обратной подстановки.

Ключевое преимущество этого подхода заключается в следующем: в отличие от численных методов, которые могут как пропускать корни, так и давать лишние решения, алгоритм Бухбергера гарантирует, что найден полный набор равновесных состояний - ни одно из них не теряется и не добавляется.

Дополнительным преимуществом является точность. Корни последнего элемента базиса в системе SageMath представлены в виде алгебраических чисел: система представляет значения в виде корней алгебраического уравнения и может вычислять их числовое значение с произвольной заданной точностью (команда **n(x, digits=k)**).

Воспроизведем систему 1.6 в окружении SageMath. Для простейшего случая можно использовать встроенную функцию **variety()**:

```
x = var('x1, x2, x3, x4')
a = 4.656449;  p = 0.582056;  r = 2.561047
b = 4.365421;  d = 2.037196
eqs = [
    a*x1 - p*x1^2 - r*x1*x3 + b*x2 - b*x1,
    a*x2 - p*x2^2 + b*x1 - b*x2,
    a*x3 - p*x3^2 - r*x1*x3 + d*x4 - d*x3,
    a*x4 - p*x4^2 + d*x3  - d*x4
```

Таблица 1 — Состояния равновесия для 1.6

	x_1	x_2	x_3	x_4
S_1	0	0	0	0
S_2	0	0	8.000	8.000
S_3	8.000	8.000	0	0
S_4	0	0	2.436	-1.436
S_5	0	0	-1.436	2.436
S_6	1.419	3.522	4.021	6.625
S_7	5.788	6.843	0.813	5.062
S_8	1.000	3.000	5.000	7.000

]

```
Dics = (QQ[x]*eqs).variety(AA)
```

Команда **variety(AA)** непосредственно находит все вариации значений переменных x, которые превращают порождающие идеала, *eqs*, в ноль. Аргумент AA определяет использование алгебраических чисел. Результаты расчетов представлены в табличном виде.

Таким образом, метод находит 8 равновесных состояний1, из которых 3 не содержат нулевых популяций. В [1] численными методами также были обнаружены три состояния равновесия с ненулевыми составляющими: S_8(0.99, 2.99, 5.06, 7.04), S_9(1.44, 3.55, 4.00, 6.64), S_{10}(5.81, 6.87, 0.81, 5.08). Сравнение показывает, что точки S_6, S_7, S_8 в настоящей работе соответствуют точкам S_{10}, S_9, S_8 и легко узнаваемы.

Предоставленное решение позволяет получить значение каждого корня с произвольной точностью, используя команду **n(val, digits=?)**, при этом **?** указывает требуемое количество знаков после запятой.

1.2.3 Построение и нахождение стационарных состояний шестимерной модели

Рассмотрим более детальное применение предложенного подхода на более сложном примере – шестимерной модели ("четыре конкурента - две области миграции"). В качестве примера рассмотрим систему (3.7) из работы [1] с предлагаемым набором параметров.

Зададим систему:

```
x=var('x1,x2,x3,x4,x5,x6')
a1=9.87107026, a2=9.97602024, a3=9.94653833
a4=9.96876543, a5=1.42213144, a6=7.54755514
p11=0.10331098, p22=0.10190956, p33=0.14403938
p44=0.10093958, p55=2.44771688, p66=4.59652965
p13=0.12519358, p31=0.10242677, p56=0.97589257
p65=0.27400194, q15=0.10455808, q16=0.10524564
q35=0.11179302, q36=0.1032072, d15=6.31530649
d35=7.24952535, d16=8.66821615, d36=8.96873736
b=3.47237116, g=3.53131469, d=3.15533821, e= 3.04329167
eqs = [
    a1*x1 - p11*x1^2 - p13*x1*x3 - q15*x1*x5 - q16*x1*x6 + b*x2 - g*x1,
    a2*x2 - p22*x2^2 + g*x1 - b*x2,
    a3*x3 - p33*x3^2 - p31*x1*x3 - q35*x3*x5 - q36*x1*x6 + e*x4 - d*x3,
    a4*x4 - p44*x4^2 + d*x3 - e*x4,
    a5*x5 - p55*x5^2 - p56*x5*x6 + d15*x1*x5 + d35*x3*x5,
    a6*x6 - p66*x6^2 - p65*x5*x6 + d16*x1*x6 + d36*x3*x6,
]
```

Теперь объявим кольцо полиномов наших переменных и зададим над ним идеал в соответствии с уравнениями:

```
P = PolynomialRing(QQ, variables, order='lex')
J=P*[P(eq) for eq in eqs]
```

Найдем базис Гребнера:

```
B=J.groebner_basis()
```

Последний элемент базиса всегда зависит лишь от одного неизвестного:

```
B[-1]
```

$$x_6^{33} - 148\ldots872/801\ldots385 * x_6^{32} - \ldots$$

Моном x_6^{32} здесь имеет коэффициент с 182 знаками в числителе и 179 знаками в знаменателе.

Теперь, для нахождения корней уравнения, соответствующего последнему элементу базиса, выполним следующую команду:

```
Roots = QQ[x6](B[-1]).roots(AA, multiplicities=False)
Roots
```

[-148.94?, -142.45?, -77.77?, -77.16?, -74.88?, -30.17?, -13.25?, -13.18?, 0, 0.55?, 0.74?, 1.64?, 1.64?, 10.97?, 12.27?, 21.05?, 37.41?, 38.28?, 52.34?, 70.07?, 100.89?]

Теперь мы можем приступить к процедуре обратной замены, воссоздавая полные корни для нашей системы.

Конечно не следуют забывать, что нахождение базиса Гребнера является трудоемкой задачей. Для этого примера вычисления проводились на **Сервере**, с конфигурацией оборудования, описанной в более позднем разделе 4.1. На этом оборудовании вычисления по нахождению базиса в системе Sage заняли 205 секунд.

Для ускорения вычислительного процесса и расширения спектра моделей, доступных для исследований, попробуем использовать систему GInv, более подробно описываемую в последующих главах диссертации.

Тогда, для решения нашей задачи выполним следующий код:

```
from ginv.monom import *
from ginv.poly import *
from ginv.gb import *

variables = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6']
init(variables, Monom.TOPdeglex)
equations = [
    9.87107026*x1 - 0.10331098*x1**2 - ...,
```

```
    9.97602024*x2 - 0.10190956*x2**2 + ...,
    ...
]

G = GB()
G.algorithm2(equations)

print('Grobner Basis:')
print(G)
```

При той же конфигурации вычислительного оборудования, базис Гребнера был найден всего за 0,67 секунды. Дальнейшее нахождение состояний равновесия следует уже описанному алгоритму.

Для рассматриваемой модели, использование системы GInv не только сокращает время ожидания при работе с шестимерной моделью, но и открывает перспективы для исследования моделей б**о**льших размерностей. Подробнее работа с данной моделью рассматривается в работе [51].

В этом свете, рекомендуется использовать построение базиса Гребнера (алгоритмом Бухбергера или же инволютивным алгоритмом) в качестве основного инструмента для нахождения равновесных состояний при изучении миграционных и демографических моделей любой размерности. Приведенный пример решения может быть также использован для моделей из [1; 19—21] и их обобщениям с большим количеством видов и диапазонов.

1.3 Моделирование нелинейных осцилляторов

1.3.1 Движение нелинейного осциллятора

Как уже упоминалось, одна из сильных сторон математического моделирование – возможность переносить модель или решение из одной области в другую. Так, для биологических осцилляторов, чем и являются модели популяционной динамики, можно построить решения, аналогичные осцилляторам механическим.

Математический осциллятор – это система, совершающая колебания, в которой присутствует зависимость восстанавливающей силы, а значит и ускорения, от смещения системы относительно точки равновесия. Нелинейный осциллятор – осциллятор, в котором зависимость ускорения от смещения не является линейной. В отличие от линейного осциллятора, где период колебаний не зависит от амплитуды, в нелинейном осцилляторе период колебаний может зависеть от амплитуды, а также могут возникать другие нелинейные эффекты.

Для работы с нелинейными осцилляторами можно использовать метод решения по схеме средней точки. Этот метод используется в разностных схемах для численного решения дифференциальных уравнений, где данный метод применяется для оценки значения функции в середине интервала между двумя узловыми точками.

Так, например, рассмотрим уравнение:

$$x'' = F(x)$$

Его также можно записать в виде:

$$x' = y,$$

$$y' = F(x).$$

Используемая схема средней точки:

$$x_{k+1} - x_k = \frac{dt}{2}(y_{k+1} + y_k),$$

$$y_{k+1} - y_k = F(\frac{y_{k+1} + y_k}{2})dt.$$

Эта схема дает периодическую последовательность с периодом ndt в том и только в том случае, когда

$$x_n = 0, y_n = y_0$$

Таким образом, для построения приближенного периодического решения необходимо определить количество точек N, после чего построить систему нелинейных уравнений. Эти уравнения можно записать в виде $2N$ полиномов, которые, в свою очередь, можно объединить в идеал, для которого найти базис Гребнера, что сведет решение системы нелинейных уравнений к решению одного нелинейного уравнения.

1.3.2 Кубический нелинейный осциллятор

Рассмотрим нелинейный осциллятор:

$$\frac{dx}{dt} = y, \frac{dy}{dt} = -x^3$$

Для такого осциллятора существует периодическое решение, полученное по схеме Кагана, что было показано в работе [22]. В работе тех же авторов [23], однако, ставится вопрос о существовании периодических решений, найденных по схеме средней точки. Рассмотрим предлагаемую ими задачу:

Возьмем $N = 5$, достаточно небольшое для возможности выполнения реальных вычислений. Для схемы средней точки получается система, из которой после исключения $x_1, \ldots, x_4$ получается уравнение, связывающее начальную точку и период колебаний T. Это естественное следствие, так как в нелинейном случае период колебаний всегда зависит от начальных условий.

Зафиксируем начальную точку:

$$x_0 = (x_0, y_0) = (0, 1),$$

тогда получается уравнение для отыскания T^2, дающее один нулевой и два положительных корня. Оба ненулевых значения для T дают периодические решения, имеющие, однако, разный фазовый портрет. Так, для $T = 7.595$ на фазовой плоскости получается пятиугольник 1.1, а для $T = 60.75$ – пентаграмма 1.2. Точно такую же картину авторы получали для периодических решений эллиптического осциллятора, найденных по схеме Кагана [22].

Отсюда следует, что схема средней точки подражает периодичности рассматриваемого эллиптического осциллятора в схожей мере, что и схема Кагана.

Тем не менее, как и реализация схемы Кагана, нахождение приближенных периодических решений для нелинейного осциллятора, является трудновычислимой задачей. Компьютерные алгоритмы поиска решений имеют различную реализацию, однако их вычислительная сложность в любом случае остается экспоненциальной. Такая сложность значительно затрудняет вычисление приближенных решений для более больших значений N, что может быть интересно для исследования поведения таких решений на больших размерностях. Таким образом это дает повод протестировать систему GInv для решения данной задачи.

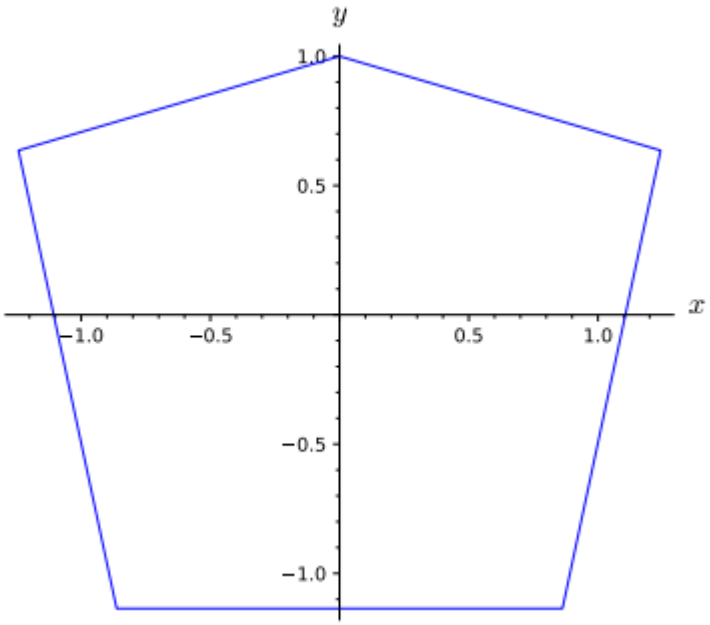

Рисунок 1.1 — Приближенные периодические решения системы уравнений для T ≈ 7.6

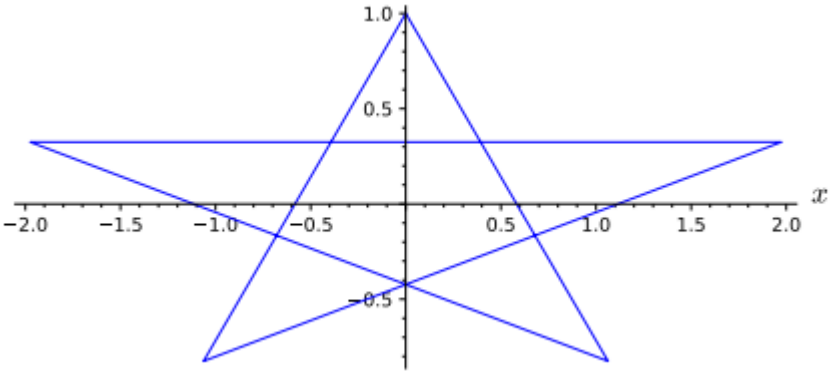

Рисунок 1.2 — Приближенные периодические решения системы уравнений для T ≈ 60.75

1.3.3 Нахождение решения для нелинейного осциллятора по схеме Кагана

Рассмотрим применение уже упомянутой системы GInv, позволившей добиться лучших результатов для прошлой модели, в рамках этой задачи. Начнем с формирования полиномиального идеала на основе системы нелинейных уравнений. Так для начала возьмем $N = 5$. Применяя схему средней точки, описанную ранее, получаем следующую систему нелинейных уравнений:

$$x_1 = 1/10T(y_1 + 1),$$

$$-x_1 + x_2 = 1/10T(y_1 + y_2),$$
$$-x_2 + x_3 = 1/10T(y_2 + y_3),$$
$$-x_3 + x_4 = 1/10T(y_3 + y_4),$$
$$-x_4 = 1/10T(y_4 + 1),$$
$$y_1 - 1 = -1/40Tx_1^3,$$
$$-y_1 + y_2 = -1/40T(x_1 + x_2)^3,$$
$$-y_2 + y_3 = -1/40T(x_2 + x_3)^3,$$
$$-y_3 + y_4 = -1/40T(x_3 + x_4)^3,$$
$$-y_4 + 1 = -1/40Tx_43$$

Перенесем все в левую часть уравнений, получая набор уравнений формата $p_1 = 0, \ldots, p_N = 0$, где $\{p_1, \ldots, p_N\}$ – набор полиномов, формирующих идеал. Для корректного нахождения базиса Гребнера, сформируем идеал над кольцом целых чисел, что значит – избавимся от дробей, помножив полиномы на входящие в их состав знаменатели. Раскроем скобки и запишем идеал в виде списка полиномов на языке Python. Итоговый вид идеала, записанного в среде Jupyter Notebook, можно видеть ниже:

```
def get_osc_5():
    return [
        10*x1 - T*y1 - T,
        -10*x1 + 10*x2 - T*y1 - T*y2,
        -10*x2 + 10*x3 - T*y2 - T*y3,
        -10*x3 + 10*x4 - T*y3 - T*y4,
        -10*x4 - T*y4 - T,
        40*y1 - 40 + T*x1**3,
        -40*y1 + 40*y2 + T*x1**3 + T*x2**3
                + 3*T*x1**2*x2 + 3*T*x1*x2**2,
        -40*y2 + 40*y3 + T*x2**3 + T*x3**3
                + 3*T*x2**2*x3 + 3*T*x2*x3**2,
        -40*y3 + 40*y4 + T*x3**3 + T*x4**3
                + 3*T*x3**2*x4 + 3*T*x3*x4**2,
        -40*y4 + 40 + T*x4**3
    ]
```

Далее, попробуем решение классическим алгоритмом, реализованным в системе компьютерной алгебры Sage. Объявим полиномиальное кольцо над 9 переменными, а от него – идеал с полученными полиномами, указав $\prec deglex$. Выполним поиск базиса Гребнера реализованным в Sage алгоритмом Бухбергера, при этом засекая затраченное время. Вычисления выполнялись на том же оборудовании, что и в пункте 1.2. Из результатов вычислений 1.3 видно, что общее время вычислений заняло 21.5 минут, или 1291 секунд.

```
B [29]: R.<x1, x2, x3, x4, y1, y2, y3, y4, T> = PolynomialRing(GF(32003))
        I = R.ideal(get_osc_5(), order='deglex')

        now = datetime.datetime.now()
        print('Start',now)
        G = buchberger(I)
        now = datetime.datetime.now()
        print('Finish', now)
        print(G)

        Start 2025-04-30 13:30:41.159108
        Finish 2025-04-30 13:52:12.779552
        Polynomial Sequence with 192 Polynomials in 9 Variables
```

Рисунок 1.3 — Результат вычислений Sage для осциллятора ($N = 5$)

Для более большого значения N, например, $N = 10$, повторим действия по формированию полиномиального идеала. Получим следующее объявление:

```
def get_osc_10():
    return [
        80*x1 - 4*T*y1 - 4,
        -80*x1 + 80*x2 - 4*T*y1 - 4*T*y2,
        -80*x2 + 80*x3 - 4*T*y2 - 4*T*y3,
        -80*x3 + 80*x4 - 4*T*y3 - 4*T*y4,
        -80*x4 + 80*x5 - 4*T*y4 - 4*T*y5,
        -80*x5 + 80*x6 - 4*T*y5 - 4*T*y6,
        -80*x6 + 80*x7 - 4*T*y6 - 4*T*y7,
        -80*x7 + 80*x8 - 4*T*y7 - 4*T*y8,
        -80*x8 + 80*x9 - 4*T*y8 - 4*T*y9,
        -80*x9 - 4*T*y9 - 4*T,
        T*x1**3 + 80*y1 - 80,
        x2**3*T + x3**3*T + 3*x2**2*x3*T + 3*x2*x3**2*T - 80*y2 + 80*y3,
```

```
    x3**3*T + x4**3*T + 3*x3**2*x4*T + 3*x3*x4**2*T - 80*y3 + 80*y4,
    x4**3*T + x5**3*T + 3*x4**2*x5*T + 3*x4*x5**2*T - 80*y4 + 80*y5,
    x5**3*T + x6**3*T + 3*x5**2*x6*T + 3*x5*x6**2*T - 80*y5 + 80*y6,
    x6**3*T + x7**3*T + 3*x6**2*x7*T + 3*x6*x7**2*T - 80*y6 + 80*y7,
    x7**3*T + x8**3*T + 3*x7**2*x8*T + 3*x7*x8**2*T - 80*y7 + 80*y8,
    x8**3*T + x9**3*T + 3*x8**2*x9*T + 3*x8*x9**2*T - 80*y8 + 80*y9,
    T*x9**3 - 80*y9 + 80
]
```

Поиск базиса Гребнера был запущен на том же оборудовании и для $N = 10$, однако при непрерывном вычислении на протяжении двух недель, не было получено результатов и процесс вычисления пришлось прервать.

Теперь повторим то же самое с использованием системы GInv. Так как она интегрируется в среду Jupyter Notebook, можно использовать уже имеющиеся объявления набора полиномов.

Объявим вспомогательную функцию init, для задания используемых переменных и выбранного мономиального порядка:

```
def init(variables, order = Monom.TOPdeglex):
    Monom.init(variables)
    Monom.variables = variables.copy()
    Monom.zero = Monom(0 for _ in Monom.variables)
    Monom.cmp = order
    Poly.cmp = order
    for i in range(len(Monom.variables)):
        p = Poly()
        p.append([Monom(0 if l != i else 1
                for l in range(len(Monom.variables))), 1])
        globals()[Monom.variables[i]] = p
```

Далее используем ее для задачи 9 переменных, необходимых для поиска приближенных решений в случае $N = 5$, и оставим порядок, используемый в системе GInv по умолчанию – $\prec deglex$, такой же, как и использованный в тесте с Sage. Итоговый код представлен ниже:

```
def ginv_basis():
```

```
init(['x1', 'x2', 'x3', 'x4', 'y1', 'y2', 'y3', 'y4', 'T'])
G = GB()
G.algorithm2(get_osc_5())

# вывод базиса
print('Grobner Basis:')
print(G)
print(", ".join(str(g.lm()) for g in G)) # вывод старших степеней
print("crit1 =", G.crit1, "crit2 =", G.crit2)
print("time %.2f" % G.time) # вывод времени вычислений
```

Полученный код запустим в среде Python3, через терминал на том же оборудовании. Как видно из представленных логов 1.4, время вычисления – 273 секунды. Таким образом, для решения системы уравнений по задаче об эллиптическом осцилляторе, с параметром $N = 5$, система GInvDist дает результат в $\tilde{4}.73$ раза быстрее, чем классический алгоритм, выполненный через Sage.

```
user@gerdt:~/notebooks/ginv/pyginv$ python3 osc.py
Grobner Basis:
y2*-1 + y3,
y1*-1 + y4,
x3*20 + x4*-20 + T*-1,
x2*20 + x4*20 + T,
x1*-1 + x4*-1,
y4*T*-1 + x4*-10 + T*-1,
y3*T*2 + x4*-20 + T*-1,
y3*y4*-2 + y4**2*-2 + y3*2 + y4 + 1,
y3**2*-2 + y4**2*2 + y3*-1 + y4,
x4*y3*-2 + x4*y4*-2 + x4*-1,
T**4 + y4**3*307200000 + y4**2*238080000 + y3*-130880000 + y4*-248640000 + -165760000,
x4*T**3 + y4**3*-3840000 + y4**2*-2816000 + y3*1600000 + y4*3056000 + 2000000,
x4**2*T**2 + y4**3*32000 + y4**2*20800 + y3*-12800 + y4*-24000 + -16000,
x4**3*T + y4*-40 + 40,
y4**4*-80 + y4**3*-132 + y4**2*-24 + y3*64 + y4*117 + 55,
x4*y4**3*400 + x4*y4**2*260 + x4*y4*-140 + x4*-125 + T*-1,
x4**2*y4**2*4000 + x4**2*y4*-1400 + x4*T*125 + T**2,
x4**3*y4*40000 + x4**3*-54000 + x4**2*T*-5400 + x4*T**2*-125 + T**3*-1,
x4**4*-1 + y4**2*-4 + 4
y2, y1, x3, x2, x1, y4*T, y3*T, y3*y4, y3**2, x4*y3, T**4, x4*T**3, x4**2*T**2, x4**3*T, y4**4, x4*y4**3, x4**2*y4**2, x4**3*y4
, x4**4
crit1 = 2773 crit2 = 3835
time 273.06
user@gerdt:~/notebooks/ginv/pyginv$
```

Рисунок 1.4 — Результат вычислений GInv для осциллятора ($N = 5$)

Аналогичные вычисления были также проведены и для случая $N = 10$, однако дали схожий с предыдущим результат – после непрерывного вычисления на протяжении двух недель, результат не был получен и процесс вычисления был вынужденно прерван.

Таким образом мы получаем положительный, но пока лишь ограниченно полезный результат в применении системы GInvDist для задачи моделирования нелинейного осциллятора. Применение инволютивного алгоритма кратно

повышает эффективность вычислений, однако недостаточно для решения задач значительно большей размерности.

Глава 2. Математическая основа численных и символьных методов для решения систем нелинейных уравнений

2.1 Базисы Гребнера

Базис Гребнера, для полиномиального идеала $Ip_1, , p_n$ над кольцом F является конечным множеством $G = g_1, , g_m$ полиномов над F, которые порождают идеал I, и для любого $p \in I$ существует полином $g \in G$ такой, что старший терм p кратен начальному члену g.

Изучение задач механики и математической физики часто сводится к решению систем полиномиальных уравнений. Такие задачи не имеют универсального численного метода решения в случаях с двумя и более неизвестными. При этом нахождение базиса Гребнера для идеала, формируемого полиномами, соответствующими уравнениям системы, позволяет свести решение системы нелинейных уравнений с конечным числом решений к решению одного уравнения с одним неизвестным.

Алгоритм Бухбергера позволяет найти базис Гребнера за конечное число шагов, но на практике его сложно использовать для решения систем двузначной и более высокой размерности. Так как в основе этого алгоритма лежит полный перебор всевозможных пар делимого и делителя, очевидна его экспоненциальная сложность.

Так, например, рассмотрим классическое для этой области семейство систем уравнений *cyclic*. Системы этого семейства разнятся размерностью и имеют простой механизм формирования, так *cyclic4*, с размерностью $n = 4$, будет состоять из четырех уравнений, составленных из четырех переменных:

$f_1 = a + b + c + d$

$f_2 = ab + ad + bc + cd$

$f_3 = abc + abd + acd + bcd$

$f_4 = abcd - 1$

Базис Гребнера для такой системы будет равен:

$g_1 = c^2d^6 - c^2d^2 - d^4 + 1$

$g_2 = c^3d^2 + c^2d^3 - c - d$

$g_3 = bd^4 - b + d^5 - d$

$g_4 = bc - bd^5 + c^2d^4 + cd - d^6 - d^2$

$g_5 = b^2 + 2bd - d^2$

$g_6 = a + b + c + d$

Таким образом, для четырех полиномов от четырех переменных, мы получаем базис Гребнера с шестью полиномами, суммарно с двадцатью пятью мономами. При этом, при размерности равной пяти, с пятью переменными и пятью полиномами, базис Гребнера, вычисленный в системе компьютерной алгебры SageMath, насчитывает сорок шесть полиномов, с коэффициентами мономов значением до сотен тысяч. Количество операций, требующихся для вычисления базиса, при переходе от $n = 4$ к $n = 5$ выросло на несколько порядков. Это следствие экспоненциальной сложности алгоритмов нахождения базиса Гребнера. Таковая вычислительная сложность, несмотря на удобство его использования в широком спектре задач, затрудняет широкое применение базиса Гребнера в реальных задачах.

2.1.1 Полиномы от одной переменной

Полиномы от одной переменной образуют евклидово кольцо. При этом операция деления задается формулой $f = qg + r$, где $deg(lm(r)) < deg(lm(g))$ или $r = 0$, где f – делимое, а g – делитель. При этом частное q и остаток r определены однозначно, а старший моном определяется по степени. Полиномы от одной переменной образуют кольцо главных идеалов, т.е. идеалов порожденных одним полиномом. В результате, задача о принадлежности полинома f к идеалу $(g_1, ..., g_m)$ может быть решена двумя шагами:

1. построение базиса идеала специального вида (представленного одним полиномом $g = gcd(g_1, ..., g_m)$);
2. нахождение остатка от деления полинома f на базисный элемент g.

Основной процедурой в обоих шагах алгоритма является операция взятия остатка от деления полиномов. При этом продолжать выполнять деление и брать остаток можно при условии $(\exists g \in G)(\exists u \in r)[deg(lm(g)) <= deg(u)]$.

