

**МАТЕМАТИКА 5.\* ДЛЯ НЕМАТЕМАТИКА**

Николай Вавилов, Владимир Халин

Computers are not intelligent. They only think they are. — Компьютеры не могут мыслить; они лишь думают, что могут мыслить.

Рене Декарт

The purpose of computing is insight, not numbers! — Целью вычисления является понимание, а не числа!

Richard Hamming

If you can't learn to do it well, learn to enjoy doing it badly! — Если вы не можете научиться делать это хорошо, научитесь получать удовольствие от того, что вы делаете это как умеете!

The Tao of Real Programming

М  
М А М  
М А Т А М  
М А Т Н Т А М  
М А Т Н Е Н Т А М  
М А Т Н Е М Е Н Т А М  
М А Т Н Е М А М Е Н Т А М  
М А Т Н Е М А Т А М Е Н Т А М  
М А Т Н Е М А Т И Т А М Е Н Т А М  
М А Т Н Е М А Т И С И Т А М Е Н Т А М  
М А Т Н Е М А Т И С А С И Т А М Е Н Т А М

## ВЫПУСК 1. ПЕРВОЕ ЗНАКОМСТВО С СИСТЕМОЙ Mathematica

ПРЕДИСЛОВИЕ .....	7
§ 1. Обзор литературы по компьютерной алгебре .....	9
§ 2. Благодарности .....	12
Гл. 1. ЧТО ТАКОЕ КОМПЬЮТЕРНАЯ АЛГЕБРА? .....	13
§ 1. Математика и компьютеры .....	15
§ 2. Компьютерная алгебра .....	17
§ 3. Компьютерная алгебра в преподавании .....	19
§ 4. Влияние компьютеров на математическое мышление .....	21
§ 5. Системы компьютерной алгебры: общая классификация .....	24
§ 6. Системы общего назначения: Axiom, Derive, Maple, Mathematica, MuPAD .....	28
§ 7. Другие математические пакеты: MatLab и Mathcad .....	30
§ 8. Системы компьютерной алгебры как языки программирования сверхвысокого уровня .....	32
§ 9. Возможности систем компьютерной алгебры .....	34
§ 10. Об “ошибках” компьютерной алгебры .....	36
Гл. 2. ЧТО ТАКОЕ Mathematica? .....	42
§ 1. Достоинства системы Mathematica .....	42
§ 2. Структура системы Mathematica .....	44
§ 3. Главное меню .....	47
§ 4. Getting help .....	49
§ 5. Палитры .....	53
§ 6. Сессии и вычисления .....	56
§ 7. Блокноты и клетки .....	57
§ 8. Пакеты расширений .....	59
§ 9. Концептуальное программирование вместо процедурного и рекурсивного .....	64
§ 10. Простейшие правила и типичные ошибки .....	65
§ 11. Несколько общих советов .....	67
Гл. 3. ПРАКТИЧЕСКОЕ ВВЕДЕНИЕ В СИСТЕМУ Mathematica .....	69
§ 1. Арифметика .....	72
§ 2. Многочлены и рациональные дроби .....	77
§ 3. Решение алгебраических уравнений .....	83
§ 4. Решение систем уравнений и неравенств .....	92
§ 5. Элементарные функции .....	100
§ 6. Графики функций .....	109
§ 7. Суммы, произведения, пределы, .....	125
§ 8. Дифференцирование .....	133
§ 9. Интегрирование .....	139

§ 10. Решение дифференциальных уравнений .....	147
§ 11. Запись векторов и матриц .....	153
§ 12. Линейная алгебра .....	161
GRAPHICS GALLERY .....	171

#### СОДЕРЖАНИЕ ДРУГИХ ВЫПУСКОВ:

##### ВЫПУСК 2. ОСНОВЫ СИНТАКСИСА

Гл. 1. ОБЪЕКТЫ И ВЫРАЖЕНИЯ

Гл. 2. ПЕРЕМЕННЫЕ И ИХ ЗНАЧЕНИЯ

Гл. 3. ФУНКЦИИ

##### ВЫПУСК 3. ОСНОВЫ ПРОГРАММИРОВАНИЯ

Гл. 1. ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Гл. 2. СПИСКИ

Гл. 3. FLOW CONTROL

##### ВЫПУСК 4. РАБОТА С ГРАФИКОЙ И ТЕКСТОМ

Гл. 1. ДВУМЕРНАЯ ГРАФИКА

Гл. 2. ТРЕХМЕРНАЯ ГРАФИКА

Гл. 3. ВВОД И ВЫВОД

## ПРЕДИСЛОВИЕ

Бывает, что человек обладает знанием, но не умеет пользоваться им. Бывает и так, что владеющий искусством сам не знает его секрета. В царстве Вэй жил человек, умевший хорошо считать. Перед смертью он поведал секрет своего искусства сыну. Сын запомнил слова, но не знал, как применить их. Какой-то человек стал его расспрашивать об этом, и он передал ему те слова. И тот человек научился считать не хуже, чем это делал отец.