Следовательно, для построения базиса идеала достаточно найти наибольший общий делитель для множества полиномов. Для его нахождения необходимо будет провести все возможные взаимные редукции пар полиномов,

так как процесс редукции – сокращения полинома, и дает результат деления а также получаемый остаток.

В случае полиномов от одной переменной, выполнение такого алгоритма дает каноническое представление для базиса идеала, поскольку полином, образующий идеал, в процессе алгоритма будет приведен к виду, при котором его старший коэффициент равен единице.

2.1.2 Полиномы от нескольких переменных

Теперь рассмотрим общий случай для полиномов от нескольких переменных.

Для выполнения рассматриваемого алгоритма ключевым будет задать упорядочение мономов. Если в случае полиномов от одной переменной задача порядка является тривиальной, то для полиномов от нескольких переменных необходимо четко обозначить используемый порядок. Такое упорядочение должно быть линейным и должно сохранять порядок между мономами при домножении на одинаковые термы. Таким образом,

Определение 1. Линейное отношение порядка $\prec$ на множестве мономов называется допустимым, если выполнены следующие условия:

1. $(\forall u \in M)[1 \prec u]$;
2. $(\forall u, v, w \in M)[u \prec v \rightarrow uw \prec vw]$.

Примером допустимого мономиального упорядочения может служит упорядочение мономов от одной переменной по степени. Его наиболее применяемыми обобщениями на случай многих переменных являются: лексикографическое ($\prec lex$) и сначала по полной степени и затем обратное лексикографическое ($\prec degrevlex$).

Имея заданный порядок, мы можем обозначить $lm(p)$ как старший моном полинома p, т.е. первый моном по всем выбранным порядкам, $lt(p)$ – старший терм, т.е. старший моном моном без коэффициента, $lc(p)$ – наоборот, старший коэффициент.

Для полиномов от одной переменной было достаточно перебрать все возможные редукций с использованием стандартного деления. Для полиномов от

многих переменных это сделать невозможно, так как в данном случае деление мономов не имеет линейного упорядочения по умолчанию. Возможна ситуация, когда образующие идеала имеют старшие мономы, которые не позволяют провести взаимную редукцию непосредственно, но при этом редукция может быть осуществлена с помощью определенной комбинации этих образующих. В таком случае дополним определение базиса с помощью выбираемого упорядочения:

Определение 2. Для допустимого упорядочения $\prec$ конечное множество $G \subset R$ называется базисом Гребнера идеала $(G) \in R$, если
$(\forall f \in (R))(\exists g \in G)[lm(g)|lm(f)]$.

Алгоритмическое решение задачи построения базисов Гребнера было предложено Бухбергером [18]. Для этого было введено определение S-полинома:

Определение 3. [18] S-полиномом для $f, g \in R$ называется комбинация
$Spoly(f,g) = (lcm(lm(f), lm(g))/lt(f))f - (lcm(lm(f), lm(g))/lt(g))g$.
Эту комбинацию будем считать остатком от деления f на g.

Теорема 1. [18] Пусть (G) — некоторый полиномиальный идеал. Тогда базис $G = g_1, ..., g_m$ идеала (G) является базисом Гребнера тогда и только тогда, когда для всех пар $i \neq j$ остаток от деления $Spoly(g_i, g_j)$, где $g_i, g_j \in G$, равен нулю.

Исходя из этой теоремы, можно предложить алгоритм построения базисов Гребнера. Как и в случае полиномов от одной переменной, задача о принадлежности идеалу решается в два этапа.

Определение 4. Алгоритм Бухбергера для нахождения базиса полиномиального идеала I с образующими от кольца $F[x_1, \cdots, x_n]$:

1. $G = F$.
2. Для каждой пары полиномов $g_i, g_j \in G$ найдем результат редукции $Spoly(g_i,g_j)$ - остаток от деления полиномов.
3. Продолжим редуцировать $Spoly$, рассматривая для этого различные комбинации полиномов из G пока не получим не редуцируемый остаток. Если он не равен нулю, добавим его к G.
4. Повторим шаги 2 и 3 для всех возможных пар полиномов в G, включая новые полиномы, добавляемые на шаге 3.

После исчерпания всех возможных пар, G — базис Гребнера.

Данный алгоритм имеет экспоненциальную вычислительную сложность, а также использует экспоненциальные объемы памяти.

2.1.3 Нахождение базиса Гребнера

Потребность в нахождении базисов Гребнера обусловлена следующей задачей.

Задача 1. В полиномиальном кольце задан идеал. Требуется выяснить, принадлежит ли ему заданный многочлен.

Решение этой задачи достигается путем построения базиса Гребнера идеала J, более того, решение многих других задач (исключение неизвестных, вычисление радикала из идеала) в существенном используют базисы Гребнера [6]. Первый и до сих пор активно используемый алгоритм отыскания базисов Гребнера, как и само это понятие был предложен учеником Гребнера — Бухберегером. К сожалению, с ростом числа переменных алгоритм Бухбергера требует слишком много ресурсов, фактически сейчас доступны вычисления с десятком переменных, причем сами вычисления могут занимать дни и гигабайты памяти [24]. Особенно неприятно то, что не удается эффективно использовать симметрии идеала.

По этой причине безусловно важным являются поиски альтернативных методов вычисления базисов Гребнера. Один из таких методов основан на инволютивном делении. Понятие об инволютивном делении пришло в алгебру из исследований совместности систем дифференциальных уравнений в частных производных, восходящих к работам Riquier (1910), Janet (1920), Thomas (1937). Начиная с середины 1990-х годов В.П. Гердт и его ученики А. Ю. Жарков и Ю.А. Блинков публикуют серию работ, в которых развили это понятие в абстрактно алгебраическом виде и указали на широкие возможности применения инволютивных базисов как альтернативы обычных базисов Гребнера. Первый пример инволютивного деления — деление Поммаре — был введен Жарковым в 1993 году [10; 11; 25—28]. В общем виде понятие инволютивного деления было введено в работах В.П. Гердта и Ю.А. Блинкова [8; 9; 29—34]. Наиболее полно этот метод представлен в докт. дисс. Ю.А. Блинкова [35], где в частности

приведены доказательства основных теорем. Уже после защиты этой работы в [13] В.П. Гердт и Ю.А. Блинков предложили новый способ введения инволютивных делений и несколько модифицировали основные определения теории. Наиболее современное изложение вопроса было представлено в докладе, который В.П. Гердт сделал в РУДН в ноябре 2020 года.

Определение 5 (Гердт, Блинков, 2020)**.** Говорят, что на M задано инволютивное деление L, если любому множеству $U \in P_{fin}(M)$, где $P_{fin}(M)$ есть множество всех конечных подмножеств M, и любому моному $u \in U$ поставлен в соответствие такое множество $L(u,U)$, что

1. $L(u,U)$ — подмоноид в M,
2. $w \in L(u,U)$ и $v|w$ влечет $v \in L(u,U)$,
3. $u,v \in U$ и $uL(u,U) \cap vL(v,U) \neq \varnothing$ влечет $u \in vL(v,U)$ или $v \in uL(u,U)$ (единственность делителя),
4. $u,v \in U$ и $v \in uL(u,U)$ влечет $L(v,U) \subseteq L(u,U)$ (транзитивность)
5. $V \subseteq U$ влечет $L(v,U) \subseteq L(v,V)$ $\forall v \in V$.

Замечание. Эта формулировка взята из доклада 2020 г. и существенно совпадает с определением 1 из [13].

Определение 5 очень сложно и в действительности содержит несколько конструкций, связь между которыми только еще предстоит изучить. Из общих соображений не ясно, по какому принципу взяты именно такие аксиомы, можно ли внести в них изменения. Из недавних работ Ю.А. Блинковым было почерпнуто определение посредством введения мономиальных конусов, что делает теорию более конструктивной. В настоящей работе, среди прочего, делается попытка разделить определение инволютивного деления на отдельные части.

2.2 Линейные деления

2.2.1 Обозначения

Условимся обозначать как $P_{fin}(M)$ множество всех конечных подмножеств множества M, причем для определенности будем считать, что $\varnothing \in$

$P_{fin}(M)$, и что в рассматриваемых подмножествах ни один из элементов M не повторяется дважды.

Пусть k — произвольное поле, $X = [x_1, \ldots, x_n]$ — список независимых переменных, а M — множество всех мономов кольца $k[x_1, \ldots, x_n]$. Тогда любое множество $G \in P_{fin}(k[x_1, \ldots, x_n])$ порождает идеал, который мы будем обозначать как $\langle G \rangle$. При этом G называется базисом идеала $\langle G \rangle$.

Пусть на M задан некоторый мономиальный порядок $(<)$ и пусть

$$\operatorname{lm} G = \{\operatorname{lm}(g) \quad \forall f \in G\}.$$

Будем всюду предполагать, что в G нет двух многочленов, имеющих один и тот же старший моном, тогда

$$G \in P_{fin}(k[x_1, \ldots, x_n]) \Rightarrow \operatorname{lm} G \in P_{fin}(M).$$

Заметим, что базис идеала нетрудно преобразовать так, чтобы выполнялось указанное условие.

2.2.2 Линейное деление

Определение 6. Линейное деление L — это бинарное отношение на множестве $U \times M$, обладающее следующими свойствами: для любых u,$v \in U \in P_{fin}(M)$ и $w \in M$ верно

1. если $u\mathrm{L}w$, то u делит w в обычном смысле,
2. $u\mathrm{L}u$,
3. если $u\mathrm{L}w$ и $v\mathrm{L}w$, то $u\mathrm{L}v$ или $v\mathrm{L}u$,
4. если $u\mathrm{L}v$ и $v\mathrm{L}w$, то $u\mathrm{L}w$

Замечание. Нетрудно видеть, что в этом определении учтены аксиомы 2-4 определения 5. Символ L мы будем использовать так же, как символ обычного деления |. В символьном выражении символ L имеет наивысший приоритет по сравнению с арифметическими действиями и вычислением с старшего монома и т.п.

В кольце $k[x_1]$ обычное деление удовлетворяет всем указанным аксиомам и является линейным. Однако, при $n > 1$ обычное деление не является линейным и это и явилось препятствием на пути применения алгоритма Евклида в кольцах многочленов многих переменных.

Следует подчеркнуть, что в силу аксиомы 1

$$u \mathrm{L} w \Rightarrow u | w,$$

но

$$u | w \not\Rightarrow u \mathrm{L} w.$$

Не всякий обычный делитель является делителем в смысле линейного деления. Это — плата за единственность делителя, указанную в аксиоме 3 определения 6.

2.2.3 Алгоритм Евклида

Классический алгоритм Евклида деления многочлена $f \in k[x_1]$ на многочлен g можно перенести на случай многих переменных, если заменить деление на линейное деление, а многочлен g на множество многочленов G.

Алгоритм Евклида деления многочлена $f \in k[x_1, \ldots, x_n]$ на многочлены из списка $G \in P_{fin}(k[x_1, \ldots, x_n])$ можно описать следующим образом.

Пусть $r = 0$.

Шаг 1. Ищем мономы из $\mathrm{lm}\, G$, которые делят $\mathrm{lm}\, f$ в смысле определения 6. Среди них в силу свойства 2 определения 6 определяем тот, который делит остальные. Допустим, что таковым оказался $\mathrm{lm}\, g$. Тогда заменим f на

$$f - \frac{\mathrm{lc}(f)}{\mathrm{lc}(g)} g.$$

Применим к новому многочлену ту же процедуру до тех пор, пока $\mathrm{lm}\, f$ не делится ни на один из мономов множества $\mathrm{lm}\, G$. Убавим из r и f слагаемое $\mathrm{lt}(f)$.

Шаг 2. К новому многочлену f применяем шаг 1.

На каждом шаге мы уменьшаем $\mathrm{lm}\, f$, поэтому за конечное число шагов мы придем к $f = 0$. Полученный при этом многочлен r мы будем называть остатком от деления f на G.

Замечание. В [35] вместо термина остаток используется термин нормальная форма.

Замечание. От обобщения алгоритма Евклида по Бухбергеру [6] этот алгоритм отличается лишь одной деталью. На каждом шаге мы ищем моном, который делит старший моном в f. В варианте Бухбергера выбор этого монома не определен однозначно, его можно брать первым по списку G или, напр., случайным образом. В нашем же случае берется моном, который делит старший моном в f в смысле определения 6, и этот выбор определен однозначно, для чего и нужна аксиома 3.

2.2.4 Редуцируемые суммы

В описанном алгоритме не используются те элементы g множества G, для которых можно найти $g' \in G$ такой что, $\mathrm{lm}(g')\mathrm{L}\,\mathrm{lm}(g)$. Множество G, из которого удалены неиспользуемые элементы мы будем обозначать как G'.

Замечание. Разумеется, вполне очевидно, что нельзя просто так выкидывать элементы из G: идеалы, порожденные G и $R(G)$, могут и не совпадать. Чтобы избежать этого, в [35] вводится понятие авторедукции базиса. Мы, однако, не хотим здесь спешить.

Заметим, что на каждом шаге алгоритма Евклида $f - r$ отличается от f на некоторый элемент cmg, где $c \in k$ — константа из k, m — моном и многочлена g — многочлен из G', причем $\mathrm{lm}\, g$ делит линейно произведение $m \cdot \mathrm{lm}\, g$.

Определение 7. Конечную сумму, где каждое слагаемое в сумме представляет собой произведение константы $c \in k$, монома $m \in M$ и многочлена $g \in G'$, причем $\mathrm{lm}\, g$ делит линейно произведение $m \cdot \mathrm{lm}\, g$, будем называть редуцируемой суммой.

Условимся, что в редуцируемой сумме слагаемые вида cmg и $c'mg$ всегда приведены к $(c + c')mg$. Множество всех редуцируемых сумм образует бесконечномерное линейное пространство, будем обозначать как $R(G)$.

Приняв это определение, мы можем сказать, что остаток r обладает следующими свойствами:

1. $f = r + s$, где s — редуцируемая сумма или нуль,
2. все мономы, входящие в r, не делятся на мономы $\mathrm{lm}\, G$ в смысле определения 6.

Лемма 1. Пусть f — ненулевая редуцируемая сумма, тогда в ней найдется такое слагаемое cmg, $c \in k$, $m \in M$, $g \in G'$, что

$$\mathrm{lt}(f) = cm\,\mathrm{lt}(g).$$

Доказательство. Пусть

$$f = \sum cmg \in R.$$

Тогда для некоторого $g \in G'$ верно

$$\mathrm{lm}(f) = m\,\mathrm{lm}(g).$$

Вообще говоря, в сумме может быть несколько слагаемых, дающих старший моном. Допустим, что

$$\mathrm{lm}(mg) = \mathrm{lm}(m'g'), \quad g \neq g'.$$

Обозначим этот моном как w, тогда

$$\mathrm{lm}(g) \mathrm{L} w, \quad \mathrm{lm}(g') \mathrm{L} w.$$

В силу аксиомы 3 определения 6 это означает, что или $\mathrm{lm}\, g$ делит $\mathrm{lm}\, g'$, или $\mathrm{lm}\, g'$ делит $\mathrm{lm}\, g$. Ни то, ни другое невозможно, поскольку $g, g' \in G'$. Поэтому старший член дает в точности одно слагаемое. □

Теорема 2. Остаток от деления f на G равен нулю тогда и только тогда, когда f — редуцируемая сумма.

Доказательство. В одну сторону теорема уже доказана выше. Для доказательства в обратную сторону, допустим, что

$$f = \sum cmg \in R.$$

Тогда в силу леммы 1

$$\mathrm{lt}(f) = cm\,\mathrm{lt}(g).$$

Но тогда на первом шаге алгоритма Евклида мы избавимся от одного из слагаемых из суммы. Следующие шаги избавят нас от оставшихся слагаемых ровно по тем же причинам. □

Теорема 3. Остаток от деления на G суммы двух многочленов равен сумме их остатков.

Доказательство. Положим $f_3 = f_1 + f_2$, тогда в силу теоремы 2

$$f_i = r_i + s_i, \quad s_i \in R,\ i = 1,2,3,$$

поэтому

$$r_1 + r_2 - r_3 = s, \quad s \in R.$$

Если сумма s – ненулевая, то в силу леммы 1 ее старший моном равен $m \operatorname{lm} g$, то есть делится на $\operatorname{lm} g$ в смысле определения 6. С другой стороны страший моном $r_1 + r_2 - r_3$ является одним из мономов r_1, r_2 или r_3. По построению ни один из этих мономов не делится $\operatorname{lm} g$ в смысле определения 6. Поэтому $r_1 + r_2 - r_3 = 0$. □

Замечание. Теоремы 2 и 3 — вариация на тему теоремы 40 и ее следствия из [35] с заменой авторедуцированного базиса на G'.

Теорема 4. Многочлен r является остатком от деления f на G тогда и только тогда, когда $f = r + s$, где s — редуцируемая сумма, а все мономы, входящие в r, не делятся на мономы $\operatorname{lm} G$ в смысле определения 6.

Нетрудно видеть, что доказанные теоремы является следствием аксиомы 3 определения 6. Ее аналог можно доказать для алгоритма Евклида с обычным делением только для того случая, когда G — базис Гребнера. Собственно говоря, в этом и состоит выигрыш от введения линейного деления.

2.2.5 Симметрии

Пусть σ — произвольная перестановка множества X.

Определение 8. Множество $G \in P_{fin}(k[x_1, \ldots, x_n])$ будем называть симметричным относительно перестановки σ, если $\sigma G = G$.

Определение 9. Линейное деление на $U \times M$ будем называть симметричным, если U — симметрично относительно σ и из $u \mathrm{L} w$ следует $\sigma u \mathrm{L} \sigma w$.

Теорема 5. Пусть $G \in P_{fin}(k[x_1, \ldots, x_n])$ и линейное деление L на $\operatorname{lm} G \times M$ симметричны относительно перестановки σ, тогда перестановка σ переводит остаток от деления f на G в остаток от деления σf на G.

Доказательство. Пусть r — остаток от деления f на G, тогда

$$f = r + s,$$

где s — редуцируемая сумма, и поэтому

$$\sigma f = \sigma r + \sigma s.$$

Перестановка переводит член cmg редуцируемой суммы в выражение

$$c\sigma(m)\sigma(g).$$

В силу симметричности G, $\sigma(g) \in G$, а в силу симметричности L из $\operatorname{lm} g \mathrm{L} m \operatorname{lm} g$ следует $\operatorname{lm} \sigma g \mathrm{L} \sigma m \operatorname{lm} \sigma g$. Поэтому σg — редуцируемая сумма. В силу теоремы 2 остаток от этой сумму равен нулю, а тогда в силу теоремы 3 остаток от $\sigma f = \sigma r + \sigma s$ равен остатку от σr. Все мономы, входящие в r, не делят мономы из $U = \sigma U$. Следовательно, мономы, входящие в σr, тоже не делят мономы из U. Поэтому остаток от σr совпадает с σr. □

2.3 Конические деления

2.3.1 Коническое деление

Все известные способы задания линейного деления на $U \times M$ используют конструкцию с мономиальными конусами.

Определение 10. Пусть $m \in M$, $X' \in P_{fin}(X)$, тогда бесконечное подмножество множества M, образованное произведением m на всевозможные произведения элементов из X', называется мономиальным конусом с вершиной m и образующими X'. Это множество будем обозначать как $c(m,X')$.

Примем для определенности, что

$$c(m,\varnothing) = \{m\}.$$

Теорема 6. Пусть отображение

$$\mathcal{X} : U \to P_{fin}(X)$$

обладает следующим свойством: если $u,v \in U$, то конусы $c(u,\mathcal{X}(u))$ и $c(v,\mathcal{X}(v))$ или не пересекаются, или один из них целиком содержит другой. Тогда оно задает линейное деление L на $U \times M$:

$$u\mathrm{L}w \Leftrightarrow w \in c(u,\mathcal{X}(u)).$$

Замечание. Вообще говоря, нет определенного запрета на рассматрение случая, когда $\mathcal{X}(u) = \varnothing$ и поэтому $c(u, \mathcal{X}(u)) = u$.

Доказательство. Прежде всего $w \in c(u,\mathcal{X}(u))$ есть бинарное отношение на $M \times U$. Остается проверить аксиомы определения 6. Если $w \in c(u,\mathcal{X}(u))$, то w есть произведение u и некоторой комбинации мономов из $\mathcal{X}(u)$, поэтому выполнено свойство 1. Поскольку вершина всегда принадлежит конусу, выполнено свойство 2. Если

$$w \in c(u,\mathcal{X}(u)), \quad w \in c(v,\mathcal{X}(v)),$$

то эти конуса пересекаются. По условию теоремы это означает, что один из них принадлежит целиком другому. Пусть, например,

$$c(u,\mathcal{X}(u)) \subset c(v,\mathcal{X}(v)),$$

тогда $u \in c(v,\mathcal{X}(v))$ и поэтому выполняется аксиома 3. Наконец, пусть

$$v \in c(u,\mathcal{X}(u)), \quad w \in c(v,\mathcal{X}(v)).$$

Из $v \in c(u,\mathcal{X}(u))$ следует, что конусы $c(u,\mathcal{X}(u))$ и $c(v,\mathcal{X}(v))$, пересекаются. По условию теоремы это означает, что

$$c(u,\mathcal{X}(u)) \subseteq c(v,\mathcal{X}(v))$$

или

$$c(v,\mathcal{X}(v)) \subseteq c(u,\mathcal{X}(u)).$$

В первом случае $u \in c(v,\mathcal{X}(v))$, то есть u делится на v в обычном смысле, а v делится на u. Это означает, что $u = v$, а это невозможно. Но тогда

$$w \in c(v,\mathcal{X}(v)) \subseteq c(u,\mathcal{X}(u)),$$

что и дает нам свойство 4. □

Определение 11. Линейное деление, описанное в теореме 6, будем называть коническим.

Замечание. Конические деления — это линейные деления, для которого выполнены все аксиомы определения 5, кроме последней.

2.3.2 Визуализация конических делений

Свойства конического деления можно легко визуализировать для монов от двух переменных. Отложим по оси x и y соответствующие данным переменным степени. Тогда дискретные точки на графике будут представлять собой терм, получаемый комбинацией степеней x и y. Например, терм x^2y образует следующий конус 2.1. Все точки, попадающие в конус, соответствуют термам, делимым на образующий.

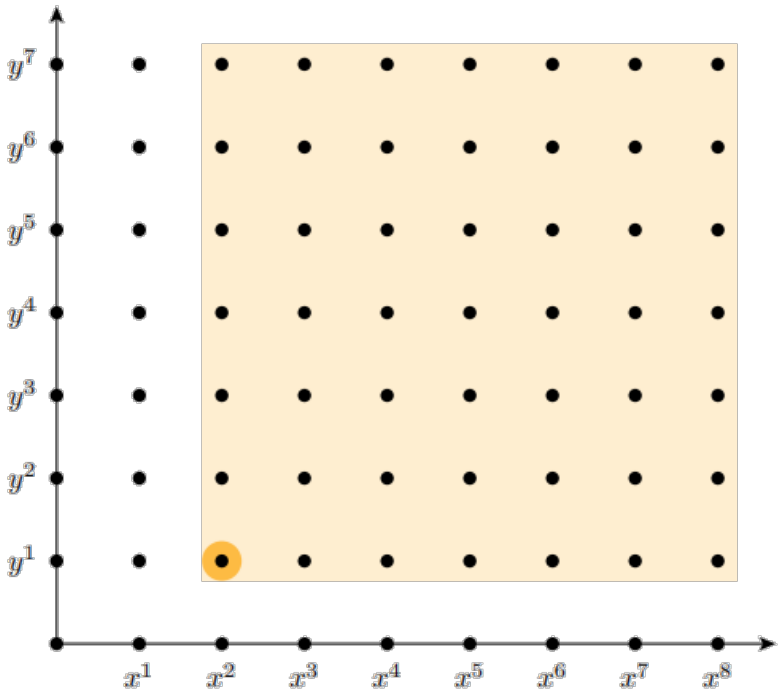

Рисунок 2.1 — Конус терма x^2y.

Рассмотрим набор термов $[x^7y, x^5y^2, x^4y^3, x^2y^4]$. Для такого набора термов можно построить соответствующий график 2.2. Точки, попадающие под пересечение конусов соответствуют термам, имеющим сразу несколько делителей в рассматриваемом наборе.

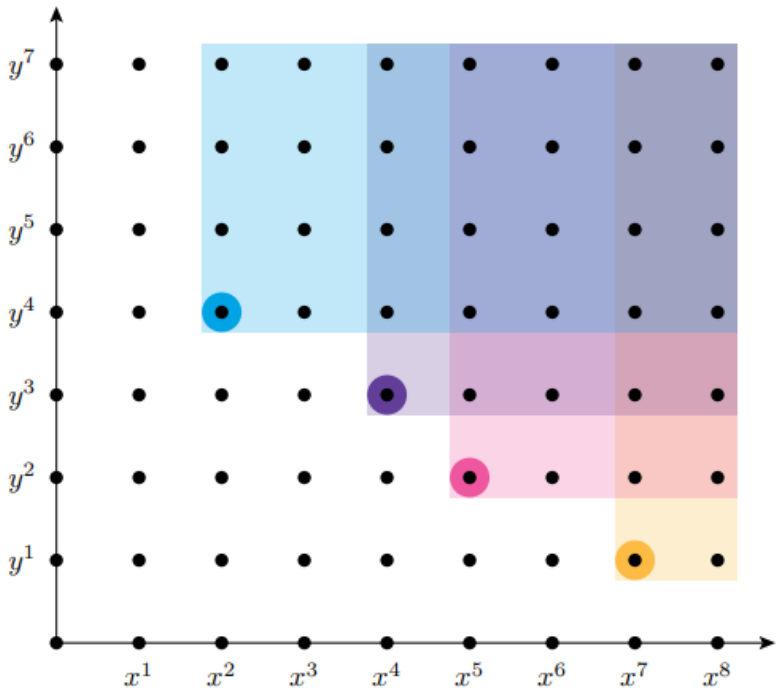

Рисунок 2.2 — Коническое представление термов $[x^7y, x^5y^2, x^4y^3, x^2y^4]$.

Возможность выбрать сразу несколько делителей нарушает линейную природу деления. Отдавая предпочтения старшему из возможных делителей, относительно выбранного мономиального порядка, можно однозначно определить соответствие между термом – точкой на графике, и его делителем среди рассматриваемого набора – образующей точки конуса. Так для порядка $\prec deglex$ мы получаем следующее разделение 2.3.

При этом, в набор термов можно добавить новые элементы. Это соответствует пополнению базиса Гребнера в процессе его построения новым полиномом с новым старшим термом. Например, рассматриваемый набор можно дополнить термами x^6y^2, x^3y^4 2.4.

При этом следует заметить, что добавление терма x^3y^3 невозможно, без создания пересечения – т. е. без нарушения однозначности деления 2.5. Однако, рассматривая термы x^3y^3, x^3y^4, следует отдать предпочтение первому. Он дает б**о**льшее покрытие пространства, а следовательно – является делителем для б**о**льшего числа термов.

Описанное выше верно и для случев с большим количеством переменных, и соответствует разделениям на конусы в n-мерном пространстве, где n – число переменных.

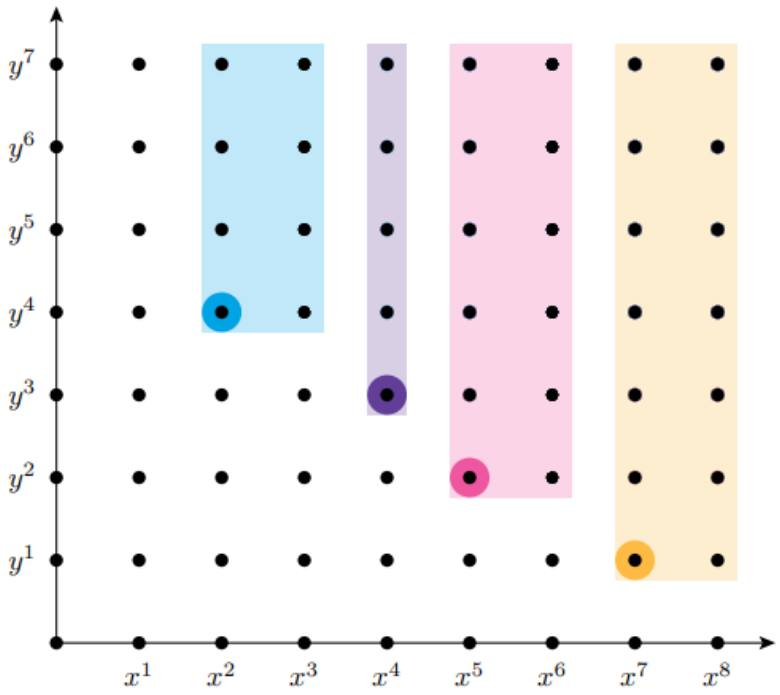

Рисунок 2.3 — Однозначное представление термов $[x^7y, x^5y^2, x^4y^3, x^2y^4]$.

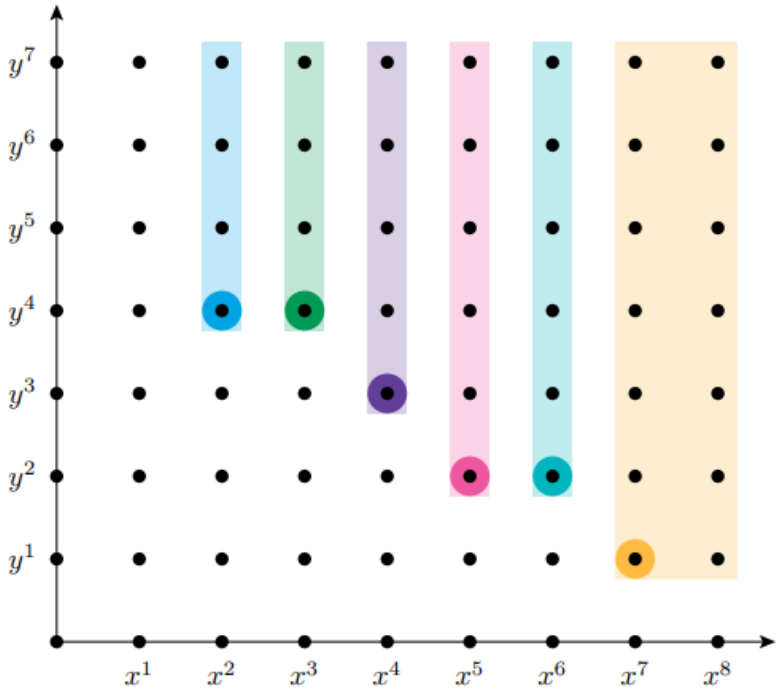

Рисунок 2.4 — Дополнение термов конусами x^6y^2, x^3y^4.

2.3.3 Примеры конических делений

Теорема 6 сводит задание линейного деление к заданию отображения $\mathcal{X} : U \to X$. Рассмотрим стандартные примеры делений.

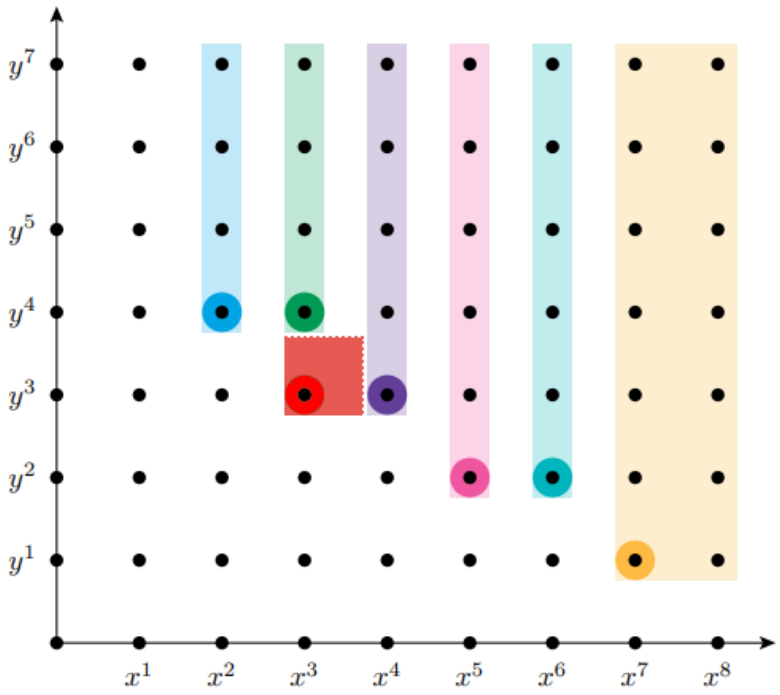

Рисунок 2.5 — Конус терма x^3y^3.

Пример 1. Деление Томаса $\mathcal{T}$ можно определить так. Пусть ∂_i — степень монома в связи с переменной x_i, тогда

$$x_i \in \mathcal{T}(u) \Leftrightarrow \partial_i u = \max_{v \in U} \partial_i v.$$

Проверим, что выполняется условие теоремы 6. Положим для краткости, что

$$\max_{v \in U} \partial_i v = d_i.$$

Пусть

$$w \in c(u, \mathcal{T}(u)) \cup c(v, \mathcal{T}(v)).$$

Тогда возможно 4 случая. Во-первых,

$$\partial_i w > \partial_i u, \quad \partial_i w > \partial_i v.$$

Это возможно только в том случае, когда

$$\partial_i u = \partial_i v = d_i.$$

Во-вторых,

$$\partial_i w = \partial_i u, \quad \partial_i w = \partial_i v,$$

что опять дает

$$\partial_i u = \partial_i v.$$

В третьих,

$$\partial_i w > \partial_i u, \quad \partial_i w = \partial_i v.$$

Первое соотношение возможно лишь тогда, когда $\partial_i u = d_i$. Но тогда

$$\partial_i v = \partial_i w > d_i,$$

что невозможно. По тем же причинам случай

$$\partial_i w = \partial_i u, \quad \partial_i w > \partial_i v$$

тоже невозможен. Следовательно, при всех i верно

$$\partial_i u = \partial_i v,$$

а это означает, что $u = v$. Таким образом, отображение

$$\mathcal{T} : U \to P_{fin}(X)$$

удовлетворяет условию теоремы 6 и поэтому задает некоторое линейное деление T:

$$uTw \Leftrightarrow w \in c(u,\mathcal{T}(u)),$$

которое называют делением Томаса.

Пример 2. Рассмотрим также деление Жане. Возьмем определение данного деления, например, представленное Яновичем [36].

Такое отображение также удовлетворяет условиям теоремы 6 и, следовательно задает некоторое линейное деление J:

$$uJw \Leftrightarrow w \in c(u,\mathcal{J}(u)),$$

Это деление основывается на следующем разделении переменных на мультипликативные и немультипликативные для мономов:

Определение 12. Переменная x_1 является мультипликативной по Жане для полинома $f \in F$, если $deg_1(lm(f)) = max\{deg_1(lm(g))\}$, где $g \in G$.

В общем же случае, где порядок переменной $i > 1$, она является мультипликативной, если

$$deg_i(lm(f)) = max\{deg_i(lm(g))\}|g \in [d_1, \ldots, d_{i-1}]$$

где,

$$[d_1, \ldots, d_{i-1}] = f \in F|d_j = deg(lm(f)), 1 <= j <= i$$

При этом предлагаемое разделение переменных на мультипликативные и немультипликативные не является инвариантным, так как зависит от выбранного порядка. Это разделение можно представить в виде графа деления, имеющего вид бинарного дерева. Будем называть такие деревья *деревьями Жане*.

Рассмотрим пример построения такого дерева 2.6 для случая трех переменных. Начнем построение с узла, в котором степени всех переменных равны 0, и текущий фокус направлен на первую переменную. Из каждого узла можно постоить максимум два перехода. Переход влево будет означать переход фокуса к следующей по порядку переменной, а переход вправо – повышение степени переменной в фокусе. На указанном рисунке приведен пример конкретного дерева, построенного для задачи . На рисунке 2.6 опущена часть узлов и ветвей дерева, где не происходит разветвления.

Для монома из числа старших мономов $u \in lmF$, зададим v, w такие, что $w = uv$. Тогда, если v состоит только из мультипликативных относительно u по Жане переменных, то u будет являться делитем w по Жане.

Пример 3. Еще одним примером является деление Поммаре. Оно отличается другим условием разграничения мультипликативных и немультипликативных переменных.

Для монома $u = x_1^{d_1} \dots x_k^{d_k}$, где $d_k > 0$, мультипликативными считаются такие переменные x_j, где $j > k$.

В остальном деление Поммаре также задает отображение, удовлетворяющее условие теоремы 6. Можем обозначать такое деление литерой P:

$$uPw \Leftrightarrow w \in c(u,\mathcal{P}(u)),$$

Разделение мультипликативных и немультипликативных переменных ждя деления Поммаре, также как и для деления Жане, может отображаться через построение графа деления, имеющего вид бинарного дерева.

2.3.4 Сравнение делений

Поскольку имеется много различных делений, естественным образом возникает вопрос об их сравнении.

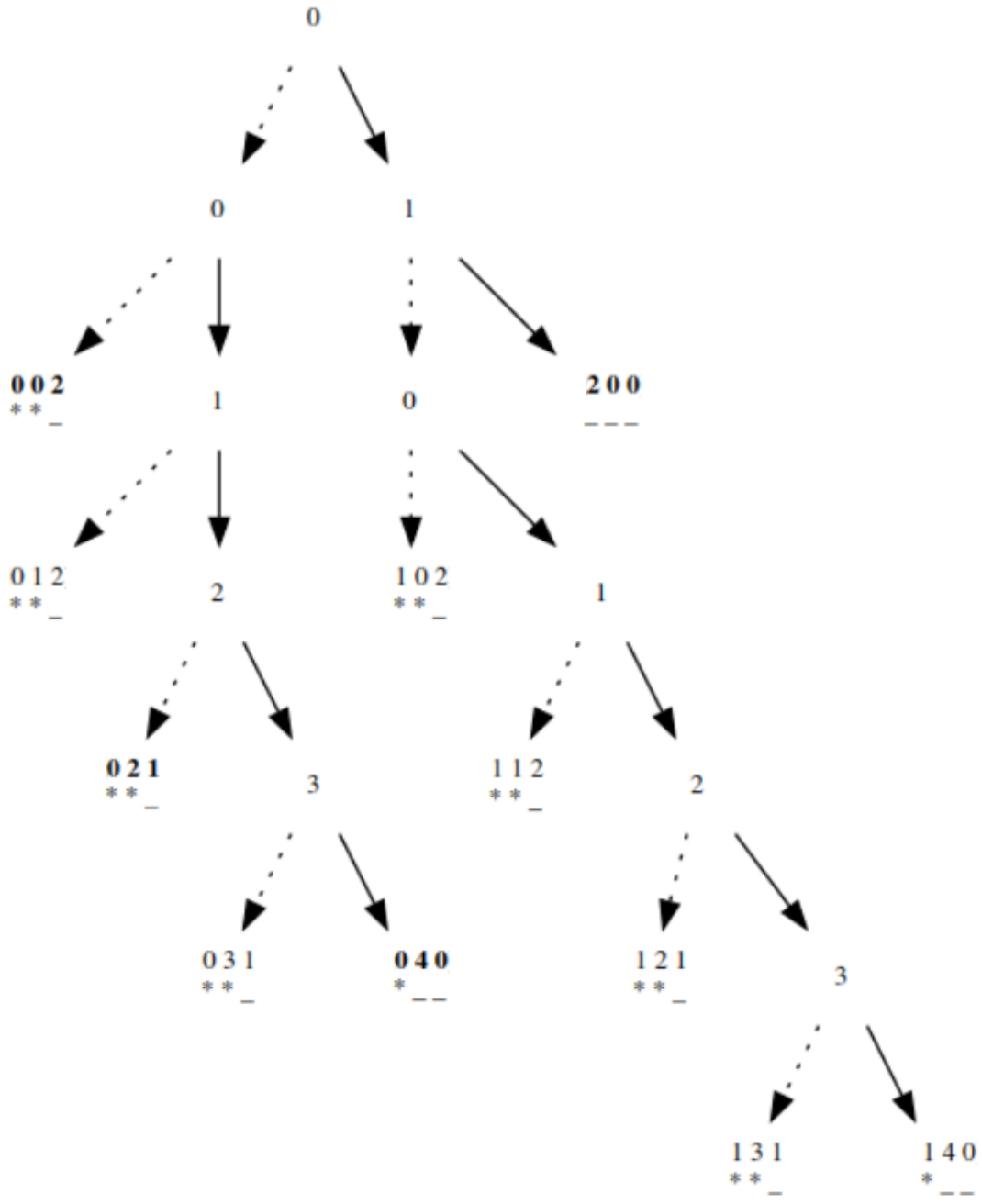

Рисунок 2.6 — Дерево Жане от трех переменных для задачи *diablo*.

Определение 13. Пусть L, L' — два линейных деления на $U \times M$. Будем говорить, что

$$\mathrm{L} \geqslant \mathrm{L}',$$

если $u\mathrm{L}w$ верно всякий раз, как верно $u\mathrm{L}'w$.

Пример 4. Среди линейных делений имеется тривиальное деление O, для которого

$$\mathscr{X}(u) = \varnothing.$$

В этом случае на u делится только сам моном u. Поскольку для любого деления верно $u\mathrm{L}u$, то $u\mathrm{L}w$ верно всякий раз, как верно uOw (т.е. только при $w = u$). Поэтому

$$\mathrm{L} \geqslant O.$$

Теорема 7. Пусть L, L' — два конических деления на $U \times M$. Тогда следующие условия эквиваленты:

1. $L' \leqslant L$,
2. для любой $u \in U$ и $w \in M$ верно: $u\mathrm{L}'w \Rightarrow u\mathrm{L}w$,
3. для любой $u \in U$ верно: $\mathcal{X}'(u) \subseteq \mathcal{X}(u)$

Доказательство. Пусть $\mathrm{L} \geqslant \mathrm{L}'$ заданы при помощи конусов. Тогда

$$w \in c(u, \mathcal{X}(u))$$

верно всякий раз, как верно

$$w \in c(u, \mathcal{X}'(u))$$

Это означает, что

$$\mathcal{X}'(u) \subseteq \mathcal{X}(u).$$

Таким образом,

$$\mathrm{L}' \leqslant \mathrm{L} \Leftrightarrow \mathcal{X}'(u) \subseteq \mathcal{X}(u).$$

□

Замечание. Ранее сравнения делений не рассматривались. Все рассматриваемые деления не только положительны, они отделены от нуля делением Томаса:

$$\mathrm{L} \geqslant \mathcal{T}.$$

Это следует из предл. 15 и 16 в [35] и теоремы 7.

2.3.5 Характеристические петли

Одной из важнейших структур, связанных с коническим делением, является петля.

Определение 14. Конечная последовательность мономов из U называется путем на U. Если среди этих мономов имеются повторяющиеся, то говорят, что путь содержит петли.

Определение 15. Путь $u, u', u'', \ldots$ на U будем называть характеристическим для конического деления L, если на каждом звене можно указать такую переменную $x^{(k)} \notin \mathcal{X}(u^{(k)})$, что $u^{(k+1)} \mathrm{L} x^{(k)} u^{(k)}$.

Поскольку множество U конечно, отсутствие петель означает, что всякий характеристический путь обрывается.

Пример 5. Рассмотрим одно звено характеристического пути для деления Томаса. Условие $u' T x u$ означает, что

$$xu = wu',$$

где моном w образован переменными из $\mathcal{T}(u')$. Если w содержит $y \neq x$, то

$$\partial_y u' < \partial_y u \leqslant \max_{v \in U} \partial_x v.$$

По определению деления Томаса это означает, что $y \notin \mathcal{T}(u')$, что невозможно. Если w содержит x, то

$$\partial_x u' \leqslant \partial_x u.$$

По определению 15

$$x \notin \mathcal{T}(u),$$

а согласно определению деления Томаса это эквивалентно тому, что

$$\partial_x u < \max_{v \in U} \partial_x v.$$

Следовательно,

$$\partial_x u' \leqslant \partial_x u < \max_{v \in U} \partial_x v.$$

По определению деления Томаса это означает, что $x \notin \mathcal{T}(u')$ и поэтому и x не входит в w. Поэтому $w = 1$ и

$$u' = xu.$$

Таким образом, всякий характеристический путь для деления Томаса представляет собой последовательность

$$u, xu, x'xu, \ldots,$$

среди элементов которой не может быть равных. Деление Томаса не дает характеристических петель.

Замечание. Все конические деления, рассматриваемые далее, не имеют характеристических петель. Это — очень важное условие появляется в [35] в определении 29. Там предлагается называть деление непрерывным, если оно не дает характеристических петель.

Интересно заметить, что линейные деления никак не связаны с мономиальным порядком на M. По существу это — вторая структура, существующая независимо от мономиального порядка.

2.4 Инволютивные базисы

2.4.1 Инволютивный базис

Убедившись в том, что линейные деления существуют и указав конструктивный способ их построения вернемся к задаче 1.

Пусть задан идеал $\langle G\rangle$ и многочлен f. Если на $\operatorname{lm} G \times M$ задано коническое деление L, то алгоритм Евклида позволяет найти остаток r от деления f на $\operatorname{lm} G$, причем

$$f - r \in J.$$

Если $r = 0$, то $f \in J$. Обратное верно, если базис G — инволютивный базис.

Определение 16. Множество $G \in P_{fin}(k[x_1, \ldots, x_n])$ называется базисом Гребнера идеала J, если оно порождает этот идеал и для любого $f \in J$ найдется такой элемент $u \in \operatorname{lm} G$, что

$$\operatorname{lm} u | \operatorname{lm} f.$$

Определение 17. Множество $G \in P_{fin}(k[x_1, \ldots, x_n])$ называется инволютивным базисом идеала J, если оно порождает этот идеал и для любого $f \in J$ найдется такой элемент $u \in \operatorname{lm} G$, что

$$\operatorname{lm} u \mathrm{L} \operatorname{lm} f.$$

Поскольку

$$u \mathrm{L} w \Rightarrow u | w,$$

всякий инволютивный базис является базисом Гребнера, но не наоборот.

Теорема 8. Если G — инволютивный базис идеала $\langle G\rangle$, то остаток от деления f на G равен нулю тогда и только тогда, когда $f \in \langle G\rangle$.

2.4.2 Цепной критерий Гердта-Блинкова

Гердт и Блинков [35] предложили следующий критерий: базис G является инволютивным в том и только в том случае, когда остаток от любого элемента вида xg, где $x \in X$ и $g \in G$, равен нулю. Мы будем называть его цепным критерием, поскольку он заменяет в этой теории цепной критерий Бухбергера.

Условие 1 (цепное условие Гердта-Блинкова)**.** Остаток от любого элемента вида xg, где $x \notin \mathcal{X}(\operatorname{lm} g)$ и $g \in G$, равен нулю.

Тонкость здесь состоит в том, что помимо этого условия, нужно еще одно про характеристические петли. Заметим, что при выполнении цепного условия можно строить характеристические пути на U, начав с любого монома $u \in U$ следующим образом. Если $\mathcal{X}(u) \neq X$, возьмем $x \notin \mathcal{X}(u)$. В силу условия 1 остаток от xf, $\operatorname{lm}(f) = u$, должен быть равен нулю, поэтому существует такой моном $u' \in U$, что $u'\mathrm{L}(xu)$. Если $\mathcal{X}(u') \neq X$, то возьмем $x' \notin X(u)$. Тогда найдется такой элемент $u'' \in U$, что $u''\mathrm{L}(x'u')$ и т.д. В результате получится путь $u,u',u'',\ldots$, который в соответствии с определением 15 мы условились называть характеристическим. Если на каком-то шаге мы придем к $\mathcal{X}(u^{(m)}) = X$, то будем говорить, что характеристический путь $u,u',\ldots,u^{(m)}$ обрывается.

Условие 2 (путевое)**.** Всякий характеристический путь обрывается и не имеет самопересечений.

Отсутствие петель означает, что всякий характеристический путь должен обрываться, поэтому выполнение условий 1 и 2 возможно лишь в том случае, когда среди мономов U имеются вершины с $\mathcal{X} = X$ — концы характеристических путей.

Фундаментальная теорема состоит в следующем.

Теорема 9 (цепной критерий Гердта-Блинкова)**.** Пусть идеал J порожден многочленами множества $G \in P_{fin}(k[x_1, \ldots, x_n])$ и пусть $U = \operatorname{lm} G$ не содержит равных элементов. Пусть на $U \times M$ введено коническое деление L и пусть выполнены цепное условие 1 и путевое условие 2. Тогда G — инволютивный базис идеала J.

2.4.3 Доказательство цепного критерия

Лемма 2. Пусть выполнены условия теоремы 9. Тогда любой элемент вида mf, $f \in G$ и $m \in M$, можно представить в виде

$$mf = cwg + \sum g_i w_i, \tag{2.1}$$

где

1. $g, g_1, \cdots \in G$, $w, w_1, \cdots \in M$, $c \in k$,
2. $\operatorname{lm}(g) \mathrm{L} m \operatorname{lm}(f)$,
3. $w \operatorname{lm}(g) > \operatorname{lm}(g_i) w_i$.

Замечание. Эта лемма содержится в доказательстве теоремы 46 в [35].

Доказательство. Если

$$\operatorname{vars} m \subseteq \mathcal{X}(\operatorname{lm} f),$$

то $m \operatorname{lm} f$ принадлежит конусу $c(\operatorname{lm}(f), \mathcal{X}(\operatorname{lm} f))$ и поэтому $m \mathrm{L} m \operatorname{lm} f$. Значит, (2.1) верно при $g = f$ и $g_i = 0$.

Если же

$$\operatorname{vars} m \not\subseteq \mathcal{X}(\operatorname{lm} f),$$

то в m содержится переменная

$$x \notin \mathcal{X}(\operatorname{lm} f).$$

В силу условия 1 остаток от деления xf на G равен нулю, поэтому, в силу теоремы 2, многочлен xf можно представить в виде редуцируемой сумму

$$xf = cwg + \sum g_i w_i,$$

где

1. $\operatorname{lm}(g) \mathrm{L} x \operatorname{lm}(f)$,
2. $w \operatorname{lm}(g) > \operatorname{lm}(g_i) w_i$.

Но тогда

$$mf = g\frac{cwm}{x} + \sum g_i \frac{w_i m}{x}.$$

Поскольку x содержится в m, выражение $\frac{m}{x}$ является мономом. Если все содержащиеся в нем переменные принадлежат $\mathcal{X}(\operatorname{lm} g)$, то есть

$$\operatorname{vars} \frac{wm}{x} \subseteq \mathcal{X}(\operatorname{lm} g),$$

то

$$\operatorname{lm}(g) w \frac{m}{x} \in c(\operatorname{lm} g, \mathcal{X}(\operatorname{lm} g))$$

и поэтому

$$\operatorname{lm} g \mathrm{L} m \operatorname{lm} f.$$

Таким образом, полученное представление для mf обладает всеми требуемыми свойствами.

Если же

$$\operatorname{vars} \frac{wm}{x} \not\subseteq \mathcal{X}(\operatorname{lm} g),$$

то в $\frac{wm}{x}$ входит переменная

$$x' \notin \mathcal{X}(\operatorname{lm} g).$$

В силу условия 1 остаток от $x'g$ при делении на G равен нулю, поэтому $x'g$ можно представить в виде редуцируемой сумму

$$x'g = c'w'g' + \sum g'_i w'_i,$$

где

1. $\operatorname{lm}(g') \mathrm{L} x' \operatorname{lm}(g)$,
2. $w' \operatorname{lm}(g') > \operatorname{lm}(g'_i) w'_i$.

Но тогда

$$mf = g'\frac{c'cw'wm}{xx'} + \sum g''_i \frac{c'cw'_i wm}{xx'}.$$

Если

$$\operatorname{vars} \frac{ww'm}{xx'} \subseteq \mathcal{X}(\operatorname{lm} g'),$$

то лемма доказана.

В противном случае, мы можем продолжить описанный процесс. При этом мы получим путь

$$\operatorname{lm} f, \operatorname{lm} g, \quad \operatorname{lm} g', \quad \ldots,$$

элементы которого подчинены следующему условию:

$$\operatorname{lm} g \mathrm{L} x \operatorname{lm} f, \quad \operatorname{lm} g' \mathrm{L} x' \operatorname{lm} g, \quad \ldots$$

причем

$$x \notin \mathcal{X}(\operatorname{lm} f), x' \notin \mathcal{X}(\operatorname{lm} g), \ldots$$

Это означает, что описанный процесс возвращает нам характеристический путь, который в силу условия 2 обрывается. Следовательно, за конечное число шагов мы придем к представлению вида (2.1). □

Опираясь на лемму можно доказать теорему 9 следующим образом. Условимся для суммы

$$f = \sum h_i g_i$$

обозначать старший моном среди слагаемых как как

$$m(f) = \max_i \left(\operatorname{lm}(h_i) \operatorname{lm}(g_i)\right)$$

Если теорема 9 не верна, то множество всех сумм, остатки которых не равны нулю, не пусто и не содержит 0. Среди них выберем ту, для которой $m(f)$ минимально. Тогда

$$f = \sum h_i g_i = \sum c_i m_i g_i + \sum h'_i g_i,$$

где

1. $m_i \operatorname{lm}(g_i) = m$,
2. $m > \operatorname{lm}(g_i) \operatorname{lm}(h'_i)$.

В силу леммы 2

$$\sum c_i m_i g_i = \sum c'_i w_i g_i + \sum g_i h'_i,$$

1. $w_i \operatorname{lm} g_i = m$
2. $\operatorname{lm}(g_i) \mathrm{L} m$,
3. $m > \operatorname{lm}(g_i) \operatorname{lm}(h'_i)$.