Ле-цзы, Гл. VIII. Рассказы о совпадениях

Человек может пользоваться только тем, чем он умеет пользоваться.

Человек может использовать только то, что он знает, как использовать

Идрис Шах, ‘Сказки дервишей’

Was man nich versteht, besitzt man nicht. — Чем человек не понимает, то ему не принадлежит.<sup>1</sup>

Johann Wolfgang von Goethe, Maximen und Reflexionen

Nessuna certezza è dove non si pò applicare la matematica. — Там, где нельзя применить *Mathematica*, не может быть никакой достоверности.

Leonardo da Vinci

They spell it *da Vinci* and pronounce it *da Vinchy*. Foreigners always spell better than they pronounce. — Они пишут *da Vinci*, а произносят *да Винчи*. Иностранцы всегда пишут гораздо лучше, чем говорят.

Mark Twain

Описываемая в настоящей брошюре система *Mathematica* является СИСТЕМОЙ КОМПЬЮТЕРНОЙ АЛГЕБРЫ ОБЩЕГО НАЗНАЧЕНИЯ, при помощи которой можно решать **любой** тип задач, в которых в той или иной форме

---

<sup>1</sup>Канонический русский перевод этого афоризма таков: “Чем человек не владеет, тем он не обладает” — однако при этом вводится игра слов, которой нет в оригинале. Между тем, господин тайный советник абсолютно категоричен: “Чем человек не умеет пользоваться, **того у него нет**” — WAS DU ERERBT VON DEINEN VATERN HAST, ERWIRB ES, UM ES ZU BESITZEN.

встречается математика. При этом *Mathematica* наряду с *Maple* является **единственной** такой **high-end**<sup>2</sup> системой, которая настолько проста в использовании, что доступна школьникам и студентам младших курсов.

*Mathematica*, как и любая система компьютерной алгебры, может излагаться с точки зрения

- математика,
- программиста,
- пользователя.

Пользователь интересуется этой системой не с точки зрения принципов ее работы, а как законченным программным продуктом для проведения практических вычислений в конкретной предметной области, скажем, в математике, физике, информатике, технических науках, химии, астрономии, геологии, экономике, управлении, проектировании, архитектуре, лингвистике, компьютерной графике, музыке и т.д. При этом он совершенно не обязан понимать ни математических основ работы системы, ни используемых в ней алгоритмов, ни собственно программистских аспектов ее реализации, ее взаимодействия с платформой, операционной системой и другими программами и других подобных вещей. Тем не менее, любой *серьезный* пользователь, который хочет использовать эти системы не просто в качестве редактора формул и большого научного калькулятора (в духе *Scientific Workplace* или *Mathcad*), любой *грамотный* пользователь, который хочет избежать хотя бы наиболее очевидных ошибок и эффективно использовать возможности этих систем, должен понимать хотя бы основные принципы компьютерной алгебры, взаимоотношение математической и вычислительной точек зрения и, в первую очередь, принятые в этих системах стандартные соглашения и специфику используемого ими **языка**.

Предлагаемое учебное пособие рассчитано как раз на такого **серьезного** пользователя и состоит из четырех частей:

- Часть 1. ПЕРВОЕ ЗНАКОМСТВО,
- Часть 2. ОСНОВЫ СИНТАКСИСА,
- Часть 3. ОСНОВЫ ПРОГРАММИРОВАНИЯ,
- Часть 4. РАБОТА С ГРАФИКОЙ И ТЕКСТОМ.

Содержащиеся в настоящей брошюре главы посвящены основам синтаксиса языка *Mathematica*. Подобно другим системам компьютерной алгебры общего назначения *Mathematica* является *в первую очередь* ЯЗЫКОМ ПРОГРАММИРОВАНИЯ СВЕРХВЫСОКОГО УРОВНЯ. Отличие этих систем, в первую очередь *Axiom*, *Mathematica* и *Maple*, от традиционных языков программирования таких, как *Fortran*, *Lisp*, *C*, *Pascal*, *Delphi* или *Java* состоит в том, что они содержат в *десять* раз больше командных слов и

---

<sup>2</sup>Словарь дает следующие переводы компьютерного термина **high-end**: мощный, профессиональный, высококачественный; высокого класса; с широкими функциональными возможностями. Поскольку ни один из этих переводов не отражает всего пафоса и всех коннотаций оригинала, мы оставляем этот термин **as is**.