Но тогда в силу теоремы 2 остаток от

$$\sum c'_i w_i g_i$$

равен нулю. В силу линейности остатка (теорема 3), остаток от

$$\sum g_i h'_i$$

равен остатку от f, то есть не равен нулю. Однако,

$$\mathrm{lm}(g_i h_i') < m(f),$$

против предположения о минимальности m на f.

2.5 Пополнение базиса до инволютивного

2.5.1 Инволютивное деление

Множество всех конических делений на $U \times M$ будем обозначать как $\mathrm{div}(U)$. Инволютивное деление $\mathcal{L}$ — это произвольное отображение

$$\mathcal{L} : P_{fin}(M) \to \prod_{U \in P_{fin}(M)} \mathrm{div}(U).$$

Например, деление Томаса определено на любом $U \times M$ и в этом смысле может рассматриваться как отображение

$$\mathcal{T} : P_{fin}(M) \to \prod_{U \in P_{fin}(M)} \mathrm{div}(U).$$

Замечание. В предложенной формулировке нетрудно заметить конструкцию из Красной книжечки Мамфорда. Однако мы сочли чрезмерным говорить здесь о функторах. Поскольку аксиомы 1-4 из оригинального определения 5, предложенного в 2020 г., вошли в определения конического деления, все отличие нашего определения инволютивного деления от определения 5 сводится к 5-ой аксиоме. Как мы сейчас покажем, для доказательства существования инволютивного базиса она не нужно. Нужно другое, более тонкое условие о существовании пополнения.

2.5.2 Пополнение базиса

Определение 18. Множество $U \in P_{fin}(M)$ будем называть полным относительно инволютивного деления $\mathcal{L}$, если

$$\langle U \rangle \cap M = \bigcup_{u \in U} c(u, \mathcal{X}_{\mathcal{L}}(U))$$

Определение 19. Пополнением подмножеств множества M будем называть правило, по которому этим подмножествам ставятся в соответствия некоторое другие подмножества этого множества

$$F : P_{fin}(M) \to P_{fin}(M),$$

если $\forall U \in P_{fin}(M)$ верно

1. $F(F(U)) = F(U)$
2. $U \subseteq F(U)$

Множество $F(U)$ будем называть пополнением множества U.

Определение 20. Пополнением относительно деления $\mathcal{L}$ будем называть пополнение $F_{\mathcal{L}}$, удовлетворяющее следующим свойствам

1. $F_{\mathcal{L}}(U)$ — полное относительно $\mathcal{L}(U)$,
2. $\langle U \rangle = \langle F_{\mathcal{L}}(U) \rangle$.

Замечание. В [35] предлагается называть эту операцию $\mathcal{L}$-замыканием, см. определение 23. Однако замыкание — это пополнение, образ которого должен быть множеством всех замкнутых множеств. Фактически, это означает, что должны выполняться аксиомы типа $F(\cup U) = \cup F(U)$ и $F(\cap U) = \cap F(U)$. В нашем случае они не выполняются.

Определение 21. Инволютивное деление, относительно которого можно ввести пополнение, называется нетеревым.

Пример 6. Пусть $U \in P_{fin}(M)$. Положим для краткости

$$d_x = \max_{u \in U} \partial_x u$$

и примем, что $\tilde{U}$ — этом множество мономов v, для каждого из которых можно найти такой моном $u \in U$, что

$$\partial_x(u) \leqslant \partial_x(v) \leqslant d_x, \quad \forall x \in X.$$

Докажем, что это множество полное. Пусть моном $m \in \langle \tilde{U} \rangle$. Тогда найдется такой $u \in U$, что $u|m$. Тогда

$$\partial_x(u) \leqslant \partial_x(m).$$

Пусть X' — множество тех переменных, для которых

$$\partial_x m < d_i,$$

а X'' — дополнительное к нему множество. Тогда

$$v = \prod_{x \in X'} x^{\partial_x(m)} \prod_{x \in X''} x^{d_x} \in \tilde{U}$$

По определению деления Томаса на $\tilde{U} \times M$

$$\mathcal{X}(v) = X'',$$

поэтому v делит m в смысле деления $\mathcal{T}(\tilde{U})$. Поэтому $\tilde{U}$ — полное множество. Сопоставление $F(U) = \tilde{U}$ является пополнением деления Томаса, поскольку $\langle \tilde{U} \rangle = \langle U \rangle$.

2.5.3 Существование инволютивного базиса

Процесс пополнения не является конструктивным, но его можно использовать для доказательства существования инволютивного базиса.

Итак, начнем с G. Пусть $U = \operatorname{lm} G$ и $\tilde{U}$ его пополнение относительно деления $\mathcal{L}$, тогда

$$\langle U \rangle = \langle \tilde{U} \rangle.$$

Элементы $\tilde{U}$, которые не принадлежат U, получаются из элементов U путем умножения на некоторые мономы. Умножая соответствующие элементы G на эти мономы, мы найдем такое множество $\tilde{G} \in P_{fin}(k[x_1, \ldots, x_n])$, что $\tilde{U} = \operatorname{lm} \tilde{G}$ и

$$\langle G \rangle = \langle \tilde{G} \rangle.$$

Если на $\tilde{G}$ выполняется цепное условие 1, то $\tilde{G}$ — инволютивный базис исходного идеала. В противном случае можно взять такие $x \in X$ и $g \in \tilde{G}$,

что остаток xg относительно $\tilde{G}$ не равен нулю. Добавим этот остаток в $\tilde{G}$ и получим G', причем

$$\langle G\rangle = \langle \tilde{G}\rangle = \langle G'\rangle.$$

Старший член этого остатка, скажем, u' не делится на $\tilde{U}$. Но это множество — полное, поэтому u' не принадлежит $\langle U\rangle$, то есть строго

$$\langle \tilde{U}\rangle \subset \langle U'\rangle.$$

Пусть $\tilde{U}'$ — пополнение U', тогда

$$\langle \tilde{U}\rangle \subset \langle \tilde{U}'\rangle.$$

Тем же путем, что и выше построим $\tilde{G}'$, тогда

$$\langle G\rangle = \langle \tilde{G}\rangle = \langle \tilde{G}'\rangle.$$

Если на $\tilde{G}'$ выполняется цепное условие 1, то $\tilde{G}'$ — инволютивный базис исходного идеала. В противном случае продолжим процесс расширения дальше. В итоге мы получим строго монотонную последовательность

$$\langle U\rangle \subset \langle U'\rangle \subset \langle U''\rangle \subset \dots$$

Поскольку полиномиальное кольцо нетерово, эта последовательность должна обрываться. Следовательно, описанный процесс должен завершиться за конечное число действий построением инволютивного базиса исходного идеала.

Теорема 10. Пусть идеал J порожден многочленами множества $G \in P_{fin}(k[x_1,\dots,x_n])$ и пусть $U = \operatorname{lm} G$ не содержит равных элементов. Пусть на M введено инволютивное деление L, которое не имеет характеристических петель (условие 2). Тогда существует инволютивный базис идеала J относительно этого деления, который содержит G как подмножество.

2.6 Вычисление инволютивного базиса

2.6.1 Алгоритм построения инволютивного базиса

Устранить неконструктивный элемент — пополнение — можно следующим образом.

Пусть задано множество $G \in P_{fin}(k[x_1, \ldots, x_n])$ и пусть $U = \operatorname{lm} G \in P_{fin}(M)$, то есть среди элементов $\operatorname{lm} G$ нет равных. Пусть на $U \times M$ задано некоторое коническое деление L, которое удовлетворяет условию 2 об отсутствии петель.

Станем вычислять остатки многочленов из множества

$$\partial G = \{xg \quad \forall x \in X, \forall g \in G\}$$

Если все остатки равны, то есть выполнено цепное условие 1, то G — инволютивный базис идеала $\langle G \rangle$.

В противном случае мы остановим вычисление на первом ненулевом остатке и добавим этот остаток к G, тем самым мы получим новый базис G' того же идеала. При этом $\operatorname{lm} G'$ получит еще один элемент, который заведомо не совпадает с остальными.

Пусть на $U' \times M$ определено какое угодно коническое деление L$'$, которое удовлетворяет условию 2 об отсутствии петель. Проделав все вышеперечисленное с G', мы или убедимся в том, что G' — инволютивный базис $\langle G \rangle = \langle G' \rangle$ относительно деления L$'$, или опять расширим G' до G'' и т.д.

Этот процесс или возвращает инволютивный базис, или продолжается до бесконечности. Последнее невозможно, поскольку инволютивный базис существует, а сам предложенный алгоритм устроен таким образом, что перебирает все многочлены, которые могут попасть в инволютивный базис.

2.6.2 Сужение линейного деления

Пусть $\mathrm{L} \in \mathrm{div}(U)$ и $V \subset U$, тогда сужение L на $V \times M$ можно рассматривать как деление из $\mathrm{div}(V)$, поэтому

$$V \subset U \Rightarrow \mathrm{div}(U)|_V \subset \mathrm{div}(V).$$

Разумеется, при таком сужении два различных элемента $\mathrm{div}(U)$ могут оказаться равными.

Пример 7. Сужение деления Томаса на $U \times M$ не совпадает со своим сужением на $V \times M$. В самом деле, положим

$$d_i(W) = \max_{v \in W} \partial_i v,$$

тогда

$$d_i(U) \geqslant d_i(V). \tag{2.2}$$

По определению деление Томаса на $U \times M$ задается при помощи отображения $\mathcal{T}_U$, заданного формулой

$$x_i \in \mathcal{T}_U(u) \Leftrightarrow \partial_i u = d_i(U) \quad \forall u \in U.$$

Сужение этого деления на $V \times M$ задается при помощи отображения $\mathcal{T}_U|_V$, заданного той же формулой

$$x_i \in \mathcal{T}_U|_V(u) \Leftrightarrow \partial_i u = d_i(U),$$

где u пробегает только $V \subset U$. Деление Томаса на $V \times M$ задается при помощи отображения $\mathcal{T}_V$, заданного формулой

$$x_i \in \mathcal{T}_V(u) \Leftrightarrow \partial_i u = d_i(V) \quad \forall u \in V.$$

В силу (2.2) при некоторых i степени $\partial_i u$ могут не дотягивать до $d_i(U)$, поэтому

$$\mathcal{T}_U|_V(u) \subseteq \mathcal{T}_V(u).$$

Поэтому

$$T_U|_V \leqslant T_V,$$

то есть сужение деления Томаса всегда меньше самого деления Томаса.

Глава 3. Комплекс программ для построения инволютивных базисов полиномиальных идеалов

3.1 Система GInv

3.1.1 История и архитектура системы GInv

GInv - это программная система, предназначенная для нахождения инволютивного базиса идеалов над кольцами многочленов. Она основана на вариации алгоритма Бухбергера для нахождения базиса Гребнера для полиномиальных идеалов. Гердт, Жарков и Блинков предложили новую версию алгоритма Бухбергера, основанную на инволютивном делении, которая подробно обсуждалась в первой главе диссертации Блинкова. В 2000-х годах для реализации этого алгоритма было создано программное обеспечение GInv, которое использовалось для решения ряда задач математической физики. Недавно это программное обеспечение было обновлено, протестировано и модифицировано для использования в более широких областях.

Ядро GInv написано в виде библиотечного пакета на языке программирования C++ [7]. В нем используются библиотеки GMP для работы с динамическими структурами данных, такими как двоичные или красно-черные деревья. Это позволяет выполнять вычисления с целыми числами произвольной точности и легко управлять перераспределением памяти. Интерфейс для ядра GInv разработан в виде языкового модуля Python3. Для этого использовался язык Cython, служащий в качестве промежуточного слоя между языками C++ и Python. Это сделало использование GInv совместимым со средой Jupyter Notebook. Исходный код для GInv был размещен в открытом доступе [14].

В дальнейшем система GInv была несколько раз обновлялась, а также получила частичную реализацию в виде самостоятельного кода на Python. Новая версия GInv была протестирована в том числе на реальной задаче – задаче вычисления узлов и весов кубатурных формул на сфере [24]. Эта задача приводит к очень сложной системе нелинейных уравнений, которую ранее можно было решить только численно. Экономное отношение к памяти в системе GInv

расширило вычислительные возможности исследователей, что позволило найти базис Гребнера для данной задачи, и с помощью него решить эту проблему аналитически [37].

Программное обеспечение GInv также может быть использовано в математической логике, используя многочлены Жегалкина. Многочлены Жегалкина представляют собой функции в булевой алгебре и принимают форму многочленов от поля Z_2. Программное обеспечение GInv может быть адаптировано для использования модульной арифметики и находить инволютивные базисы, следовательно и базисы Гребнера, для идеалов над кольцами Z_2. Таким образом, система GInv может быть использована для решения задач SAT или поиска точных решений.

Текущая обновленная версия системы GInv работает на чистом Python, версии Python3.1 и более поздней. Для этого требуется установка библиотек NumPy и SymPy для численных и символьных вычислений, а также библиотеки GraphViz, которая используется для визуализации в некоторых случаях использования. Использование библиотеки GraphViz опционально, и может быть опущено – в таком случае не будет выполняться визуализация построения деревьев Жане, однако основной функционал будет сохранен.

Основные классы системы – Monom, Poly и GB - отражают суть мономов, или, скорее, термов, многочленов и базиса Гребнера, соответственно. Их основные компоненты и взаимосвязи отражены на диаграмме классов 3.1.

Класс Monom наследует коллекцию tuple, неизменяемый набор элементов, который является стандартным для языка Python. Коллекция содержит целые положительные числа, которые являются степенями переменных, включенных в терм. Для него реализовано несколько различных функций перевода в строковый формат, включая и встроенные в стандартную систему Python, такие как $\_\_repr\_\_(self)$ для внутреннего представления и $\_\_str\_\_(self)$ для стандартного превращения в строку.

Класс Poly наследует другую стандартную коллекцию – list, список. Полином представляет собой набор из мономов, в свою очередь являющихся парой из Monom'а, то есть терма, и коэффициента.

Класс GB также наследует список, и включает в себя набор полиномов. Основная функция здесь – *algorithm2*, используемая для нахождения базиса в степенном-лексикографическом ($\prec deglex$) порядке.

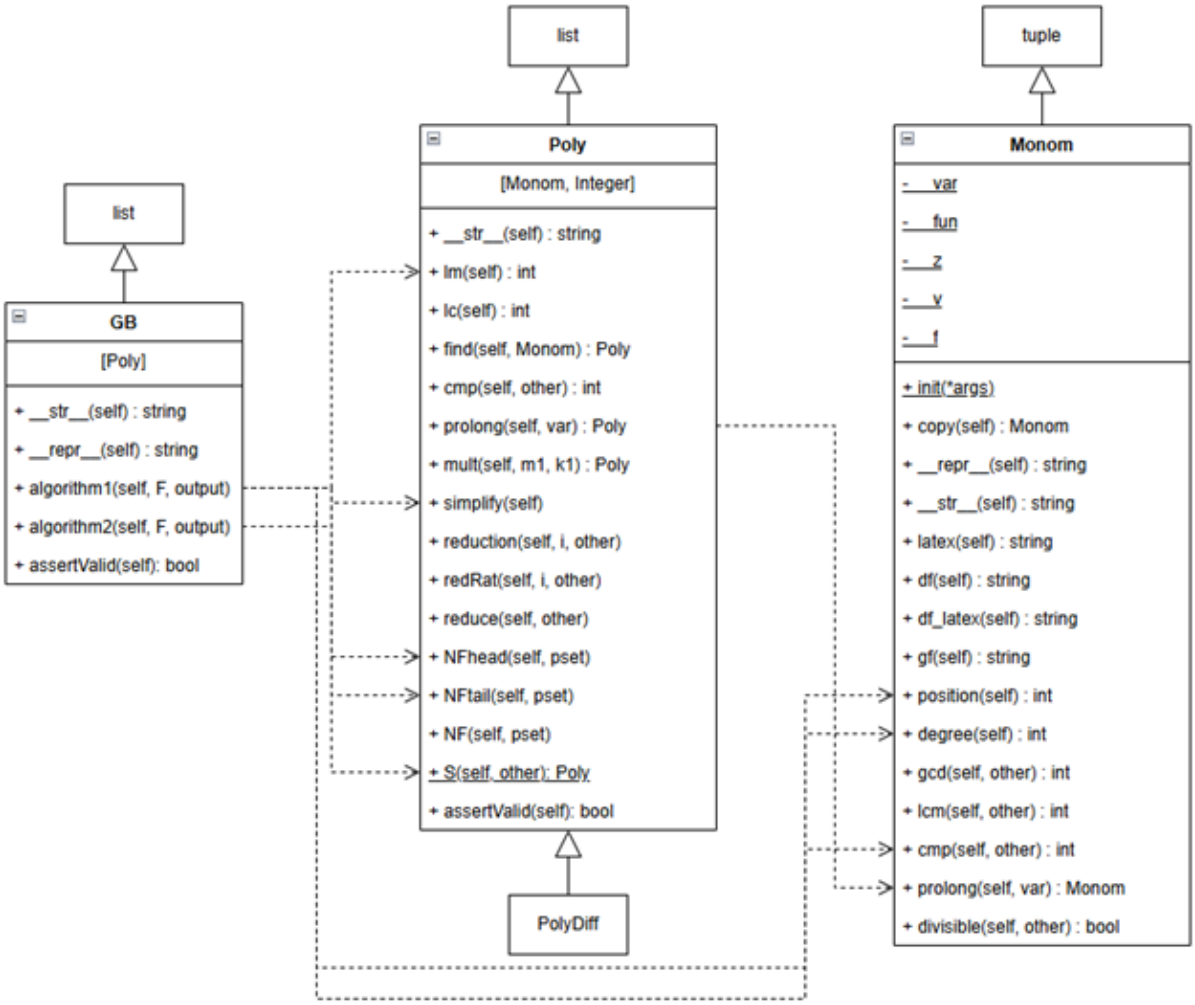

Рисунок 3.1 — Диаграмма основных классов системы GInv.

3.1.2 Пример работы системы GInv для решения систем дифференциальных уравнений

Система GInv также была адаптирована для решения систем дифференциальных уравнений с помощью специального класса PolyDiff, который представляет собой многочлен от функций, в котором вместо степеней сохраняются порядки производных. Такое расположение позволяет использовать тот же механизм хранения данных в памяти, что и для обычных многочленов.

Так, например, система GInv была использована для решения широко применяемого в гидродинамике уравнения Кортевега-де Фриза, а также менее известное уравнение Линя-Рейснера-Цзяня.

Приведем пример использования GInv для КдФ:

Уравнение КдФ можно представить в следующем виде, добавив условие на наличие высших симметрий:

$$u_t + 6u_1u + u_3 = 0$$

$$u_\varepsilon - F(t, x, u_1, u_2, \ldots, u_5) = 0$$

При этом используем обозначение u_1 для отображения u_x, т.е. производной u по x. Тогда u_3 есть u_{xxx}

Выведем полные производные от F по x и t.

$$\frac{dF}{dt} = F_t + F_u(6u_1u + u_3) + \sum F_{u_i}(6u_1u + u_3)_i$$

$$\frac{dF}{dx} = F_x + F_u u_1 + \sum F_{u_i} u_{i+1}$$

Условие для случая, когда полученная система будет совместна, а значит и интегрируема, можно записать через S-полином, например на лексикографическом порядке ($\prec lex$). Т.е. $t \prec x$ для данной системы.

$$(u_t + 6u_1u + u_3)_\varepsilon - u_{\varepsilon t} + \frac{d}{dt}F = 0$$

По правилу дифференцирования Лейбница, выведем из этого уравнения то, что производная по u_ε входит в S-полином и условие интегрируемости линейно.

$$6uu_{\varepsilon 1} + 6u_1u_\varepsilon + u_{\varepsilon 3}\frac{d}{dt}F = 0$$

Если же мы заменим u_ε на F, явно выражая через полные производные от F, заданные выше:

$$6u\frac{d}{dx}F + 6u_1F + \frac{d^3}{dx^3}F + \frac{d}{dt}F = 0$$

Далее, так как F не зависит от производных от шестого порядка и выше, можно вывести систему линейных уравнений на F.

Выполнив эту аналитическую подготовку, мы теперь можем использовать нотацию библиотеки SymPy для переноса задачи в программную систему. Сначала объявим используемые переменные и функции.

```
eps, t, x = sympy.symbols('e, t, x', real=True)
U = sympy.Function('u')(eps, t, x)
```

```
u, ut, u1, u2, u3, u4, u5
    = sympy.symbols('u, ut, u1, u2, u3, u4, u5', real=True)
F = sympy.Function('f')(t, x, u, ut, u1, u2, u3, u4, u5)
```

Далее дадим определение целевому уравнению, для данной задачи – уравнение КдФ.

```
eq1 = U.diff(t) +6*U*U.diff(x) + U.diff(x,3)
```

Компиляция этого кода использует пакет sympy что дает математической формуле уравнения привычный вид 3.2.

```
In [3]: eq1 = U.diff(t) + 6*U*U.diff(x) + U.diff(x, 3)
        eq1
```

Out[3]: $$6u(\varepsilon,t,x)\frac{\partial}{\partial x}u(\varepsilon,t,x)+\frac{\partial}{\partial t}u(\varepsilon,t,x)+\frac{\partial^3}{\partial x^3}u(\varepsilon,t,x)$$

Рисунок 3.2 — Объявление уравнения.

И, наконец, перейдем к использованию программной системы GInv.

Вначале инициализируем систему GInv, аналогично объявлению используемых переменных и функций для SymPy .

```
Monom.cmp = Monom.TOPdeglex

var = 't, x, u, ut, u1, u2, u3, u4, u5'.split(', ')
fun = ['f']
Monom.init(var, fun)
df = PolyDiff.df
diff2poly = PolyDiff.diff2poly
var, fun = PolyDiff.init()
t, x, u, ut, u1, u2, u3, u4, u5 = var
F = fun[0]
```

При этом воспользуемся статической функцией diff2poly, описанной классе PolyDiff – классе для полиномов, состоящих из мономов от переменных **и**

функций. В ней реализован процесс распознавания символьных формул, представленных в формате библиотеки SymPy, и их последующий перевод в формат данных, используемых в системе GInv. Для ее корректной работы необходимо вначале указать, какие переменные и математические функции будут составлять мономы в решаемой задаче.

Работа функции diff2poly выполняется рекурсивно. Выход из этой рекурсии происходит в случае вызова от одиночного монома, без дополнительных аргументов. В таком случае в формате PolyDiff напрямую формируется полином из одиночного монома.

При распознавании операций сложения (вычитания) или умножения, вначале рекурсивно выполняется перевод операндов в нужный формат, после чего выполняется распознанная операция. Возведение в степень работает аналогично, в то время как специальные функции обрабатываются отдельно.

В последнюю очередь распознается взятие производной. Для этого случая в формате PolyDiff составляется моном, во внутреннем представлении которого указывается соответствующее дифференцирование.

```
@staticmethod
def diff2poly(a):
# перевод из символьного выражения SymPy в дифференциальный полином
    if not a.args:
        return PolyDiff(a)
        # если a не содержит аргументов, переводим в дифф. полином
    else:
        if a.func == sympy.Add: # для функции сложения
            r = PolyDiff.diff2poly(a.args[0])
            # переводим первый аргумент в дифф. полином
            for s in a.args[1:]:
                r = r + PolyDiff.diff2poly(s)
                # и прибавляем все последующие,
                # также переводя их в дифф. полином
        elif a.func == sympy.Mul: # для функции умножения
            r = PolyDiff.diff2poly(a.args[0])
            # переводим первый аргумент в дифф. полином
            for s in a.args[1:]:
```

```
            r = r * PolyDiff.diff2poly(s)
            # и домножаем на все последующие,
            # также переводя их в дифф. полином
    elif a.func == sympy.Pow: # для функции степени
        r = PolyDiff.diff2poly(a.args[0])**a.args[1]
        # (первый аргумент в дифф. полином)^(второй аргумента)
    elif repr(a.func) in Monom._Monom__fun:
        # обработка специальных функций, заданных для мономов)
        r = PolyDiff(Monom(
                    pos=Monom._Monom__fun.index(repr(a.func))))
    else:
        assert a.func == sympy.Derivative
        # последний вариант - производная
        assert repr(a.args[0].func) in Monom._Monom__fun
        # проверяем - под производной стоит обрабатываемая функция?
        m = [0 for _ in Monom._Monom__var]
        # задаём список нулей по количеству объявленных переменных
        for v, d in a.args[1:]:
            m[Monom._Monom__var.index(repr(v))] = d
            # заполняем список значениями переменных
        r = PolyDiff(Monom(m,
            pos=Monom._Monom__fun.index(repr(a.args[0].func))))
        # переводим получаемый моном в дифф. полином
    return r
```

Результатом работы будет являться полином, представленный в формате PolyDiff. Как говорилось выше, его можно вывести как во внутреннем 3.3, так и во внешнем строковом 3.4 представлении.