конструкций и по своей гибкости, выразительной силе и расширяемости приближаются к живым языкам. Поэтому изучать их надо так же, как учат иностранный язык, скажем, французский или греческий. Однако все обычные введения описывают их не как языки, даже не как языки программирования, а как обычные приложения, такие же как Word или Excel.

## § 1. ОБЗОР ЛИТЕРАТУРЫ ПО КОМПЬЮТЕРНОЙ АЛГЕБРЕ

На чистом, без всяких помарок листе бумаги можно писать самые новые, самые красивые иероглифы, можно создавать самые новые, самые красивые рисунки.

Председатель Мао Цзе-дун “Об одном кооперативе” (15 апреля 1958 года)

Diffidate dei libri che trattano di quest'arte: sono la maggior parte fallaci o incomprensibili, specialmente quelli italiani. — Не доверяйте ни одной книге, посвященной этому искусству. Большая часть из них либо совершенно беграмотны, либо абсолютно невразумительны, особенно в русских переводах.

Pellegrino Artusi. La cucina per gli stomachi deboli<sup>3</sup>

Bienaventurados los que no saben leer ni escribir porque serán llamados analfabetos. — Блаженны те, кто не умеют ни читать, ни писать, потому что их назовут неграмотными.

J.B.La Cabeza a Pàjaros

Подавляющее большинство выходящей последние годы на русском языке компьютерной литературы относится к жанру ‘Blablabla для полного идиота’. Все магазины завалены **тысячами** руководств под названиями:

- ‘Microsoft Windows для полного идиота’,
- ‘Microsoft Word для полного идиота’,
- ‘Mathcad для полного идиота’,

etc., etc., etc., etc., etc., etc. Эти замечательные *продукты* начинаются с объяснения того, что при нажатии **кнопки** (рисунок) **компьютер** (цветной рисунок) *как правило не включается* (цветной рисунок), если предварительно не **воткнуть** (рисунок и анимация на прилагаемом диске) **вилку** (2 рисунка, профиль и фас) в **розетку** (рисунок в 3/4). После этого во всех деталях объясняется, как с помощью **мышки** (4 рисунка, 3 проекции и общий вид) **кликнуть** (5 рисунков и 3 анимации) по **иконке** (3 рисунка) и вызвать **программу** (17 рисунков), а потом **шуршать** (13 рисунков и 8 анимаций) мышкой (см. выше) по **меню** (237 цветных рисунков). Несомненно, авторы создавали эти бессмертные *творения* с полной духовной самоотдачей и невиданным напряжением всех творческих сил.

---

<sup>3</sup>Кухня для дебильных желудков — знаменитая поваренная книга, впервые напечатанная во Флоренции в 1891 году.

Полную противоположность этому составляет серьезная литература. На русском имеется несколько совершенно замечательных книг, посвященных алгебраическим алгоритмам, в том числе<sup>4,5,6,7,8,9,10,11,12,13</sup> и несколько других. Однако эти книги посвящены собственно алгоритмике и **компьютерной алгебре** как таковой, с ударением на слове **алгебра**. Иными словами, в них обсуждаются общие вопросы построения и анализа алгоритмов, вне зависимости от используемой платформы и конкретных систем и т.д. При этом для иллюстрации алгоритмов используются искусственные языки, специально придуманные для целей преподавания, такие как MIX или Pascal, а вовсе не имеющиеся коммерческие или открыто распространяемые системы научных вычислений. Эти книги адресованы либо профессионалу в области алгоритмики, алгебры, логики, сложности вычислений или теоретического программирования, либо **серьезному** студенту, который готов потратить час или два на разбор одного параграфа. К тому же многие из них безнадежно устарели, так как ориентируются на коммерческие системы 1980 годов и не учитывают революцию в эффективности и доступности совершенную в 1990-х годах системами компьютерной алгебры нового поколения Maple, Mathematica, Axiom и MuPAD.

Сюда же относятся технические описания конкретных языков программирования и **серьезные** учебники по этим языкам, такие, скажем, как книга Дейтелей по языку C и C++, и несколько совершенно блистательных книг Вирта, Дейкстры, Страуструпа, Кернигана и других ведущих специалистов по общим вопросам программирования<sup>14, 15, 16</sup>, или профессиональные учебники конкретных систем такие, как книги Кнута по TeX'у, Metafont'у, компьютерной типографии<sup>17,18</sup>. Имеется несколько замечательных книг, посвященных проблемам компьютерного кодирования и сжатия информа-

---

<sup>4</sup>А.Ахо, Дж.Хонкрофт, Дж.Ульман, Построение и анализ вычислительных алгоритмов. — М., Мир, 1979.