Далее найдем базис Жане 3.5 – нормальную форму для систем линейных однородных дифференциальных уравнений в частных производных. Это необходимо для устранения присущей системе произвольности. Построение базиса Жане выполняется с помощью отдельного класса Janet, реализующего классическим алгоритм, адаптированный под представление данных, используемое в GInv.

```
In [17]: diff2poly(eqs[1])

Out[17]: [[[0;0 2 0 0 0 0 0 1], 3],
          [[0;0 1 1 0 0 0 0 1], 6*u1],
          [[0;0 0 2 0 0 0 0 1], 3*u1**2],
          [[0;0 1 0 1 0 0 0 1], 6*u2],
          [[0;0 0 1 1 0 0 0 1], 6*u1*u2],
          [[0;0 0 0 2 0 0 0 1], 3*u2**2],
          [[0;0 1 0 0 1 0 0 1], 6*u3],
          [[0;0 0 1 0 1 0 0 1], 6*u1*u3],
          [[0;0 0 0 1 1 0 0 1], 6*u2*u3],
          [[0;0 0 0 0 2 0 0 1], 3*u3**2],
          [[0;0 1 0 0 0 1 0 1], 6*u4],
          [[0;0 0 1 0 0 1 0 1], 6*u1*u4],
          [[0;0 0 0 1 0 1 0 1], 6*u2*u4],
          [[0;0 0 0 0 1 1 0 1], 6*u3*u4],
          [[0;0 0 0 0 0 2 0 1], 3*u4**2],
          [[0;0 1 0 0 0 0 1 1], 6*u5],
          [[0;0 0 1 0 0 0 1 1], 6*u1*u5],
          [[0;0 0 0 1 0 0 1 1], 6*u2*u5],
          [[0;0 0 0 0 1 0 1 1], 6*u3*u5],
          [[0;0 0 0 0 0 1 1 1], 6*u4*u5],
          [[0;0 0 0 0 0 0 2 1], 3*u5**2],
          [[0;0 1 0 0 0 0 1 0], 3],
          [[0;0 0 1 0 0 0 1 0], 3*u1],
          [[0;0 0 0 1 0 0 1 0], 3*u2],
          [[0;0 0 0 0 1 0 1 0], 3*u3],
          [[0;0 0 0 0 0 1 1 0], 3*u4],
          [[0;0 0 0 0 0 0 2 0], 3*u5],
          [[0;0 0 1 0 0 0 0 1], 3*u2],
          [[0;0 0 0 1 0 0 0 1], 3*u3],
          [[0;0 0 0 0 1 0 0 1], 3*u4],
          [[0;0 0 0 0 0 1 0 1], 3*u5]]
```

Рисунок 3.3 — Внутренне представление дифференциальных полиномов

Далее нам остается лишь вызвать от объекта класса Жане функцию построения базиса Гребнера 3.6.

```
In [18]: print(diff2poly(eqs[1]))

df(f, x, 2, u5)*3 + df(f, x, u, u5)*6*u1 + df(f, u, 2, u5)*3*u1**2 + df(f, x, u1, u5)*6*u2 + df(f, u, u1, u5)*6*u1*u2 + df(f,
u1, 2, u5)*3*u2**2 + df(f, x, u2, u5)*6*u3 + df(f, u, u2, u5)*6*u1*u3 + df(f, u1, u2, u5)*6*u2*u3 + df(f, u2, 2, u5)*3*u3**2
+ df(f, x, u3, u5)*6*u4 + df(f, u, u3, u5)*6*u1*u4 + df(f, u1, u3, u5)*6*u2*u4 + df(f, u2, u3, u5)*6*u3*u4 + df(f, u3, 2, u5)
*3*u4**2 + df(f, x, u4, u5)*6*u5 + df(f, u, u4, u5)*6*u1*u5 + df(f, u1, u4, u5)*6*u2*u5 + df(f, u2, u4, u5)*6*u3*u5 + df(f, u
3, u4, u5)*6*u4*u5 + df(f, u4, 2, u5)*3*u5**2 + df(f, x, u4)*3 + df(f, u, u4)*3*u1 + df(f, u1, u4)*3*u2 + df(f, u2, u4)*3*u3
+ df(f, u3, u4)*3*u4 + df(f, u4, 2)*3*u5 + df(f, u, u5)*3*u2 + df(f, u1, u5)*3*u3 + df(f, u2, u5)*3*u4 + df(f, u3, u5)*3*u5
```

Рисунок 3.4 — Строковое представление дифференциальных полиномов

```
In [19]: LRT = Janet()
         res = ginvBlockLow([diff2poly(eq) for eq in eqs], \
         LRT, level=1)
         print(f"     time: {res[0]:.2f} sec")
         print(f"    count: {LRT.count()}")
         print(f"reduction: {LRT.reduction()}")
         print(f"       HP: {LRT.HP()}")

     time: 8.21 sec
    count: 19
reduction: 2039
       HP: 5
```

Рисунок 3.5 — Построение базиса Жане

```
In [20]: for w in LRT.GB():
             print(w.poly)

df(f, u5, 2)
df(f, u4)
df(f, u3, u5)
df(f, u3, 2)
df(f, u2) + df(f, u5)*(-20*u1)
df(f, u1, u5)
df(f, u1, u3)
df(f, u1, 2)
df(f, u, u5)
df(f, u, u3) + df(f, u5)*(-10)
df(f, u, u1) + df(f, u3)*(-6)
df(f, u, 2) + df(f, u5)*(-60*u1)
df(f, x)*(-2) + df(f, u)*u1 + df(f, u3)*(-6*u1**2) + df(f, u5)*(-10*u1*u3)
df(f, t)*2 + df(f, u)*(6*u*u1 + 3*u3) + df(f, u1)*(-12*u1**2) + df(f, u3)*
(-36*u*u1**2 - 30*u1*u3) + df(f, u5)*(-60*u*u1*u3 - 12*u1*u5 - 30*u3**2) + df(f)*12*u1
```

Рисунок 3.6 — Построение базиса Гребнера

3.1.3 Работа алгоритма

Рассмотрим работу функции algorithm2, находящей базис в порядке $\prec$ *deglex*. В качестве аргументов она принимает список полиномов и значение флага output, обозначающее, нужно ли выводить логи процесс вычислений.

```
def algorithm2(self, F, output=False):
    assert (Monom.cmp == Monom.TOPdeglex
        or Monom.cmp == Monom.POTdeglex)
    super(GB, self).__init__()
    self.time, self.crit1, self.crit2 = time.time(), 0, 0
    self.F, self.B = F, []
```

В список F помещаются все полиномы из идеала. Объявляется список B. В будущем в него будут добавляться S-пары полиномов с указанием монома делителя. Запускается цикл до полного исчерпания списков F и B. Список B сортируется.

```
    self.F, self.B = F, []
    while F or self.B:
     self.B.sort()
```

Запускается подцикл для всех элементов B. Для всех пар полиномов и делящего монома из B выполняется проверка на прохождение критерия инволютивности. В случае, если деление полиномов будет инволютивным, выполняется нахождение S-полинома по формуле из определения 3. Если S-полином не нулевой, он упрощается, и добавляется в список F. В любом случае рассмотренный элемент B удаляется.

```
    while self.B:
        m, p1, p2 = self.B[0]
        if p1 and p2 and p1.lm().position() == p2.lm().position()
                and m == p1.lm().lcm(p2.lm())
                and not self.__crit2(m, p1, p2):
            p = Poly.S(p1, p2)
            if p:
                p.simplify()
                F.append(p)
        del self.B[0]
```

Далее в новом подцикле перебираются все элементы списка F. Для Каждого элемента применяется деление на подходящие полиномы из текущего базиса

Гребнера. Деление выполняется с головы, с помощью функции NFhead, по алгоритму NormalForm из работы Блинкова [12]. Если в процессе деления полином обращается в 0, удаляем его из списка F, так как он полностью редуцируется уже имеющимся базисом. В противном случае, упрощаем полином. Если еще не был назначен полином p или же старший терм i-го полинома меньше старшего терма из полинома p по выбранному порядку, присваиваем p значение i-того полинома. В случае, если не было выполнено удаление полинома, инкрементируем i для перехода к следующему полиному. p, i = None, 0

```
while i < len(F):
 F[i].NFhead(self)
 if not F[i]:
    del F[i]
 else:
    F[i].simplify()
    if not p or F[i].lm().cmp(p.lm()) < 0:
       p = F[i]
    i += 1
```

После этого, если был определен полином p, т.е. не все полиномы из списка F были редуцированы, выполняется дополнительный шаг. Если оставшийся нередуцированным полином не включает в себя ни одной переменной, т.е. состоит из константного значения, обнуляется составленный базис и единственным его членом записывается константный полином, а цикл прерывается досрочно. Если же полином p не является константой, удаляем его из списка F, применяем к нему функцию NFTail, аналогичную функции NFhead, но выполняющей деление с хвоста. Полученный результат упрощается и добавляется в базис. Функция добавления полинома в базис, в свою очередь, также пополняет B – список потенциальных S-пар полиномов.

```
if p:
 if p.lm().position() == 0 and p.lm().degree() == 0:
    del self[:]
    self.append(Poly()+1)
    break
 F.remove(p)
```

```
            p.NFtail(self)
            p.simplify()
            self.__insert(p)
```

По окончанию цикла удаляем промежуточные списки и вычисляем затраченное на построение базиса время.

```
  del self.B
  del self.F
  self.time = time.time() - self.time
```

Последовательность выполнения команд различными классами в процессе работы algorithm2 продемонстрирована на диаграмме последовательности действий 3.7.

Одна из основных функций, используемых в этом процессе – функция редукции i-того монома в полиноме при делении на другой полином.

```
def reduction(self, i, other):
```

В начале выполняется серия проверок базовых условий, с использованием команды assert. Эта системная команда генерирует ошибку и останавливает работу программы в случае ложного значения. Последовательно проверяется неравенство полиномов, участвующих в делении, наличие i-того монома, а также сама возможность выполнить деление полиномов.

```
    assert id(self) != id(other)
    assert 0 <= i < len(self)
    assert self[i][0].divisible(other.lm())
```

Далее вычисляется наибольший общий делитель для коэффициента при i-том мономе и старшего коэффициента делителя. Далее выполняется деление i-того монома на старший моном делителя. Сохраненное значение monom соответствует тому моному, на который следует домножить делитель для редукции i-того монома в делимом. Также вычисляются k1 и k2 – коэффициенты старшего монома делителя и i-того монома, разделенные на вычисленный НОД. При этом k2 берется с отраженным знаком. В случае, если при делении на НОД получился дробный формат значений, k1 и k2 сокращаются до целого.

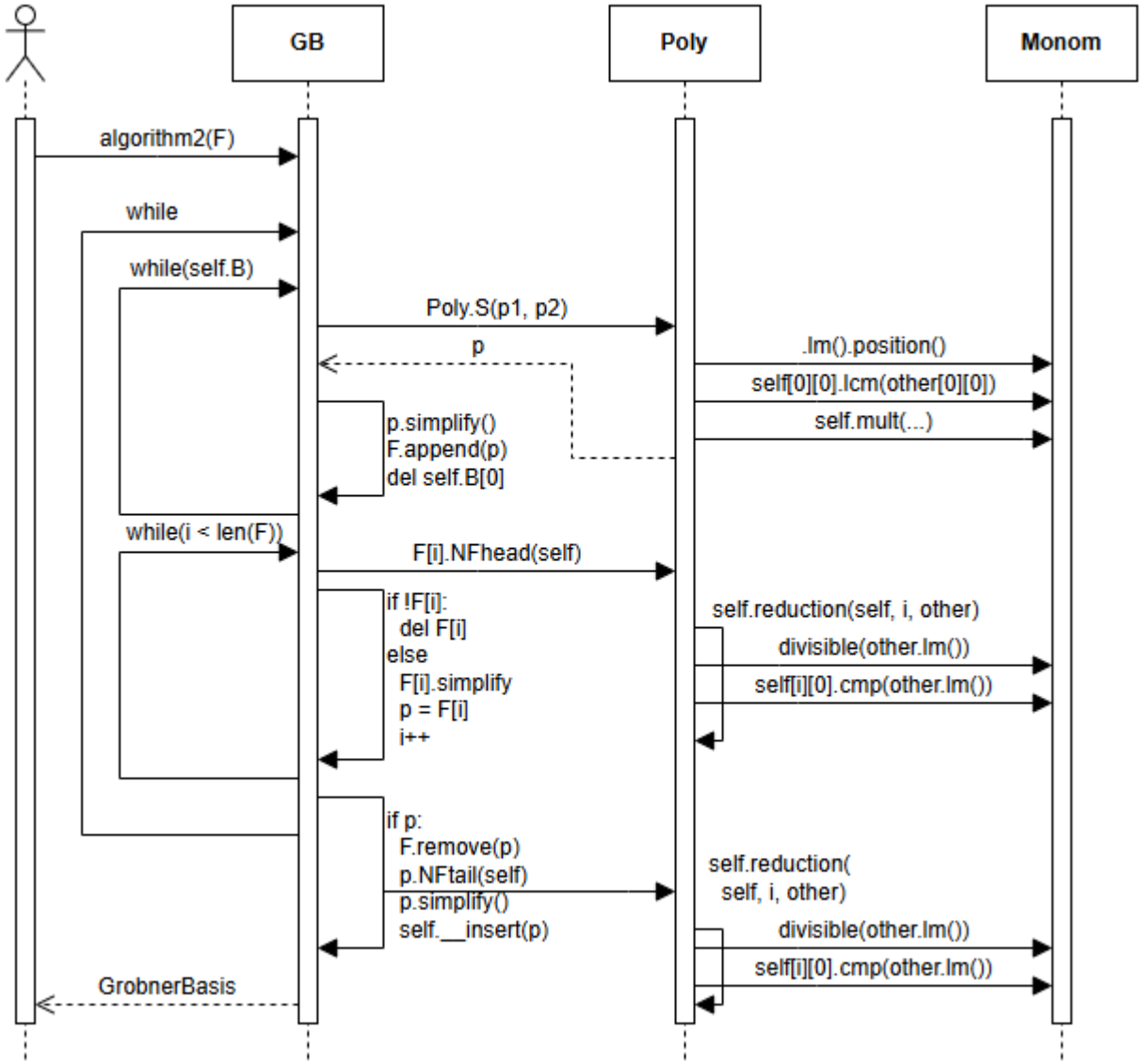

Рисунок 3.7 — Последовательность работы функции algorithm2

```
k = gcd(self[i][1], other.lc())
monom, k1, k2 = self[i][0]/other.lm(), other.lc()/k, -self[i][1]/k
if not k1.is_integer:
    k1 = cancel(k1)
if not k2.is_integer:
    k2 = cancel(k2)
```

Далее все коэффициенты мономов до i-того домножаются на k1, а i-тый моном удаляется – таким образом выполняется редукция i-того монома.

```
for mk in self[:i]:
  mk[1] *= k1
del self[i]
```

Теперь остается посчитать сумму оставшихся мономов. Начиная со второго монома делителя (j = 1), и i-того монома делимого (после удаления на i-тую позицию автоматически встает следующий), выполняем цикл до конца одного из полиномов.

```
j = 1
while i < len(self) and j < len(other):
```

В цикле выполняем сравнение i-того монома делимого и j-того монома делителя, домноженного на monom – ранее вычисленный результат деления изначального i-того монома делимого на старший моном делителя.

```
  c = self[i][0].cmp(other[j][0]*monom)
```

Если моном делимого стоит раньше в выбранном порядке, домножаем его на k1, переходим на следующий моном в делителе.

```
  if c > 0:
    self[i][1] *= k1
    i += 1
```

Если домноженный моном делителя стоит раньше в выбранном порядке, вставляем его на место i-того монома, с домноженным на k2 коэффициентом. Переходим на следующую позицию как в делимом, так и в делителе.

```
  elif c < 0:
    self.insert(i, [other[j][0]*monom, other[j][1]*k2])
    i += 1
    j += 1
```

В случае, если i-тый моном делимого и домноженный j-тый моном делителя совпадают, складываем их коэффициенты, соответственно домноженные на k1 и k2. Раскрываем скобки и, если итоговый коэффициент нулевой, удаляем i-тый моном. Если коэффициент ненулевой, записываем его как коэффициент i-того монома и переходим к следующему моному делимого. В любом случае, также переходим к следующему моному делителя.

```
    else:
      k = self[i][1]*k1 + other[j][1]*k2
      if not k.is_integer:
        k = expand(k)
      if k == S.Zero:
        del self[i]
      else:
        self[i][1] = k
        i += 1
      j += 1
```

Если после выполнения цикла остались необработанные мономы делимого, домножаем их коэффициенты на k1.

```
while i < len(self):
  self[i][1] *= k1
  i += 1
```

Если остались необработанные мономы делителя, домножаем их коэффициенты на k2, а термы на значение monom, и добавляем в полином.

```
while j < len(other):
  self.append([other[j][0]*monom, other[j][1]*k2])
  j += 1
```

В конце подтверждаем, что в итоге полином соответствует выбранному порядку.

```
assert self.assertValid()
```

Часто используемые элементарные функции, реализованы крайне эффективно благодаря внутреннему представлению класса Monom. Например, функция gcd() – поиск наибольшего общего делителя. Так как терм представляет собой простой набор целых чисел, отражающий степени переменных, входящих в состав терма, НОД для двух таких объектов находится как набор минимальных степеней по каждой из позиций. Функция divisible() – определение делимости одного терма на другой, также работает через элементарное

сравнение степеней термов по позициям переменных. Один терм делиться на другой тогда и только тогда, когда в каждой паре степеней делимого и делителя по одной позиции, степень делителя не превосходит степень делимого.

3.1.4 GInvDist

Актуальная версия системы GInv была доработана, дополнена комментариями, приведена в стабильное состояние, и выложена на отдельном репозитории [50], под именем GInvDist.

Далее, эта версия системы была сжата в пакет и опубликована в PyPI – открытом репозитории библиотек на языке Python. Для установки данной библиотеки достаточно выполнить команду pip install ginvdist. Это автоматически обновляет требуемые для работы системы GInv библиотеки – NumPy, SymPy и GraphVis. При использовании классических сред разработки, например PyCharm, необходимо выполнить данную команду в системном терминале. Для сред на основе интерактивных блокнотов, таких как Jupyter Notebook или Google Colab, ту же команду можно выполнить в блоке кода, с указанием «!» в начале:

```
!pip install ginvdist
```

Для дальнейшей работы нужно импортировать необходимые модули GInv. Основные модули соответствуют основным классам, рассмотренным в пункте 2.1 – monom, poly, gb. Далее можно применять функционал системы также, как если бы она была полноценно развернута в используемой среде разработки. Пример базовой работы системы может иметь следующий вид:

```
from ginv.monom import *
from ginv.poly import *
from ginv.gb import *

# задача используемых переменных (в статическое поле класса Monom)
variables = ['x1', 'x2', 'x3', 'x4', 'x5']
Monom.init(variables)
```

```
Monom.variables = variables.copy()
Monom.zero = Monom(0 for _ in Monom.variables)
Monom.cmp = Monom.TOPdeglex # указание мономиального порядка

for i in range(len(Monom.variables)):
    p = Poly()
    p.append([Monom(0 if l != i else 1
        for l in range(len(Monom.variables))), 1])
    globals()[Monom.variables[i]] = p

G = GB()
G.algorithm2([
  x1 + x2 + x3 + x4 + x5,
  x1*x2 + x1*x5 + x2*x3 + x3*x4 + x4*x5,
  x1*x2*x3 + x1*x2*x5 + x1*x4*x5 + x2*x3*x4 + x3*x4*x5,
  x1*x2*x3*x4 + x1*x2*x3*x5 + x1*x2*x4*x5 + x1*x3*x4*x5 + x2*x3*x4*x5,
  x1*x2*x3*x4*x5 - 1
], out=True)

# Вывод результатов
print(G)
print(", ".join(str(g.lm()) for g in G)) # Вывод старших мономов
print("crit1 =", G.crit1, "crit2 =", G.crit2)
print("time %.2f" % G.time) # Время вычислений
```

Также, для отслеживания процесса вычислений, была добавлена опциональная возможность сопровождать вычисления выводом промежуточных вычислений. Для этого требуется передать в функции вычисления значение флага out=True. При этом выводимые логи будут иметь следующий вид 3.8.

При этом, стоит отметить, что

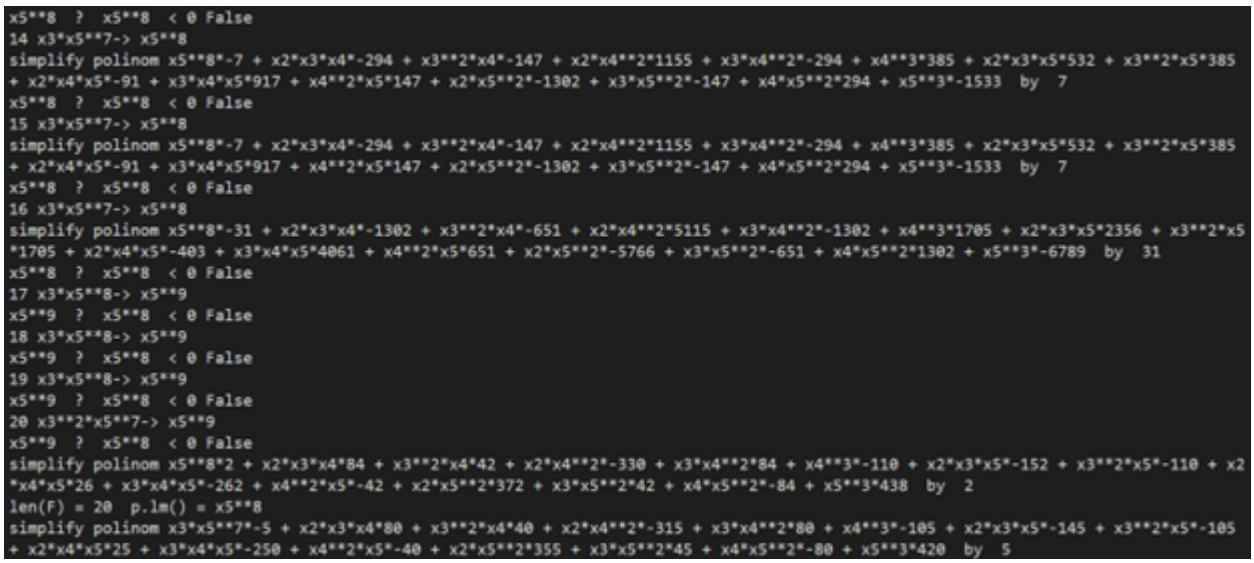

Рисунок 3.8 — Базовый тест системы

3.2 Тестирование системы GInv

3.2.1 Система тестирования

При разработке системы тестирования одним из первых этапов являлся переход от хранения данных в формате XML к использованию более современного формата обмена данными JSON. Для выполнения этого перехода был создан специальный программный скрипт. Использование формата JSON в качестве основы для хранения и обмена данными в нашей системе обусловлено рядом преимуществ. Формат основан на простых конструкциях ”ключ-значение”, что упрощает анализ и обработку данных, в том числе внесение изменений в файлы вручную. Также файлы JSON довольно компактны, что оптимизирует объемы при их хранении и передач. Кроме того, данный формат поддерживается на всех популярных платформах и во многих языках программирования, что обеспечивает кроссплатформенную совместимость и интеграцию с другими информационными системами, а также простую конвертацию, при возникновении такой потребности.

Для автоматизации тестирования системы решения уравнений GInv была создана специализированная утилита [52]. Эта утилита устраняет необходимость ручного ввода неизвестных, уравнений и других сопутствующих данных,

требуемых для решения системы уравнений. Процесс начинается с подачи на вход пути к файлу, содержащему тестовые данные.

Разработанная утилита состоит из трех ключевых функциональных блоков.

1. Модуль импорта данных в формате JSON. Осуществляет загрузку исходных данных, представленных в стандартизированном обменном формате JSON.
2. Модуль построения базиса Гребнера. Выполняет вычисление базиса Гребнера относительно заданного множества многочленов на основе алгоритмов компьютерной алгебры.
3. Модуль вывода результатов тестирования. Производит запись полученных в ходе работы программы результатов в указанное хранилище для последующего анализа и верификации.

Для корректной обработки набора уравнений, был реализован алгоритмический цикл, параметризированный массивом названий тестов.

Пределы этого цикла определяются двумя числами, соответствующими начальной и конечной позициям в этом массиве. Это позволяет гибко настраивать процесс вычислений в зависимости от конкретных потребностей при тестировании.

Сама процедура вычисления базисов Гребнера реализована следующим образом. На начальном этапе осуществляется инициализация пустого базиса. Далее производится пошаговое пополнение базиса многочленами в соответствии с выбранным алгоритмом построения базисов. На протяжении всего процесса вычислений ведется замер временных затрат.

Завершающей стадией является передача полученных в ходе эксперимента данных, а именно: вычисленного базиса Гребнера и значений затраченного времени, в функцию записи результатов тестирования для последующей обработки и анализа.

При решении каждой новой задачи происходит вывод на экран текущей совокупности математических уравнений, описывающих поставленную задачу, а также идентификатора конкретного тестового набора. Данный процесс обеспечивает возможность отслеживать смены состояний системы в процессе переходов между режимами обработки различных наборов входных данных и решения конкретных уравнений и позволяет повысить контролируемость и воспроизводимость вычислений.

После нахождения решения конкретной системы уравнений выводится время, затраченное системой на вычисление базисных векторов данной системы. Анализ этого параметра позволяет оценить производительность и выявить возможности для оптимизации алгоритмов решения.

3.2.2 Набор тестов

Основной набор тестов состоит из 135 тестовых файлов. В список тестов включены вариации известных задач, таких как: система Роуза, модель общего экономического равновесия; обобщенная задача о собственных значениях; система Лоренца, и другие. Эти тесты включают в себя задачи по алгебраическим системам из теории кодирования, химической кинетики и робототехники. Тесты хранятся в файлах формата JSON.

Внутри JSON файлов хранится следующая информация: обязательно – размерность системы уравнений, список многочленов, список переменных; опционально – текстовое описание задачи. Пример тестовых файлов приведен ниже 3.9 (без описания) и 3.10 (с описанием).

Все файлы собраны в единый архив для удобства хранения. Доступ к файлам в архиве осуществляется с помощью отдельного модуля, требуемого для чтения и вывода списка файлов ZIP.

3.2.3 Результаты тестирования системы GInvDist

Система тестировалась на серверной платформе, состоящей из двух 4-ядерных процессоров Intel Xeon L5630. Каждый процессор имел по 4 вычислительных ядра с поддержкой технологии Hyper-Threading, что позволяло запускать по 2 потока на каждом физическом ядре. Таким образом, общее количество логических ядер (потоков обработки) составило 8. Базовая тактовая частота каждого ядра процессора составила 2134 МГц.

По итогам выполнения каждого теста производится запись полученных результатов, также в формате JSON файлов. Важно, что в случае повторно-

```
{
    "variables": [
        "ca",
        "cb",
        "cc",
        "cd",
        "sa",
        "sb",
        "sc",
        "sd"
    ],
    "equations": [
        "ca^2+sa^2-1",
        "cb^2+sb^2-1",
        "cc^2+sc^2-1",
        "cd^2+sd^2-1",
        "4*(2*ca*cb*cc-2*ca*cb*sc-4*ca*cb-2*ca*cc*sb-2*ca*sb*sc-3*
        ca*sb+6*ca+2*cb*cc*sa+6*cb*cc+2*cb*sa*sc+3*cb*sa-6*cb*sc-
        12*cb+2*cc*sa*sb-6*cc*sb-cc-2*sa*sb*sc-4*sa*sb-6*sb*sc-9*sb+7*sc+12)",
        "-8*ca*cb*cd+16*ca*cb*sd+8*ca*cb-16*ca*cd*sb-8*ca*sb*sd-12
        *ca*sb-28*ca+16*cb*cd*sa-32*cb*cd+8*cb*sa*sd+12*cb*sa-86*cb
        *sd-73*cb-8*cd*sa*sb+86*cd*sb+16*cd+16*sa*sb*sd+8*sa*sb-30*sa-32*sb*sd+12*sb+28*sd+112",
        "2*(140*ca*cb*cc-490*ca*cb*cd+700*ca*cb*sc-980*ca*cb*sd+28*
        ca*cb+700*ca*cc*sb+980*ca*cd*sb-140*ca*sb*sc-490*ca*sb*sd-
        1134*ca*sb+392*ca-700*cb*cc*sa+3430*cb*cc-980*cb*cd*sa+3430
        *cb*cd+140*cb*sa*sc+490*cb*sa*sd+1134*cb*sa+770*cb*sc-4165*
        cb*sd-5047*cb+3150*cc*cd+140*cc*sa*sb+770*cc*sb-2450*cc*sd-
        4030*cc-490*cd*sa*sb+4165*cd*sb-2450*cd*sc-5740*cd+700*sa*
        sb*sc-980*sa*sb*sd+28*sa*sb-882*sa-3430*sb*sc+3430*sb*sd-
        2394*sb-3150*sc*sd+910*sc+2695*sd+9899)",
        "280*cc*cd-240*cc-336*cd-280*sc*sd+391"
    ],
    "description": null,
    "dimension": 8
}
```


Рисунок 3.9 — Пример теста – assur44.json

го выполнения теста, вместо создания новой записи обновляются предыдущие результаты. Запись результатов содержит: время вычисления, размерность системы, значение внутренних переменных алгоритма для crit1 и crit2, список старших термов базиса, сам найденный базис, а также объем использованной во время работы базиса оперативной памяти. Пример файла с результатами можно видеть ниже 3.11.

Исследование эффективности разработанной системы осуществлялось посредством специально подготовленных наборов входных данных. На первом этапе был проведен отбор наиболее представительных тестовых примеров, отличающихся по ключевому параметру - времени выполнения анализируемого уравнения. Полученные в ходе тестирования значения времени выполнения алгоритма для каждого теста представлены в сводной таблице результатов 2. Такая систематизация данных необходима для последующий удобной обработки и анализа.

```
{
    "variables": [
        "c1",
        "c2",
        "g1",
        "g2",
        "s1",
        "s2"
    ],
    "equations": [
        "g1^2+s1^2-1",
        "g2^2+s2^2-1",
        "5*c1*g1^3+5*c2*g2^3-6",
        "5*c1*s1^3+5*c2*s2^3-6",
        "10*c1*g1^2*s1+10*c2*g2^2*s2-7",
        "10*c1*g1*s1^2+10*c2*g2*s2^2-7"
    ],
    "description": "TITLE : neurofysiology, posted by Sjirk Boon
        NOTE :
                There are only 8 finite solutions for general values of
                the constant terms.
                It can be proved that it is equivalent to a quadrature formula
                problem, so that there is only one solution upon symmetry.
        REFERENCES :
                The system has been posted to the newsgroup
                sci.math.num-analysis by Sjirk Boon.
        P. Van Hentenryck, D. McAllester and D. Kapur:
        `Solving Polynomial Systems Using a Branch and Prune Approach'
        SIAM J. Numerical Analysis, Vol. 34, No. 2, pp 797-827, 1997.
        SYMMETRY GROUP :
                g2 s2 g1 s1 C2 C1
                g1 s1 g2 s2 C1 C2
                s2 g2 s1 g1 C2 C1
                s1 g1 s2 g2 C1 C2

                -s1 s2 -g1 g2 -C1 C2
                s1 -s2 g1 -g2 C1 -C2",
    "dimension": 6
}
```


Рисунок 3.10 — Пример теста – boon.json

Для демонстрации были выбраны тесты с временем выполнения, последовательно отличающимся на порядок. Очевидно, что большее время выполнения коррелирует с большим значением редукции, а также с длиной базиса. Наиболее явно это можно увидеть на графике зависимости времени выполнения от длины базиса 3.12.

Обе величины представлены на графике на логарифмических шкалах. За исключением выбросов с нестандартно долгим временем выполнения, получаемых на нетривиально сложных тестах, можно проследить общую линию

```
{
        "time": "0.19896578788757324",
        "dimension": 6,
        "crit1": "108",
        "crit2": "33",
        "leads": "c2, c1, g2*s2, s1**2, g2*s1, g1*s1, g2**2, g1*g2, g1**2, s2**3, s1*s2**2, g1*s2**2",
        "basis": "c2*-330 + g2*361 + s2*361,
                  c1*330 + g1*-361 + s1*-361,
                  g2*s2*-19 + 7,
                  s1**2*-1 + s2**2*-1 + 1,
                  g2*s1*7 + g1*s2*7 + s1*s2*-19,
                  g1*s1*-19 + 7,
                  g2**2 + s2**2 + -1,
                  g1*g2 + s1*s2*-1,
                  g1**2*-1 + s2**2,
                  s2**3*19 + g2*7 + s2*-19,
                  s1*s2**2*19 + g1*-7,
                  g1*s2**2*-19 + g1*19 + s1*-7",
        "avr memory": 84.59140625,
        "max memory": 84.59765625
}
```


Рисунок 3.11 — Результат теста boon.json

Таблица 2 — Выдержки тестовых результатов

Тест	Размерность	Длина базиса	Редукции (шт)	Время (мс)
ilias13	7	1	0	0,53
comb3000	10	35	503	11,70
hcyclic6	7	221	18634	308,41
eco9	9	189	159992	4322,05
hcyclic7	8	1182	542213	56018,99

зависимости между длительностью вычислений и длиной построенного базиса Гребнера. Также со временем вычислений коррелирует и объем использованной памяти, однако отсутствует корреляция с размерностью изначальной задачи. Это можно увидеть ниже 3.

В среднем на совокупности всех тестов система GInv дает вдвое меньшее время работы, нежели Sage. Однако стоит заметить, что количественно многие тесты, напротив, выполняются быстрее системой классическим алгоритмом, например, в системе Sage. Примером таких тестов может быть *chemkin*, тесты семейства *eco* и *cyclic*.

Это связано с эвристической природой инволютивного алгоритма построения базиса. Критерий инволютивного замыкания и накладывает фильтр на рассматриваемые S-пары полиномов, что сокращает рассматриваемое количество комбинаций. Однако, так как алгоритм итеративный, это ограничение

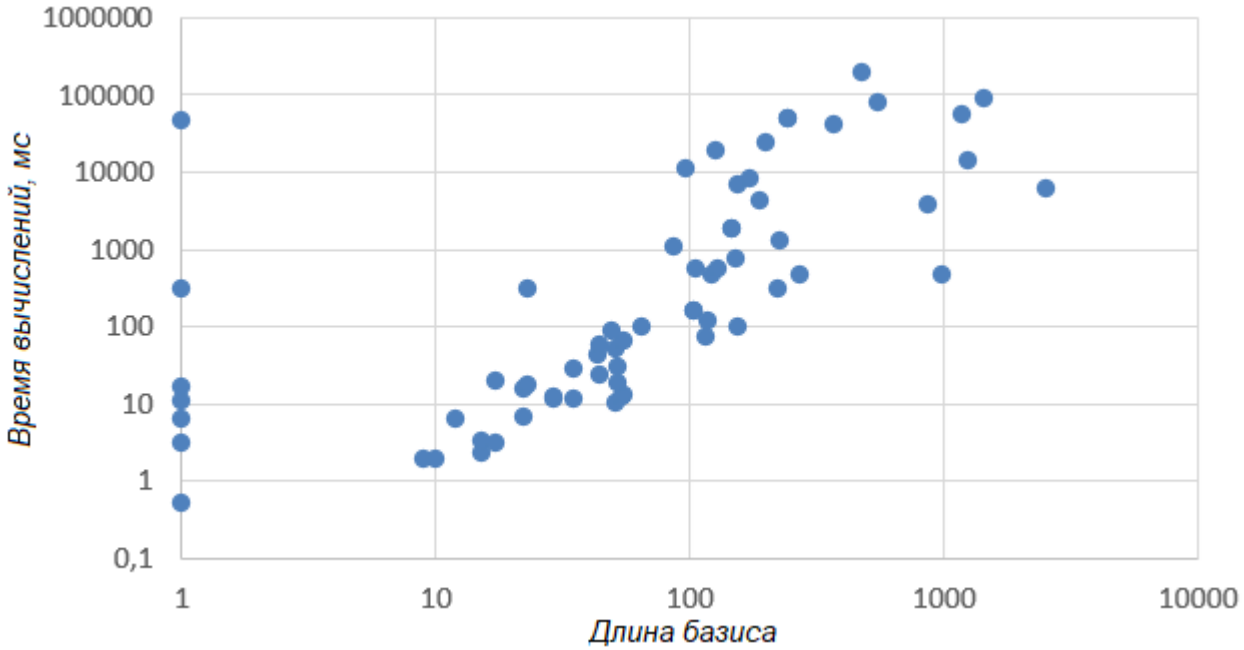

Рисунок 3.12 — Зависимость времени вычисления от длины базиса

изменяет набор рассматриваемых полиномов на последующих шагах, что в определенных задачах ведет к построению неоптимального набора полиномов и более долгой работе алгоритма.

Для наглядной демонстрации на рисунке ниже 3.13 представлены время вычисления одних и тех же тестов инволютивным (в системе GInv) и классическим алгоритмом (с использованием реализации из библиотеки sympy).

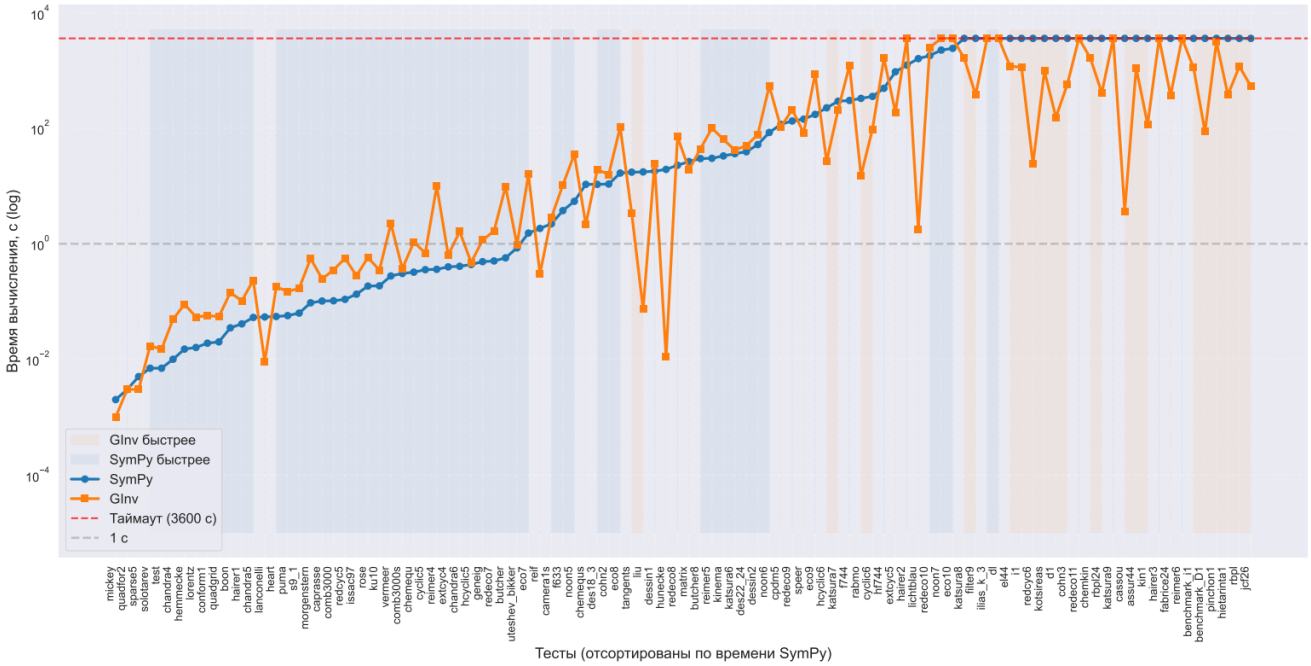

Рисунок 3.13 — Сравнение времени вычисления тестов классическим и инволютивным алгоритмом.

Таблица 3 — Выдержки тестирования с замером используемой памяти

Название	Время вычислений, с	Оперативная память, MB	Редукции (шт)	Размерность
boon	0.19	84.59	141	6
butcher	7.37	95.22	552	7
butcher8	48.66	99.31	1883	8
camera1s	3.80	88.45	267	6
caprasse	0.29	84.62	258	4
cassou	4.91	96.57	200	5
chandra4	0.06	83.70	28	4
chandra5	0.29	84.60	111	5
chandra6	1.66	86.23	459	6
chemequ	1.37	85.37	139	5
chemequs	2.55	94.73	167	5
cohn2	8.19	92.06	457	4
cpdm5	181.01	106.6	2893	5
cyclic5	0.77	84.90	447	5
cyclic6	57.75	94.84	4739	6

В данной выборке применялось ограничение на время вычислений в 1 час. Результаты упорядочены по возрастанию времени вычисления sympy. Это позволяет увидеть интересную закономерность – "быстрая" половина тестов показывает лучшие результаты с использованием классического алгоритма, в то время как "медленная половина" – с инволютивным.

Полные результаты доступны на репозитории, вместе с программой для тестирования [53].

3.3 Полученные результаты

В результате тестирования адаптированной системы GInv мы получаем положительные, но неоднозначные результаты. После обработки набора тестовых файлов и сравнения с результатами вычислений классического алгоритма, можно отметить разнородные результаты для разных семейств тестов. Это поз-

воляет поставить вопрос о классификации задач о нахождении базиса Гребнера, а также вопрос об определении оптимального алгоритма решения таковых задач для разных классов.

Эффективность инволютивного алгоритма, фактически представляющего собой использование эвристики для предположительно более оптимального подбора S-пар полиномов, также ставит вопрос о возможности использования других эвристик, представляющих иного рода ограничения на процесс перебора возможных полиномов. Применение инволютивного алгоритма для задачи моделирования движения нелинейного осциллятора, особенно в задачах биологических осцилляторов, также дает положительный результат, показывая более быстрое нахождение решения, по сравнению с классическим алгоритмом, что показано в соответствующих разделах первой главы данной диссертации (1.2, 1.3), хотя и требует дальнейшего развития.

Новая версия GInv, реализованная в виде библиотеки на языке Python, также поддерживает работу с системами дифференциальных уравнений, и не требует изменений в процессе применения как к этой, так и к другим задачам. Пример такой интеграции приведен в Приложении.

Помимо этого, реализация системы GInv в виде легко интегрируемой библиотеки облегчает развертывание системы распределенных вычислений для решения более трудоемких задач. Это открывает потенциальное направление в дальнейшей разработке системы в виде реализации алгоритмов распределенных и/или параллельных вычислений. Предоставление легкого инструментария для развертывания грид-системы для нахождения базиса Гребнера значительно расширит возможности исследователей в области математического моделирования.

Глава 4. Вычислительные эксперименты

4.1 Постановка эксперимента

4.1.1 Порядок проведения вычислительных экспериментов

Для проведения численных экспериментов использовалось разное вычислительное оборудование. Разнородность вычислительного оборудования позволяет проверить ряд гипотез о вычислительной сложности решаемых задач, а также зависимости скорости вычисления от различных характеристик используемого оборудования. Об этих гипотезах будет расписано более подробно далее.

Основными платформами для вычислений были выбраны сервер, предоставленный кафедрой, и персональный компьютер.

Под **Сервером** далее будет пониматься уже описанная ранее серверная платформа 3.2.

1. Два процессора Intel Xeon L5630, суммарно 8 ядер и 16 логических процессоров. Базовая тактовая частота каждого ядра – 2134 МГц.
2. Оперативная память: для вычисления на серверной платформе было выделено 100 гигабайт оперативной памяти.

Персональный компьютер, далее упоминаемый как **Личный ПК**, обладает следующими характеристиками:

1. Процессор: Intel Core i3-10105F, с частотой 3.70 Ггц, с 4 ядрами и 8 логическими процессорами соответственно.
2. Оперативная память: 32 Гб, со скоростью обращения 2667 МГц. Во время проведения экспериментов, было гарантированно доступно не менее 20 Гб.

Также, в рамках небольших дополнительных экспериментов, использовались различные персональные компьютеры и облачные платформы вычислений, характеристики которых будут упомянуты в связанных параграфах.

За основу для проведения эксперимента взят уже описанный ранее набор из 135 тестов в формате JSON. В него включены как уникальные тесты, так

и семейства тестов – наборы тестов со схожей формой построения образующих полиномов, и отличием только в размерности.

Для каждого из проведенных вычислительных экспериметов было определено следующее:

1. Платформа для вычислений: Сервер; Личный ПК; облачная платформа Sage Cell Server; вспомогательные персональные компьютеры.
2. Программное обеспечение для вычислений: Sage; GInv.
3. Возможность отслеживать объем используемой во время вычислений оперативной памяти.
4. Используемый мономиальный порядок: $\prec lex$; $\prec deglex$; $\prec degrevlex$.
5. Подмножество тестов из общего набора, тесты в котором были успешно выполнены на выбранной конфигурации.

Последний пункт определялся по факту проведения эксперимента.

4.1.2 Цели вычислительных экспериментов

Рассмотрим в виде отдельной вычислительной задачи нахождение базиса Гребнера или же другого инволютивного базиса, например – базиса Жане, для конкретного теста из набора тестового набора. В результате эксперимента ставится цель эмпирически подтвердить или опровергнуть следующие гипотезы:

1. Для различных классов задач более эффективно использовать различные мономиальные порядки. Таковые классы задач в дальнейшем будем называть *Классы по мономиальному порядку*. Следовательно, для отдельно взятой задачи можно подобрать наиболее эффективный мономиальный порядок.
2. Для различных классов задач более эффективно использовать различные виды мономиального деления. Как следствие, вычисления с использованием различных алгоритмов будут выполняться с разной скоростью. Таковые классы задач в дальнейшем будем называть *Классы по мономиальному делению*. Следовательно, для отдельно взятой задачи можно подобрать наиболее эффективное мономиальное деление.
3. Рассмотрим варианты *перестановок переменных* для задачи – такие замены имен переменных, которые изменяют их алфавитный,

а следовательно и лексикографический, порядок. Существуют такие несимметричные перестановки переменных, которые оказывают значимое влияние на скорость решения задачи.

Для первой и второй гипотезы следует пояснить, что предполагается не наличие одного наилучшего варианта мономиального порядка или мономиального деления, показывающего наилучшее время при применении ко всем тестам, а напротив – ситуация, в которой для некоторых задач предпочтительно использовать один вариант, а для других – другой.

Также следует указать, что правдивость указанных гипотез является, можно сказать, общеизвестной. Тем не менее, важно отметить, что нет отдельных оценок эффективности того или иного выбора мономиального порядка или деления в контексте разных классов задач, как нет и предлагаемых способов классификации задач для определения наиболее эффективного алгоритма решения.

Также нет обоснований полагать, что для широко известных тестов наиболее эффективен именно канонически представляемый набор переменных, а не какая-то иная их перестановка.

Проверка указанных гипотез также подразумевает и попытку выделить таковые оценки и способы классификации.

4.2 Результаты вычислительных экспериментов

Общее время успешно проведенных вычислений среди всех платформ – 270134, секунд, что соответствует 75 часов непрерывных вычислений.

При этом значительное количество времени было потрачено на вычисления, не увенчавшиеся успехом. Так, например, рассмотрим тест reimer8:

$$
\begin{aligned}
&\text{"}variables\text{"} : [x1, x2, x3, x4, x5, x6, x7, x8],\\
&\text{"}equations\text{"} :\\
&\quad 2*x1^2 - 2*x2^2 + 2*x3^2 - 2*x4^2 + 2*x5^2 - 2*x6^2 + 2*x7^2 - 2*x8^2 - 1,\\
&\quad 2*x1^3 - 2*x2^3 + 2*x3^3 - 2*x4^3 + 2*x5^3 - 2*x6^3 + 2*x7^3 - 2*x8^3 - 1,\\
&\quad 2*x1^4 - 2*x2^4 + 2*x3^4 - 2*x4^4 + 2*x5^4 - 2*x6^4 + 2*x7^4 - 2*x8^4 - 1,\\
&\quad 2*x1^5 - 2*x2^5 + 2*x3^5 - 2*x4^5 + 2*x5^5 - 2*x6^5 + 2*x7^5 - 2*x8^5 - 1,\\
&\quad 2*x1^6 - 2*x2^6 + 2*x3^6 - 2*x4^6 + 2*x5^6 - 2*x6^6 + 2*x7^6 - 2*x8^6 - 1,\\
&\quad 2*x1^7 - 2*x2^7 + 2*x3^7 - 2*x4^7 + 2*x5^7 - 2*x6^7 + 2*x7^7 - 2*x8^7 - 1,\\
&\quad 2*x1^8 - 2*x2^8 + 2*x3^8 - 2*x4^8 + 2*x5^8 - 2*x6^8 + 2*x7^8 - 2*x8^8 - 1,\\
&\quad 2*x1^9 - 2*x2^9 + 2*x3^9 - 2*x4^9 + 2*x5^9 - 2*x6^9 + 2*x7^9 - 2*x8^9 - 1
\end{aligned}
$$

Провести его вычисление на **Личном ПК** оказалось невозможным, т.к. вычисления в течении более 16 часов не дали никакого результата. Так как из имеющегося оборудование, вычисления длительностью более суток возможно было проводить только на **Сервере**, была предпринята попытка завершить тест на нем. На 42 день расчетов было принято решение прервать работу над тестом. К этому моменту вычисления заняли в памяти чуть более 80 из 100 доступых Гб.

После этого, с целью экономия времени, было принято прерывать вычисления, длительностью более двух часов. Несколько тестов, результаты тестирования которых будут представлены далее, составили группу исключений, т.к. были посчитаны в ночной период.

По итогам всех вычислений, как минимум одно успешное решение получили 100 тестов из 135 имеющихся в наборе. При этом, наибольшее количество решений удалось получить используя систему GInv и порядок $\prec degRevLex$. На **Сервере** общее время этих вычислений составило 133023 секунд, т.е. почти 37 часов. На **Личном ПК** удалось выполнить те же тесты, что заняло 78828 секунд, т.е. почти 22 часа.

Сравнивая полученные результаты, мы можем убедиться в их устойчивости – с небольшой погрешностью все вычисления проводятся на **Личном ПК** в 1.6 раз быстрее, чем на **Сервере**. Как видно на логарифмическом графике 4.1 зависимости времени вычислений (в секундах) от порядкового номера теста, линии графика практически совпадают. Корреляция Пирсона полученных значений – 0,95.

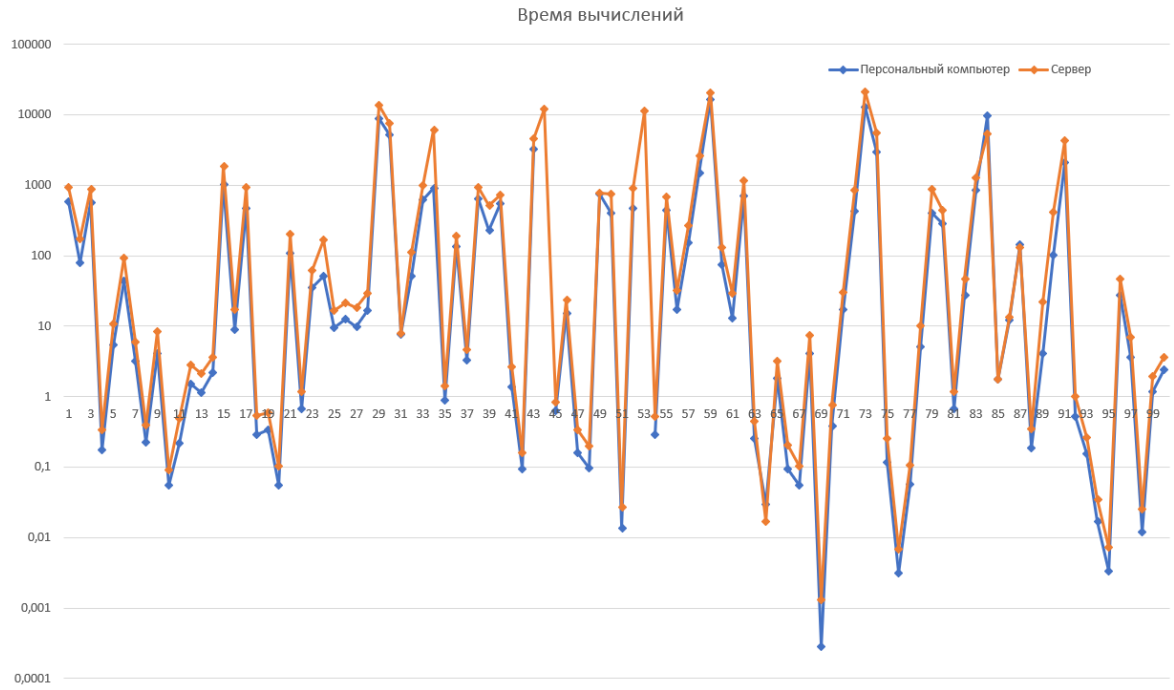

Рисунок 4.1 — Сравнение времени вычисления на разном оборудовании

Исключением из общего тренда являются тесты redeco11, hairer3 и ilias_k_3. Рассмотрим redeco11:

$$
\begin{aligned}
&"variables" : [x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, u11],\\
&"equations" :\\
&\quad -u11 + x1 * x2 + x1 + x10 * x9 + x2 * x3 + x3 * x4 + x4 * x5 + x5 * x6\\
&\qquad +x6 * x7 + x7 * x8 + x8 * x9,\\
&\quad -2 * u11 + x1 * x3 + x10 * x8 + x2 * x4 + x2 + x3 * x5 + x4 * x6 + x5 * x7\\
&\qquad +x6 * x8 + x7 * x9,\\
&\quad -3 * u11 + x1 * x4 + x10 * x7 + x2 * x5 + x3 * x6 + x3 + x4 * x7 + x5 * x8\\
&\qquad +x6 * x9,\\
&\quad -4 * u11 + x1 * x5 + x10 * x6 + x2 * x6 + x3 * x7 + x4 * x8 + x4 + x5 * x9,\\
&\quad -5 * u11 + x1 * x6 + x10 * x5 + x2 * x7 + x3 * x8 + x4 * x9 + x5,\\
&\quad -6 * u11 + x1 * x7 + x10 * x4 + x2 * x8 + x3 * x9 + x6,\\
&\quad -7 * u11 + x1 * x8 + x10 * x3 + x2 * x9 + x7,\\
&\quad -8 * u11 + x1 * x9 + x10 * x2 + x8,\\
&\quad -9 * u11 + x1 * x10 + x9,\\
&\quad -10 * u11 + x10,\\
&\quad x1 + x10 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + 1
\end{aligned}
$$

Это представитель тестов семейства redeco, соответствующих n-размерной экономической проблеме [38], с $n = 11$ в данном случае. Для записи полиномов в подходящем для обработки виде, относительно оригинальных уравнений была выполнена замена $u_n = \frac{1}{x_n}$, что позволяет избавиться от дробей.

В итоге вычисления для этого теста на **Личном ПК** заняли 9880 секунд, а на **Сервере** – 5398 секунд, т.е. в этот раз именно **Сервер** оказался быстрее.

Похожая ситуация с тестами hairer3 и ilias_k_3. Оба эти теста были посчитанны на **Сервере** чуть быстрее 3.5 часов. Следуя общей закономерности, они должны были посчитаться на **Личном ПК** примерно за 2 часа. Однако, отдельно для этих тестов было выделено дополнительно время вычислений, но даже за 3.5 часа ни один из них не посчитался. Таким образом, итоговое время вычислений на **Личном ПК** для них определить невозможно, но оно гарантированно больше, чем на **Сервере**.

Причина, почему для этих тестов, и именно для них, **Сервер** производит вычисления быстрее **Личного ПК**, требуют дальнейшего исследования. Сделать уверенный вывод без достаточного количества примеров других тестов с похожей зависимостью, нельзя. Есть ряд предположений – например, возможно для конкретно этих задач более важна скорость обращения к памяти, которая значительно выше на **Сервере**. Предположение о недостаточном объеме доступной оперативной памяти, вызывающем частую сборку мусора, не соответствует замеренному объему использованной оперативной памяти – для этих тестов он не многим выше среднего показателя в 150 Мб. Тем не менее, этот вопрос остается открытым для будущих исследований.

4.2.1 Гипотеза 1 : мономиальные порядки

Для проверки первой гипотезы сравним следующие результаты вычислительных экспериментов: время вычислений тестов с использованием различных мономиальных порядков. Для того, чтобы другие внешние параметры не искажали результат вычислений, будем проводить их на одинаковой платформе – **Сервере** и в одинаковой системе – Sage.

В данной системе используемый порядок можно задать при объявлении кольца полиномов, например, следующей командой:

$$R = PolynomialRing(ZZ, names = variables, order =' lex')$$

Параметр ZZ отображает создание кольца над полем целых чисел – т.е., что будут использоваться только целые коэффициенты. В параметр names передаются используемые переменные, а в order – порядок. Он может принимать такие значения, как *'lex' 'deglex' 'degrevlex'*, что соответствует ранее описанным мономиальным порядкам.

Нахождение базиса Гребнера выполняется с помощью реализованного в этой системе алгоритма Бухбергера, с оптимизационными улучшениями.

$$I = R.ideal(equations) \quad equations - ,$$

$$G = buchberger\_improved(I)$$

К сожалению, большинство тестов, дающих решение в данной конфигурации, решается за крайне малое время. Из-за этого, случайные колебания времени вычислений могут искажать результаты. Во избежание такого искажения, будем в первую очередь рассматривать результаты экспериментов, со временем вычислений более секунды, в идеале – более 10 секунд.

Также стоит отметить, что система Sage изредка демонстрирует неустойчивое поведение – так, некоторые тесты обычно показывают крайне малое время выполнения (менее секунды), но иногда – затягивают исполнение вплоть до часов, без изменения вызываемых команд. Это обнаружилось в ходе выполнения валидационных тестов. К сожалению, так как система Sage разрабатывается в виде открытого программного обеспечения, и включает в себя множество других программных модулей, отследить причину этого непостоянства пока не удается.

Результат проведенных вычислений можно видеть на следующем графике 4.2.

Корреляция Пирсона между временем вычисления в разных порядках составляет всего 0,02, т.е. лежит в рамках статистической погрешности. В подавляющем большинстве случаев, использование $\prec degrevlex$ уступает по скорости $\prec lex$.

Практически все исключения лежат в масштабе меньше секунды, и могут быть связаны со случайными колебаниями скорости вычислений. Несмотря на это, из этих данных можно вынести несколько интересных моментов.

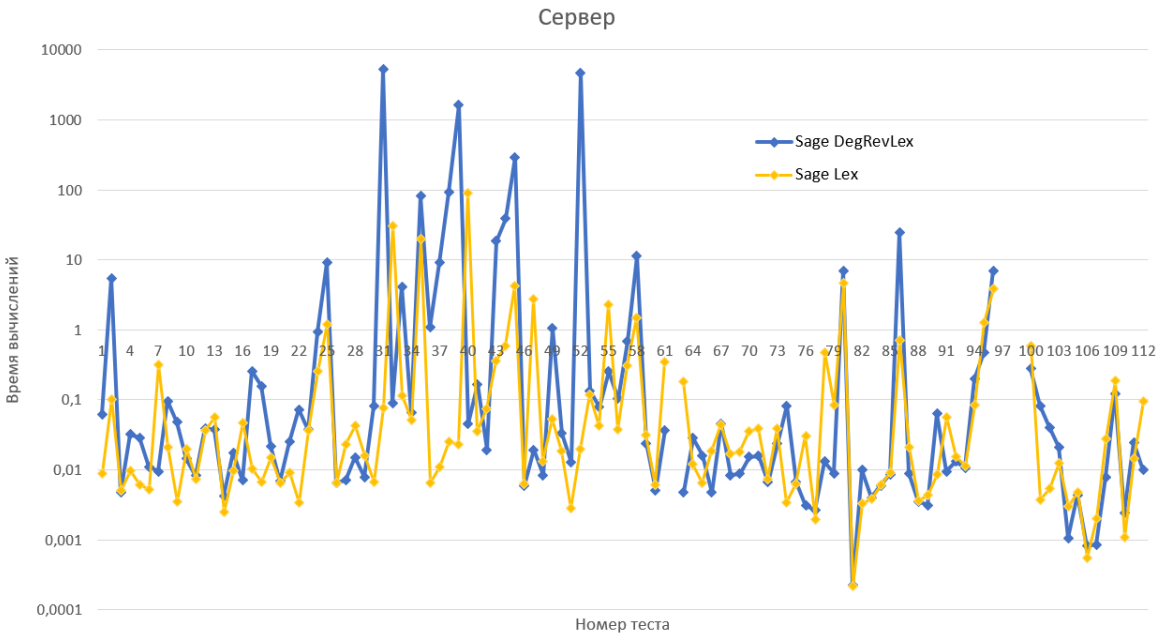

Рисунок 4.2 — Сравнение времени вычисления для разных мономиальных порядков

Во-первых, для большинства семейств тестов, при порядке $\prec lex$ не наблюдается значительного роста времени вычислений при росте размерности. При этом, порядок $\prec degrevlex$ дает значимый рост времени в одних семействах, при этом ведя себя как $\prec lex$ в других. Естественно было бы предположить, что рост времени вычисления происходит в тех семействах, в которых увеличение размерности сопровождается увеличением степеней мономов в образующих идеала, например – семейство extcyc.

Однако, семейство reimer, в котором также с размерностью растет и степень мономов, не демонстрирует такой закономерности при использовании алгоритма Бухбергера. При этом, при использовании инволютивного алгоритма, зависимость между размерностью и временем вычислений внутри таких семейств будет еще более заметна, о чем будет подробнее рассказано далее.

Во-вторых, при использовании $\prec degrevlex$ семейство тестов hairer демонстрирует странный выброс. Рассматриваются 4 теста, довольно специфического вида. Например, hairer2:

$$
\begin{aligned}
&\text{"}variables\text{"} : [a, b, c, d, e, f, g, h, i, j, k, l, m],\\
&\text{"}equations\text{"} :\\
&\quad d + e + f + g - 1,\\
&\quad 2 * a * f + 2 * b * e + 2 * c * d - 1,\\
&\quad 3 * a^2 * f + 3 * b^2 * e + 3 * c^2 * d - 1,\\
&\quad 6 * a * d * l + 6 * a * e * i + 6 * b * d * m - 1,\\
&\quad 4 * a^3 * f + 4 * b^3 * e + 4 * c^3 * d - 1,\\
&\quad 8 * a * b * e * i + 8 * a * c * d * l + 8 * b * c * d * m - 1,\\
&\quad 2 * a^2 * d * l + 2 * a^2 * e * i + 2 * b^2 * d * m - 1,\\
&\quad 24 * a * d * i * m - 1,\\
&\quad a - h,\\
&\quad b - j,\\
&\quad c - k - l - m
\end{aligned}
$$

При вычислении тестов этого семейства не только отсутствует значимый рост времени вычислений вместе с ростом размерности, но и наоборот – hairer2 вычисляется на несколько порядков дольше чем тесты как меньшей, так и большей размерности. Это можно увидеть на следующей таблице 4. При этом вычисления в порядке $\prec lex$ не демонстрируют подобного поведения.

Таблица 4 — Результаты вычисления для тестов hairer, порядок $\prec degrevlex$, система Sage

Имя	Размерность	Время ($\prec degrevlex$)	Время ($\prec lex$)
hairer1	8	0,01	0,002
hairer2	13	4615,30	0,019
hairer3	14	0,13	0,118
hairer4	20	0,08	0,042

Судя по дополнительно проведенным тестам, проблема в том, что размерность hairer2 = 13, что является простым числом. Это, вкупе с специфической формой полиномов, мешает алгоритму $buchberger_improved$ системы Sage определить эффективный порядок проведения редукций S-пар полиномов при использовании порядка $\prec degrevlex$.

Замечание. Это может подтверждать популярное предположение, что простая размерность тестов может негативно сказываться на эффективности их вычисления.

Также стоит упомянуть тесты el44 и ducos7_3. При порядке $\prec degrevlex$ они выполняются практически мгновенно – 0,04 и 0,09 секунды соответственно. При порядке $\prec lex$, несмотря на общий тренд, они занимают значимо большее время – 90,13 и 30,57 секунд. К сожалению, всего два противоположных примера не дают возможности сделать убедительные предположения о причинах такого поведения. Особенно интересен случай с ducos7_3, т.к. практически аналогичный ducos7_5 подчиняется общему тренду и вычисляется гораздо быстрее при порядке $\prec lex$.

Это дает пространство для дальнейшего изучения.

4.2.2 Гипотеза 2 : мономиальное деление

Для подтверждения или опровержения второй гипотезы, проведем аналогичную серию вычислительных экспериментов, но теперь сравним вычисления в разных системах: Sage, находящей классический базис Гребнера; и GInv, строящей инволютивный базис – в данном случае, базис Жане. Вычисления также проводятся на одинаковой платформе – **Сервере**. Для большей наглядности, было решено выполнить вычисления в порядке $\prec degrevlex$.

Несмотря на то, что разница реализации самих вычислений в разных системах неизбежно повлечет сильное отличие во времени вычисления, не обязательно коррелирующее с выбором мономиального деления, нас в первую очередь интересует корреляция между графиками. Она составила 0,39, что уже демонстрирует некоторую схожесть в распределении вычислительной сложности.

Общий тренд вычислений показывает большую эффективность системы Sage в сравнении с системой GInv. Так, целый ряд тестов, завершенных в системе Sage, так и не был завершен в системе GInv в рамках выделяемого на вычисления времени. Это тесты aubry2, cyclic7, ducos7_3, ducos7_5, ducos8, ducos10, extcyc6, extcyc7, extcyc8, hairer4, hawes4, hcyclic7.

Однако, что еще более интересно, для следующих тестов – hietarinta1, redeco9, redeco10, redeco11; наблюдается обратная картина – они были выполнены в системе GInv, но не смогли выполниться в Sage.

Так как конечность алгоритма Бухбергера, как с использованием классического, так и инволютивного деления, уже доказана, остается заключить, что данные тесты будут решены в обеих конфигурациях, но при этом в одной из них будут требовать кратно большего количества времени на вычисление.

Замечание. Большинство тестов, решаемых классическим алгоритмом за разумное время, решаются им на порядок быстрее, чем инволютивным. В то же время, существуют тесты, решаемые за разумное время только с использованием инволютивного деления. Из полученных результатов заметно, что рост вычислительной сложности для инволютивного алгоритма имеет более градиентный вид, в то время как для классического алгоритма происходит крайне резкий всплеск вычислительной сложности.

Все полученные результаты отображены на следующем графике 4.3:

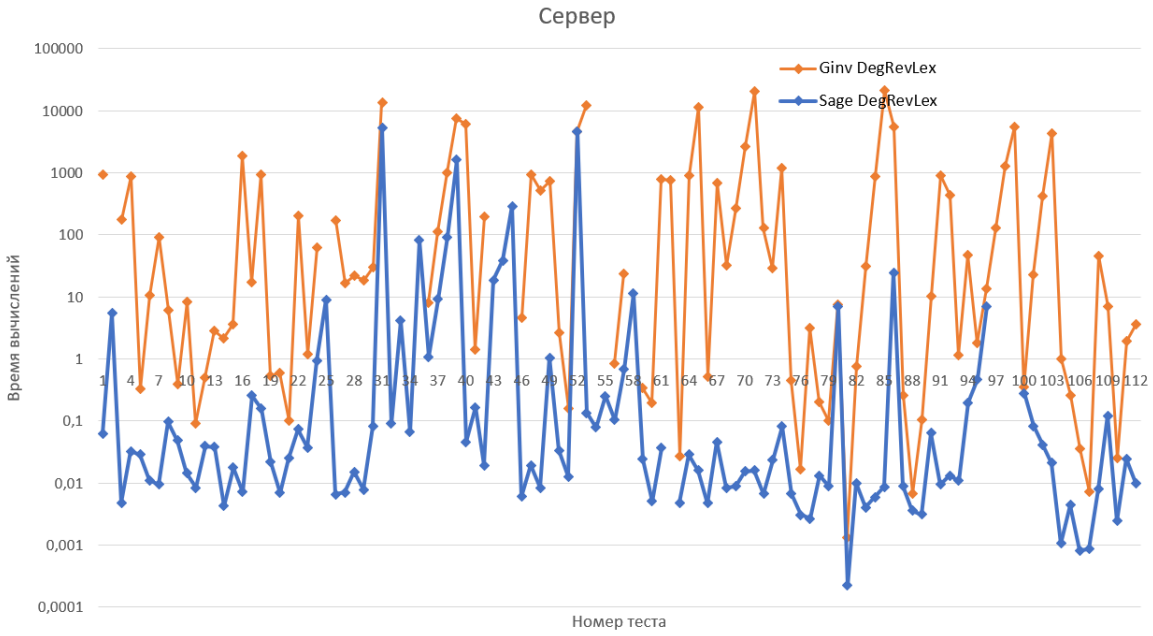

Рисунок 4.3 — Зависимость времени вычисления от длины базиса

Особенно интересно сравнить семейства extcyc и redeco. Пример теста из семейства redeco уже был представлен ранее 4.2. Для сравнения рассмотрим тест extcyc8.

$$
\begin{aligned}
&\text{"}variables\text{"} : [w, x1, x2, x3, x4, x5, x6, x7, x8],\\
&\text{"}equations\text{"} :\\
&\quad w^8 + 1,\\
&\quad w + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8,\\
&\quad w^2 + x1 * x2 + x1 * x8 + x2 * x3 + x3 * x4 +\\
&\qquad x4 * x5 + x5 * x6 + x6 * x7 + x7 * x8,\\
&\quad w^3 + x1 * x2 * x3 + x1 * x2 * x8 + x1 * x7 * x8 + x2 * x3 * x4 +\\
&\qquad x3 * x4 * x5 + x4 * x5 * x6 + x5 * x6 * x7 + x6 * x7 * x8,\\
&\quad w^4 + x1 * x2 * x3 * x4 + x1 * x2 * x3 * x8 + x1 * x2 * x7 * x8 +\\
&\qquad x1 * x6 * x7 * x8 + x2 * x3 * x4 * x5 + x3 * x4 * x5 * x6 +\\
&\qquad x4 * x5 * x6 * x7 + x5 * x6 * x7 * x8,\\
&\quad w^5 + x1 * x2 * x3 * x4 * x5 + x1 * x2 * x3 * x4 * x8 +\\
&\qquad x1 * x2 * x3 * x7 * x8 + x1 * x2 * x6 * x7 * x8 +\\
&\qquad x1 * x5 * x6 * x7 * x8 + x2 * x3 * x4 * x5 * x6 +\\
&\qquad x3 * x4 * x5 * x6 * x7 + x4 * x5 * x6 * x7 * x8,\\
&\quad w^6 + x1 * x2 * x3 * x4 * x5 * x6 + x1 * x2 * x3 * x4 * x5 * x8 +\\
&\qquad x1 * x2 * x3 * x4 * x7 * x8 + x1 * x2 * x3 * x6 * x7 * x8 +\\
&\qquad x1 * x2 * x5 * x6 * x7 * x8 + x1 * x4 * x5 * x6 * x7 * x8 +\\
&\qquad x2 * x3 * x4 * x5 * x6 * x7 + x3 * x4 * x5 * x6 * x7 * x8,\\
&\quad w^7 + x1 * x2 * x3 * x4 * x5 * x6 * x7 + x1 * x2 * x3 * x4 * x5 * x6 * x8 +\\
&\qquad x1 * x2 * x3 * x4 * x5 * x7 * x8 + x1 * x2 * x3 * x4 * x6 * x7 * x8 +\\
&\qquad x1 * x2 * x3 * x5 * x6 * x7 * x8 + x1 * x2 * x4 * x5 * x6 * x7 * x8 +\\
&\qquad x1 * x3 * x4 * x5 * x6 * x7 * x8 + x2 * x3 * x4 * x5 * x6 * x7 * x8,\\
&\quad x1 * x2 * x3 * x4 * x5 * x6 * x7 * x8 - 1
\end{aligned}
$$

Это семейство тестов происходит от задачи о N- циклических корнях [39]. Несмотря на схожий вид и размерности, между двумя семействами тестов наблюдается ключевое различие – мономы имеют б**о**льшую плотность мономов – тогда как в тестах redeco мономы состоят из произведения максимум двух переменных, в приведенном примере мономы включают в себя до восьми переменных.

4.2.3 Гипотеза 3 : перестановки переменных

Для проверки третьей гипотезы была написана специальная утилита, автоматически переименовывающая переменные при считывании тестового файла. Так как полное количество всех перестановок равно факториалу от количества переменных, $|Permutations| = dim!$, потребовалось ограничить используемые перестановки. Было решено выполнить dim перестановок для рассматриваемых тестов, причем таких, чтобы каждая из переменных была старшей в лексикографическом порядке в одной из рассматриваемых перестановок.

Выполнить такие объемные расчеты на данный момент получилось только с тестами, занимающими менее 300 секунд для выполнения, однако полученные результаты были дополнительно перепроверены – вычислительные эксперименты были повторены несколько раз и за итоговое время вычислений взято среднее время вычисления за несколько попыток. Как следствие, есть уверенность в устойчивости полученных результатов.

Так, при использовании порядка, учитывающего степень, алфавитный порядок переменных теряет часть своего значение. Это заметно из устойчивости результатов вычислений с порядком $\prec degrevlex$. Тем не менее, для некоторых тестов заметны значимые различия в скорости выполнения 5.

При том, что количество выполненных редукций различается не более чем на $\pm 10\%$, некоторые тесты показывают ускорение или замедление вычислений в 2-3 раза, в зависимости от используемой перестановки.

Порядок $\prec lex$ оказался ожидаемо менее устойчивым к перестановкам, хоть отловить устойчивую зависимость и сложнее, по уже описанной выше причине – большая часть тестов, решаемых в порядке $\prec lex$, решается крайне быстро. Тем не менее, на примерах некоторых тестов 6 хорошо видно, что определенные перестановки замедляют вычисления в 2-5 раз.

Так, например, для теста cassou, такой "замедляющей"перестановкой оказалась каноническая запись, взятая из тестового файла без изменений. Конечно, занимает вычисление сотые или тысячные доли секунд, не является значимым фактом. Однако, экстраполируя схожую зависимость на задачи, занимающие часы или же дни вычислений, можно представить себе более чем серьезную задачу по определению наиболее оптимального алфавитного порядка переменных.

Таблица 5 — Выдержка из результатов вычислений с перестановками, порядок $\prec$ $degrevlex$, система GInv

Имя	Время	Число редукций
cyclic6	51,03	6414493
cyclic6	41,62	5583844
cyclic6	35,40	5913924
cyclic6	41,85	5673883
cyclic6	47,90	5835013
cyclic6	29,99	5503222
reimer4	11,06	64313
reimer4	12,10	67317
reimer4	4,22	51198
reimer4	4,24	51196

Таблица 6 — Выдержка из результатов вычислений с перестановками, порядок $\prec$ lex, система Sage

Имя	Время
cassou	0,0118
cassou	0,0025
cassou	0,0025
cassou	0,0025
cassou	0,0025
chandra4	0,0086
chandra4	0,0080
chandra4	0,0169
chandra4	0,0067

4.3 Предварительная классификация задач

4.3.1 Внешние характеристики идеала

В связи с тем, что результаты вычислительных экспериментов подтверждают выдвинутые гипотезы о значимом различии в эффективности при использовании разных мономиальных порядков, мономиальных делений или же перестановок переменных, для разных задач, поднимается вопрос о возможности предварительного определения наиболее оптимальной конфигурации вычисления для конкретной задачи.

Для возможного ответа на этот вопрос предлагается сначала решить проблему предсказания времени, затрачиваемого на вычисления в рамках одной конфигурации. Если для каждой рассматриваемой конфигурации мы сможем получать достаточно точные оценки времени вычисления отдельной задачи, то для каждой задачи останется лишь выбрать конфигурацию с наименьшим предсказанным временем вычислений.

Для наиболее наглядного и контролируемого результата, была выбрана система GInv, с порядком $\prec$ *degrevlex*. Это дает нам более заметный разброс значения затрачиваемого времени, а так же большую воспроизводимость результатов.

Для предсказания времени нахождения базиса Гребнера, или же сводящегося к нему инволютивного базиса, мы можем использовать только входную информацию – набор переменных и полиномов, образующих идеал.

Введем следующее определение:

Определение 22. Внешней характеристикой идеала будем считать какое-либо числовое значение, вычислимое за полиномиальное время на основе образующих идеал полиномов.

Таким образом мы получаем некоторую метрику, описывающую данный идеал.

Выделим возможный набор внешних характеристик:

1. размерность идеала;
2. средний коэффициент;

3. максимальный коэффициент;
4. сумма коэффициентов;
5. средняя степень по моному;
6. средняя степень по переменной;
7. максимальная степень по моному;
8. сумма степеней;
9. количество мономов;
10. количество полиномов;
11. количество литеров;
12. среднее количество мономов на полином;
13. среднее количество переменных в мономе;
14. максимальное количество переменных в мономе;
15. сумма степеней НОК - наименьшего общего кратного;
16. сумма остатков от НОК (по всем мономам);
17. сумма остатков от НОК (по старшим мономам).

В данном случае, наименьшее общее кратное – общая степень минимально возможного монома, являющегося кратным для всех имеющихся мономов. Может быть вычисленна как сумма максимальных степеней по каждой переменной.

Также дополним схожими инструментами для оценки вычислительной сложности нахождения базиса Гребнера, предлагаемыми в других работах:

1. Степень регулярности идеала, *Castelnuovo–Mumford regularity*, обозначается reg – это наименьшая степень, до которой нужно генерировать S-полиномы (для классического алгоритма нахождения базиса Гребнера) или расширять инволютивные конусы (в случае поиска инволютивных базисов), чтобы завершить алгоритм.

   Чем выше значение reg, тем дольше работает алгоритм и тем больше промежуточных полиномов генерируется. В данной работе [40] рассматривается экспоненциальная зависимость сложности алгоритмов F4/F5 от $reg(I)$. Эта характеристика вычислется по формуле

   $$reg = max(total\_degree(g.lm())), g \in G,$$

   и соответствует внешней характеристике **максимальная степень по моному**.

2. Плотность системы уравнений, *density* – отношение фактического числа мономов в системе к максимально возможному числу мономов при заданных размерности n и максимально возможной степени монома d. При этом d выступает аналогим характеристики **сумма степеней НОК**.

   Если $total\_monomials$ – общее число мономов во всех уравнениях системы, то:

$$density = \frac{total\_monomials}{C_{n+d}^{d}}$$

   Аналогичная метрика, называемая *sparsity* (разреженность) встречается в работах по решению разреженных систем (*sparse systems*) часто говорят, что система «sparse» (разрежена), если число мономов значительно меньше максимального возможного при данных n и d [41]. В этих работах заключается, что в разреженных системах алгоритмы работают быстрее.

   Также алгоритмы F4/F5 в ходе работы на каждом шаге строят матрицу Маколея, *Macaulay matrix*. Ее размер зависит от числа всех возможных мономов степени $<= d$ [42]. Если в системе мало мономов, то и матрица более разрежена, что ускоряет гауссово исключение. Таким образом, *density* – это нормированная мера заполненности матрицы Маколея, и это прямо связано с вычислительной сложностью F4/F5.

Отдельного упоминания заслуживают следующие метрики:

1. Максимальная степень полиномов во время вычисления, Solving Degree, d_solve. Используется в алгоритмах в F5/F4, и соответствует реально достигнутой степени полиномов, непосредственно в промежуточных шагах вычислений. В F5 – степень, при которой все новые S-полиномы/продолжения нулевые [43].

$$d\_solve = maxdeg(p),$$

   где p – полином, который добавляется в базис или редуцируется в алгоритме.

2. First Fall Degree / Last Fall Degree

   First Fall – степень, при которой впервые появляются ненулевые отношения (в XL/mutant стратегиях). Last Fall - степень, при которой все

отношения найдены. В криптографии является хорошим предиктором вычислительной сложности. [44]

3. Количество немультипликативных продолжений.

   Метрика для инволютивных алгоритмов, число продолжений мономов на немультипликативные переменные. Чем больше – тем больше выполняется редукций и время. Дает лучшею корреляцию, нежели *density*, т.к. учитывает структуру инволютивных конусов [45].

$$\sum u \in U|NM(u, L, U)|,$$

   где NM - немультипликативные переменные для монома u.

4. Верхняя граница степени в базисе, *Maximal Expected Degree*, Macaulay Bound

$$bound = 1 + \sum d_i - 1,$$

   где d_i – степени генераторов, [46].

5. Доля бесполезных редукций, Fraction of Zero Reductions. Соотношение числа редукций к 0 к общему числу рассмотренных S-пар [47].

Однако их вычисление зависит от данных, получаемых только в процессе построение базиса, и, как следствие, они не могут быть использованы для предварительного предсказания времени вычисления.

4.3.2 Разбиение на классы

Предсказать время вычислений как непрерывную величину – трудная задача. Значительно проще решить задачу классификации – например, определить к какому классу по времени вычислений принадлежит конкретная задача. Для этого требуется выделить правила определения классов, и их общее количество.

Увеличение количества определяемых классов снижает показатели корреляции между классом и внешними характеристиками идеала. В будущем, с расширением набора тестов, это снижение возможно будет нивелировано. Кроме того, ошибка на один-два класса может быть не столь существенной,

при определении одного из десяти возможных классов. Тем не менее, в рамках данной работы, для предварительного выделения наибольших корреляций, рассматривается разбиение только на два класса.

Дополнительно рассматриваются значения корреляции Пирсона между внешними характеристиками и объемом используемой оперативной памяти. Однако для большинства успешно вычисленных тестов замеренный объем используемой памяти не превышает 500 Мб, что является практически несущественным требованием для современного вычислительного оборудования.

4.3.3 Выявленные корреляции для классов по порогам в 300 и 3000 секунд

В первую очередь, рассматривается разбиение классов по времени вычислений. Все решенные задачи деляться на два класса – "быстрые"и "долгие". Важным является определение порогового значения. Эмпирически было получено наиболее оптимальное значение в 300 секунд, дающие наибольшую корреляцию с рассматриваемыми метриками.

Рассмотрим разбиение на классы с порогов в 300 и 3000 секунд соответственно. Таблицы 7, 8, 9 демонстрируют рост значений корреляции при переходе от времени, к классу по порогу в 3000 секунд, по 300 секунд.

Альтернативно рассматривалась корреляция внешних характеристик идеала с логарифмом от времени вычислений, что также дает более высокие значения, нежели просто корреляция от времени вычислений. Тем не менее, разбиение на классы по оптимальному порогу дает более качественные значения корреляции.

Как видно из результатов вычислений 7, размерность, сумма степеней и количество мономов, ожидаемо оказывают значимое влияние на скорость вычислений. Несмотря на то, что корреляция непосредственно со временем показывает значения < 0.20, т.е. статистически незначимое, при оптимальном разбиении на классы, корреляции выростают до ≈ 0.40.

Результаты корреляций таких характеристик, как средний коэффициент, не преведены, т.к. оказались около нулевыми. Такой результат можно ожидать, т.к. величина коэффициентов имеет малое значение в современных системах

Таблица 7 — Выдержка из таблицы корреляций

Имя	Решение	Время	Раз-ть	Сумма степеней	Кол-во мономов
assur44	есть	940,24	8	201	103
...	...	...	...	...	...
vermeer	есть	3,58	5	37	17
Корр-ия	Время	1,00	0,18	0,21	0,17
...	Память	0,35	0,18	0,21	0,16
...	Класс <3000	0,84	0,27	0,26	0,19
...	Класс <300	0,54	0,39	0,42	0,43

компьютерной алгебры, использующих встроенную факторизацию или же модулярную арифметику.

Количество полиномов и литеров также демонстрирует значимую корреляцию. При этом максимальное, как и среднее, количество литеров в мономе, неожиданно показывает практическое отсутствие корреляции 8, так же как и максимальная сумма степеней по моному, соответствующая вышеописанной метрике reg.

Таблица 8 — Выдержка из таблицы корреляций

Имя	Кол-во полиномов	Кол-во литеров	Макс. кол-во литеров в мономе
assur44	8	193	3
...	...	...	...
vermeer	4	25	3
Время	0,14	0,15	0,00
Память	0,15	0,16	-0,10
Класс <3000	0,17	0,15	0,01
Класс <300	0,38	0,40	0,04

Сумма степеней НОК показывает некоторую значимость, но еще лучше ее демонстрирует сумма остатков от НОК по всем мономам образующих идеала 9. Сумма остатков от НОК, LCM_rem, вычисляется по формуле:

$$LCM\_rem = \sum LCM - deg(m), m \in I.$$

Таблица 9 — Выдержка из таблицы корреляций

Имя	Сумма степеней НОК	Сумма остатков от НОК (по всем мономам)	Сумма остатков от НОК (по старшим мономам)
assur44 ... vermeer	16 ... 11	1447 ... 150	120 ... 33
Время	0,22	0,17	0,15
Память	0,21	0,21	0,21
Класс <3000	0,30	0,25	0,21
Класс <300	0,32	0,39	0,35

Таким образом, наиболее высокие показатели корреляции показывают следующие характеристики: размерность, сумма степеней,количество мономов, количество полиномов, количество литеров, сумма остатков от НОК (по всем мономам).

4.3.4 Выявленные корреляции для классов по наличию решения

Значения корреляции заметно отличаются, если рассматривать также и внешние характеристики тестов, для которых не получилось выполнить вычисления в разумное время. В данном случае, разделим 135 тестов на два класса – 100 успешно посчитанных, и 35 непосчитанных. Наиболее высокие значения корреляции с внешними характеристиками отображены на следующих таблицах 10, 11, 12.

Интересно, что корреляция с размерностью, ранее показывающая высокий результат, теперь оказывается практически несуществующей 10. При этом сумма степеней и количество мономов падают с ≈ 0.40 7, до ≈ 0.30. Напротив, средняя степень по моному, ранее практически не значимая, получает корреляцию, также ≈ 0.30.

Количество мономов и литеров также демонстрирует падение корреляции с ≈ 0.40 7, 8, до ≈ 0.35 11, и 0.23 для количества полиномов.

Таблица 10 — Выдержка из таблицы корреляций

Имя	Решение	Время	Раз-ть	Средняя степень по моно-му	Сумма степеней
assur44	есть	940,24	8	1,94	201
...	...	...	...	...	...
vermeer	есть	3,58	5	2,12	37
Корр-ия	Время	1,00	0,18	0,05	0,21
...	Память	0,35	0,18	0,08	0,21
...	Класс	0,75	0,016	0,31	0,30

Таблица 11 — Выдержка из таблицы корреляций

Имя	Кол-во моно-мов	Кол-во полиномов	Кол-во литеров
assur44	103	8	193
...	...	...	...
vermeer	17	4	25
Время	0,17	0,14	0,15
Память	0,16	0,15	0,16
Класс	0,34	0,23	0,35

Аналогичное падение демонстрируют и характеристики, связанные с НОК 12.

Также повышается корреляция у ранее практически незначимых характеристик: среднее количество мономов на полином, среднее и максимальное количество литеров в мономе 13. Так среднее количество литеров в мономе достигает рекордного значения корреляции $= 0.49$.

Замечание. Следует также отметить чувствительность данных к незначительным изменениям.

Так, например, корреляция **по времени** будет разительно отличаться, если для теста remier8 указать в таблице не стандартное для непосчитанных тестов время 18000, а указать фактическое для конкретно этого теста время

Таблица 12 — Выдержка из таблицы корреляций

Имя	Сумма степеней НОК	Сумма остатков от НОК (по всем мономам)	Сумма остатков от НОК (по старшим мономам)
assur44	16	1447	120
...	...	...	...
vermeer	11	150	33
Время	0,22	0,17	0,15
Память	0,21	0,21	0,21
Класс	0,28	0,33	0,18

Таблица 13 — Выдержка из таблицы корреляций

Имя	Среднее мономов на полином	Среднее кол-во литеров в мономе	Максимальное кол-во литеров в мономе
assur44	12,88	1.87	3
...	...	...	...
vermeer	4.25	1.47	3
Время	0,07	0,10	0,00
Память	0,08	-0,06	-0,09
Класс	0,35	0,49	0,42

расчетов – 3628800 секунд, или же 42 дня, по итогам которых все еще не был получен результат.

При таком изменении, корреляция характеристики **Средняя степень по литеру** падает с 0.28 до околонулевого значения, а характеристика **Сумма степеней**, наоборот, возрастает с 0.13 до 0.21, а **Сумма степеней НОК** – с 0.22 до 0.45!

В совокупности, результаты, полученные в ходе исследования выдвинутых гипотез, позволяют задаться вопросом о реализации автоматизированной системы для классификации вычислительных задач и алгоритмов [54], и их последующего сопоставления.

4.4 Результаты предварительной классификации задач

Полученные данные о корреляции Пирсона между временем вычислений и внешними характеристиками демонстрируют достаточно высокую зависимость для разработки и реализации программного инструмента для классификации задач по времени вычисления. Несмотря на то, что ни одна характеристика не имеет подавляющей значимости, с корреляцией > 0.60, в совокупности набор ведущих характеристик можно использовать для классификации задач по длительности вычислений, и как следствие – приближенной оценки времени вычисления.

Разработка инструмента, предсказывающего хотя бы порядок времени, требуемого для вычисления той или иной задачи, значительно повысит эффективность проводимых исследований. Это поможет более эффективно планировать проведение вычислительных экспериментов и распределение располагаемых мощностей. Также это позволит обоснованно подходить к вопросу о приобретении дополнительного оборудования или аренде облачных мощностей.

Наиболее эффективным решением задачи классификации по неявным закономернотям является использование методов машинного обучения. Так, например, в работах [48], [49] выделенные численные характеристики имели корреляцию с предсказываемым показателем не более 0.30. Однако их объединение в рамках моделей машинного обучения, таких как модель случайного леса, *Random Forest*, и многослойного перцептрона, *MultiLayer Perceptron, MLP*, позволило получить *f-score* бинарной классификации > 0.80.

Во многом именно участие автора в этих работах и привело к мысли о выделении внешних характеристик и их использования для предсказания вычислительной сложности нахождения базисов Гребнера.

4.4.1 Набор данных

Основным препятствием, мешающим приступить к обучению классификационных моделей, является недостаточный объем данных, т.е. недостаточное количество рассматриваемых тестов. Для получения устойчивых результатов,

с возможностью проведения кроссвалидации, следует расширить набор минимум до тысячи тестов.

Одним из предлагаемых способов расширения набора тестовых данных является видоизменение уже имеющихся тестов. Случайные изменения степеней переменных в составах мономов, или же добавление/удаление мономов из полиномов, может исказить физический смысл задачи, а также кардинально изменить вычислительную сложность построения базиса.

Однако, возможно сохранить физический смысл задачи, при этом контролируемо изменяя её, а также вычислительную сложность. Для этого проведем замену переменных. Например, для теста cyclic5:

$$
\begin{aligned}
&\text{"}variables\text{"} : [x1, x2, x3, x4, x5],\\
&\text{"}equations\text{"} :\\
&\quad x1 + x2 + x3 + x4 + x5,\\
&\quad x1 * x2 + x1 * x5 + x2 * x3 + x3 * x4 + x4 * x5,\\
&\quad x1 * x2 * x3 + x1 * x2 * x5 + x1 * x4 * x5 + x2 * x3 * x4 + x3 * x4 * x5,\\
&\quad x1 * x2 * x3 * x4 + x1 * x2 * x3 * x5 + x1 * x2 * x4 * x5 +\\
&\qquad x1 * x3 * x4 * x5 + x2 * x3 * x4 * x5,\\
&\quad x1 * x2 * x3 * x4 * x5 - 1
\end{aligned}
$$

Выполним следующую замену переменных:

$$
\begin{aligned}
x1 &\to 2 * y1,\\
x2 &\to 3 * y2,\\
x3 &\to 5 * y3,\\
x4 &\to 7 * y4,\\
x5 &\to 11 * y5
\end{aligned}
$$

В результате получаем следующий тест, в данном случае, отличающийся коэффициентами.

$$
\begin{aligned}
&\text{"}variables\text{"} : [y1, y2, y3, y4, y5],\\
&\text{"}equations\text{"} :\\
&\quad 2 * y1 + 3 * y2 + 5 * y3 + 7 * y4 + 11 * y5,\\
&\quad 6 * y1 * y2 + 22 * y1 * y5 + 15 * y2 * y3 + 35 * y3 * y4 + 77 * y4 * y5,\\
&\quad 30 * y1 * y2 * y3 + 55 * y1 * y2 * y5 + 154 * y1 * y4 * y5 +\\
&\qquad 105 * y2 * y3 * y4 + 385 * y3 * y4 * y5,\\
&\quad 210 * y1 * y2 * y3 * y4 + 330 * y1 * y2 * y3 * y5 + 462 * y1 * y2 * y4 * y5 +\\
&\qquad 770 * y1 * y3 * y4 * y5 + 1155 * y2 * y3 * y4 * y5,\\
&\quad 2310 * y1 * y2 * y3 * y4 * y5 - 1
\end{aligned}
$$

Время вычисления теста, после произведенной замены переменных, выросло в 3.5 раза, с 0.85 до 3.04 секунд. Аналогичным образом можно изменять степень переменных в мономе, либо количество используемых переменных, т.е. литеров.

Также, как следует из замечания 4.3.4, следует расширить набор данных записями с успешным вычислением со временем более нескольких часов. В идеале, получить достаточно данных для полноценного формирования следующих классов задач: вычисление менее секунды, вычисление менее 5 минут, вычисление менее часа, вычисление менее 8 часов, вычисление менее суток, вычисление более суток.

Заключение

В настоящей диссертационной работе достигнута поставленная цель по разработке эффективного численного метода для изучения нелинейных математических моделей. В ходе исследования успешно решены все поставленные задачи.

Было выполнено оригинальное исследование о нахождении стационарных решений для популяционных моделей, найденных через вычисление базиса Гребнера 1.2.

Было выполнено оригинальное исследование о нахождении периодических решений для динамических систем, также через вычисление базиса Гребнера 1.3.

Проведён детальный анализ инволютивного подхода к вычислению базисов Гребнера, включая описание классического и инволютивного алгоритма нахождения базиса Гребнера, определение понятия инволютивного деления, приведение доказательства сводимости инволютивных базисов к базисам Гребнера.

Описана специализированная система GInv, выполняющая поиск инволютивных базисов. На её основе реализован программный комплекс GInvDist на языке Python, с использованием библиотеки символьных вычислений SymPy, опубликованный в открытом репозитории пакетов Python PyPI. Подробно описана работа этого комплекса. Разработанный комплекс облегчает решение как нелинейных систем уравнений, так и систем дифференциальных уравнений.

Доказан ряд теорем о линейном и инволютивном делении полиномов, а также о свойствах остатков от таковых делений. Выдвинут и подтверждён ряд гипотез о различной вычислительной сложности нахождения базиса для различных мономиальных порядков и делений.

Разработан специализированный скрипт на языке Python, выполняющий автоматизированное тестирование и сравнение эффективности алгоритмов вычисления базисов Гребнера в системах компьютерной алгебры.

Для реализованного программного обеспечения проведено тестирование на объёмной выборки тестов с задачами, представимыми в виде систем нелинейных уравнений. Тестирование продемонстрировало: 23 тестов из 135 (17%) не получили решения ни в одной из этих систем, с ограничением по времени

вычисления в два часа; стандартный алгоритм дал решение для 108 тестов (80%); инволютивный алгоритм дал решение для 100 тестов (74%). При этом инволютивный алгоритм даёт решение для 4 тестов, не решаемых стандартным алгоритмом, а стандартный – для 12 тестов, не решаемых инволютивным. Это подтверждает значимость подбора разных алгоритмов для разных задач.

В частности, в числе решённых данным инструментарием задач – алгебраические системы из задач математического моделирования популяционной динамики (обобщённые модели конкуренции и миграции), химической кинетики (циклические системы), а также задачи из теории кодирования и робототехники.

Рассматривается нахождение стационарных и периодических приближённых решений для задач типа нелинейного осциллятора, что показывает применимость разработанных инструментов как к задачам поиска точек равновесия, так и к задачам построения периодических траекторий.

Также в диссертации рассмотрены два примера многовидовых моделей конкуренции и миграции типа Лотки-Вольтерра. В первом воспроизводится четырехмерная модель из работы [1]: применение алгоритма Бухбергера позволяет получить корни с алгебраической точностью там, где численный метод, использованный в исходной работе, обеспечивал одну верную цифру после запятой. Во втором, более сложном примере демонстрируется успешное применение того же метода для шестимерной модели. Разработанный инструментарий справляется с этой задачей, и демонстрирует возможность использования в моделях с большей размерностью, – тем самым подтверждая универсальность предложенного подхода.

Перспективы дальнейших исследований включают в себя адаптацию системы для использования распределённых и параллельных вычислений, изучение различных классов решаемых задач, сбор более обширного набора данных и использование моделей машинного обучения для более эффективной предварительной классификации.

Результаты диссертации могут быть использованы в системах компьютерной алгебры, математическом моделировании и образовательных курсах по программированию и компьютерной алгебре.

Работа подтверждает перспективность использования инволютивных алгоритмов для решения систем нелинейных уравнений.

Список литературы

1. *Vasilyeva, I. I.* Qualitative and numerical study of multidimensional migration and population models with competition [Текст] : дис. ... канд. / Vasilyeva I. I. — Moscow : RUDN university, 2026. — In Russian.
2. *J., L. A.* Elements of Physical Biology [Текст] / L. A. J. — Baltimore: Williams, Wilkins, 1925.
3. *V., V.* Fluctuations in the abundance of a species considered mathematically [Текст] / V. V. // Nature. — 1926. — Т. 118. — С. 558—560.
4. *Жаботинский, А. М.* Концентрационные автоколебания [Текст] / А. М. Жаботинский. — Москва : Издательство Наука», 1974.
5. *Моисеев, Н. Н.* Математика ставит эксперимент [Текст] / Н. Н. Моисеев. — Москва : «НАУКА ГЛАВНАЯ РЕДАКЦИЯ ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ, 1977.
6. *Cox, D.* Ideals, varieties, and algorithms [Текст] / D. Cox, J. Little, D. O'Shea. — 3rd ed. — Springer, 2007.
7. *Blinkov, Y.* Specialized computer algebra system GINV [Текст] / Y. Blinkov, V. Gerdt // Programming and Computer Software. — 2008. — Т. 34. — С. 112—123.
8. *Gerdt, V. P.* DInvolutive divisions of monomials [Текст] / V. P. Gerdt, Y. A. Blinkov // Programmirovanie. — 1998. — Т. 24, № 6. — С. 22—24.
9. *Gerdt, V. P.* Involutive Bases of Polynomial Ideals [Текст] / V. P. Gerdt, Y. A. Blinkov // Preprint JINR E5-97-3. — Dubna, 1997.
10. *Zharkov, A. Y.* Involution approach to solving systems of algebraic equations [Текст] / A. Y. Zharkov, Y. A. Blinkov // Proceedings of the 1993 International IMACS Symposium on Symbolic Computation. — Laboratoire d'Informatique Fondamentale de Lille, France, 1993. — С. 11—16.
11. *Zharkov, A. Y.* Involutive Polynomial Bases: General Case [Текст] / A. Y. Zharkov // Preprint JINR E5-94-224. — Dubna, 1994.

12. *Blinkov, Y. A.* Involutive methods for studying models described by algebraic and differential equation systems [Текст] : дис. ... канд. / Blinkov Yu. A. — Saratov, 2023. — In Russian.

13. *Gerdt, V. P.* Involutive Division Generated by an Antigraded Monomial Ordering [Текст] / V. P. Gerdt, Y. A. Blinkov // Computer Algebra in Scientific Computing / под ред. V. P. Gerdt [и др.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. — С. 158—174.

14. Страница системы GInv на репозитории Гит-хаб [Текст]. — URL: https://github.com/blinkovua/GInv (дата обр. 22.09.2025).

15. *Беляев, А. В.* Математическое моделирование и анализ стохастической динамики дискретных популяций [Текст] : дис. ... канд. / Беляев А. В. — Екатеринбург : ФГАОУ ВО «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина», 2025.

16. *Ризниченко, Г. Ю.* Динамика популяций [Текст] / Г. Ю. Ризниченко. — Москва : Издательство Юрайт, 2026.

17. *Хакен, Г.* Синергетика, Перевод с английского [Текст] / Г. Хакен. — Москва : Издательство «Мир», 1980.

18. *Buchberger, B.* Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal [Текст] : дис. ... канд. / Buchberger B. — Universit¨at Innsbruck, 1965.

19. *Vasilyeva, I. I.* Construction and research of a population dynamic model of the "two competitors - two migration areas"type [Текст] / I. I. Vasilyeva, O. V. Druzhinina, O. N. Masina // Nonlinear world. — 2022. — Т. 20, № 4. — С. 60—68.

20. Computer research of deterministic and stochastic models two competitors two migration areas taking into account the variability of parameters [Текст] / I. I. Vasilyeva [и др.] // Discrete and Continuous Models and Applied Computational Science. — 2024. — Т. 32, № 1. — С. 60—73.

21. Construction, stochastization and computer study of dynamic population models two competitors two migration areas [Текст] / I. I. Vasilyeva [и др.] // Discrete and Continuous Models and Applied Computational Science. — 2023. — Т. 31, № 1. — С. 27—45.

22. *Baddour, A.* On periodic approximate solutions of dynamical systems with a quadratic right-hand side [Текст] / A. Baddour, M. D. Malykh, L. A. Sevastianov // Math. Sci. — 2022. — Т. 261. — С. 698—708.

23. *А. Баддур М. Д. Малых, Л. А. С.* О периодических приближенных решениях динамических систем с квадратичной правой частью [Текст] / Л. А. С. А. Баддур М. Д. Малых // Записки научного семинара ПОМИ. — 2021. — Т. 507. — С. 157—172.

24. *Popov, A. S.* Search for the best cubature formulas for a sphere that are invariant with respect to the rotation group of an octahedron [Текст] / A. S. Popov // Siberian Journal computational mathematics. — 2002. — Т. 5:4. — С. 367—372.

25. *Zharkov, A. Y.* Algorithm for constructing involutive bases of polynomial ideal [Текст] / A. Y. Zharkov, Y. A. Blinkov // International Conference on Interval and Computer-Algebraic Methods in Science and Engineering. — St-Petersburg, 1994. — С. 258—260.

26. *Zharkov, A. Y.* Involutive Bases of Zero-Dimensional Ideals [Текст] / A. Y. Zharkov, Y. A. Blinkov // Preprint JINR E5-94-318. — Dubna, 1994.

27. *Zharkov, A. Y.* Solving zero-dimensional involutive systems [Текст] / A. Y. Zharkov, Y. A. Blinkov // Progress in Mathematics. Т. 143. — Basel : Birkhauser, 1996. — С. 389—399.

28. *Zharkov, A. Y.* Involution approach to investigating polynomial systems [Текст] / A. Y. Zharkov // Mathematics and Computers in Simulation. — 1996. — Т. 42. — С. 323—332.

29. *Gerdt, V. P.* Gröbner bases and involutive methods for algebraic and differential equations [Текст] / V. P. Gerdt // Computer Algebra in Science and Engineering. — Singapore : World Scientific, 1995. — С. 117—137.

30. *Gerdt, V. P.* Involutive Polynomial Bases [Текст] / V. P. Gerdt, Y. A. Blinkov // Publication IT-95-271. — Laboratoire d'Informatique Fondamentale de Lille, 1995.

31. *Gerdt, V. P.* Involutive Bases of Polynomial Ideals [Текст] / V. P. Gerdt, Y. A. Blinkov // Preprint-Nr.1/1996. — Naturwissenschaftlich-Theoretisches Zentrum, University of Leipzig, 1996.

32. *Gerdt, V. P.* Gröbner bases and involutive methods for algebraic and differential equations [Текст] / V. P. Gerdt // Mathematical and computer modelling. — 1997. — Т. 26. — С. 75—90.

33. *Gerdt, V. P.* Involutive divisions in mathematica: Implementation and some applications [Текст] / V. P. Gerdt // Proceedings of the Rhein Workshop on Computer Algebra. — Institute for Algorithms, Scientific Computing, GMD-SCAI, 1998. — С. 74—91.

34. *Blinkov, Y. A.* Divisions and algorithms for the problem of belonging to to an ideal [Текст] / Y. A. Blinkov // Izvestija Saratovskogo universiteta. — 2001. — Т. 1, № 2. — С. 156—167.

35. *Blinkov, Y. A.* Involutive methods of studying models described by algebraic and differential equation systems [Текст] : дис. ... канд. / Blinkov Yu. A. — Saratov : RUDN university, 2009. — In Russian.

36. *Janovich, D. A.* Parallel implementation of the algorithm for computing Grobner and Janet bases at the level of polynomial reduction [Текст] / D. A. Janovich // Discrete and Continuous Models and Applied Computational Science. — 2010. — Т. 3—2. — С. 19—24.

37. Analytical study of cubature formulas on a sphere in computer algebra systems [Текст] / R. E. Bayramov [и др.] // Journal of Computational Mathematics and Mathematical Physics. — 2023. — Т. 63:1. — С. 77—85.

38. *Morgan, A.* Solving polynomial systems using continuation for engineering and scientific problems [Текст] / A. Morgan. — Prentice-Hall, Englewood Cliffs, New Jersey.

39. *Björck, G.* A faster way to count the solutions of inhomogeneous systems of algebraic equations, with applications to cyclic n-roots [Текст] / G. Björck, R. Froberg // Journal of Symbolic Computation. — 1991. — Сент. — Т. 12. — С. 329—336.

40. *Bardet, M.* On the complexity of the F5 Gröbner basis algorithm [Текст] / M. Bardet, J.-C. Faugère, B. Salvy // Journal of Symbolic Computation. — 2015. — Т. 70. — С. 49—70. — URL: https://www.sciencedirect.com/science/article/pii/S0747717114000935.

41. *Faugère, J.-C.* On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations [Текст] / J.-C. Faugère, M. Bardet, B. Salvy //. — 2004. — URL: https://api.semanticscholar.org/CorpusID:7982329.

42. *Faugére, J.-C.* A new efficient algorithm for computing Gröbner bases (F4) [Текст] / J.-C. Faugére // Journal of Pure and Applied Algebra. — 1999. — Т. 139, № 1. — С. 61—88. — URL: https://www.sciencedirect.com/science/article/pii/S0022404999000055.

43. *Ding, J.* Solving Degree and Degree of Regularity for Polynomial Systems over a Finite Fields [Текст] / J. Ding, D. Schmidt // Lecture Notes in Computer Science. — 2013. — Янв.

44. *Yeh, J.* Operating Degrees for XL vs. F4/F5 for Generic $\mathcal{MQ}$ with Number of Equations Linear in That of Variables [Текст] / J. Yeh, C.-M. Cheng, B.-Y. Yang // Lecture Notes in Computer Science. — 2013. — Янв.

45. *Gerdt, V. P.* Involutive bases of polynomial ideals [Текст] / V. P. Gerdt, Y. A. Blinkov // Mathematics and Computers in Simulation. — 1998. — Т. 45. — С. 519—541. — URL: https://api.semanticscholar.org/CorpusID:10243294.

46. *Dubé, T.* The Structure of Polynomial Ideals and Gröbner Bases [Текст] / T. Dubé // SIAM J. Comput. — 2013. — Т. 19. — С. 750—773. — URL: https://api.semanticscholar.org/CorpusID:41206771.

47. *Eder, C.* Signature-based algorithms to compute Gröbner bases [Текст] / C. Eder, J. Perry // CoRR. — 2011. — Янв. — Т. abs/1101.3589.

Публикации автора по теме диссертации

В изданиях, входящих в международную базу цитирования Web of Science

48. OperatorEYEVP: Operator Dataset for Fatigue Detection Based on Eye Movements, Heart Rate Data, and Video Information [Текст] / S. Kovalenko

[и др.] // Sensors. — 2023. — Т. 23, № 13. — URL: https://www.mdpi.com/1424-8220/23/13/6197.

49. Intelligent Human Operator Mental Fatigue Assessment Method Based on Gaze Movement Monitoring [Текст] / A. Kashevnik [и др.] // Sensors. — 2024. — Т. 24, № 21. — URL: https://www.mdpi.com/1424-8220/24/21/6805.

В прочих изданиях

50. Страница программного комплекса GInvDist на репозитории Гит-хаб [Текст]. — URL: https://github.com/TheMumblingMammoth/GInvDist (дата обр. 21.02.2026).

51. *А., М. А.* Application of Grobner bases for finding accurate states of equilibrium of migration and population models [Текст] / М. А. А. // Mathematical Modelling and Geometry. — 2026. — Т. 14, № 1. — С. 1—8.

52. Разработка системы для оценки производительности алгоритмов компьютерной алгебры при нахождении базимов Грёбнера [Текст] / Ю. А. Блинков [и др.] // КИО. — 2024. — № 2.

53. Страница программы для тестирования системы GInv на репозитории Гит-хаб [Текст]. — URL: https://github.com/MamonovAnton/ginv_testing (дата обр. 22.09.2025).

54. *Блинков, Ю. А.* Реализация автоматизированной системы классификации и оценки эффективности алгоритмов вычисления базисов Грёбнера [Текст] / Ю. А. Блинков, С. И. Салпагаров, А. А. Мамонов // Записки научных семинаров Санкт-Петербургского отделения математического института им. В.А. Стеклова. — 2025. — С. 56—67.

Список рисунков

Список таблиц