<sup>5</sup>Д.Э.Кнут, Искусство программирования, т.1–3. — Вильямс, М.—СПб–Киев, 2000.

<sup>6</sup>Д.Кокс, Дж.Литтл, Д.О'Ши, Идеалы, многообразия и алгоритмы. — Москва, Мир, 2000.

<sup>7</sup>П.Ноден, К.Китте, Алгебраическая алгоритмика. — М., Мир, 1999.

<sup>8</sup>Компьютерная алгебра, Символьные и алгебраические вычисления. — М., Мир, 1986.

<sup>9</sup>Р.Лидл, Г.Пильц, Прикладная абстрактная алгебра. — Екатеринбург, Изд. Уральского ун-та, 1996.

<sup>10</sup>Р.Грегори, Е.Кришнамурти, Безошибочные вычисления: методы и приложения. — М., Мир, 1988.

<sup>11</sup>Дж.Дэвенпорт, Интегрирование алгебраических функций. — М., Мир, 1985.

<sup>12</sup>Дж.Дэвенпорт, И.Сирэ, Э.Турнье, Компьютерная алгебра: системы и алгоритмы компьютерных вычислений. — М., Мир, 1991.

<sup>13</sup>А.Акритас Основы компьютерной алгебры с приложениями. — М., Мир, 1991.

<sup>14</sup>У.Дал, Э.Дейкстра, К.Хоор, Структурное программирование. — М., Мир, 1975.

<sup>15</sup>Н.Вирт, Алгоритмы и структуры данных. — СПб, Невский Диалект, 2001.

<sup>16</sup>Б.Керниган, Р.Пайк, Практика программирования. — Вильямс, М.—СПб–Киев, 2004.

<sup>17</sup>Д.Кнут, Все про TeX. — М., Протвино, 1993.

<sup>18</sup>Д.Кнут, Компьютерная типография. — М., Мир–Аст, 2003.



ции, быстрому преобразованию Фурье и другим подобным вопросам. Все это очень полезные книги, однако все они адресованы достаточно квалифицированному читателю и вряд ли могут быть с пользой прочитаны тем, кто уже не пользовался в течение нескольких лет описываемыми там языками программирования, математическими пакетами или системами компьютерного набора.

Имеется еще один очень многочисленный тип произведений, обильно представленный бессмертными творениями В.Дьяконова. Эти книги *pretendуют* на некоторую серьезность, но в действительности в их основе лежит **автоматический перевод встроенной помощи**, с сохранением всех приведенных там примеров (или их *творческой* модификацией, состоящей в замене  $a$  на  $b$  и  $x$  на  $y$ ). Иногда этот перевод *слегка* редактируется, но в подавляющем большинстве случаев без всякого понимания. Например, на стр. 172 книги В.Дьяконова *Mathematica 4* (Питер, 2001, с.1–654) говорится следующее: “ExtendedGCD[n,m] — возвращает расширенный наибольший общий делитель целых чисел  $n$  и  $m$ ”. Кто знает, что такое “расширенный наибольший общий делитель”, поднимите руку! — ALL SNAKES WHO WISH TO REMAIN IN IRELAND, PLEASE RAISE YOUR RIGHT HAND (Saint Patrick). Правильно, потому что такого термина ни по-русски, ни по-английски не существует. Что существует, это **extended Euclid algorithm**, который вместе с наибольшим общим делителем ищет его **линейное представление**<sup>19</sup>. Это значит, что либо — скорее всего!! — у автора не возникло желания проверить, как в действительности работает функция ExtendedGCD, либо проделав это, он не смог понять, что получилось, и решил оставить все *as is!!!* Ясно, что для того, кто владеет английским, подобные произведения абсолютно **бесполезны**, так как не содержат ничего нового по сравнению со встроенным Help. Для того, кто не владеет английским, эти произведения **опасны**, так как могут его лишь дезориентировать, накладывая на его собственное непонимание дополнительное непонимание, щедро приносимое автором от себя!!!

Чего однако в нашей литературе очень мало, так это книг на *промежуточном* уровне, адресованных читателю, который, **не** является полным идиотом, но не хочет и/или не готов тратить месяц напряженной работы на изучение нового языка или новой системы. У нас пока не возникла культура написания книг в духе учебников для американских колледжей, которые с одной стороны, говорят достаточно серьезные и *полезные* вещи, а с другой стороны, говорят их на простом и понятном языке, в четко структурированном виде, с большим количеством повторений и *несложных* примеров, чем искусно *мотивируют* студента их понять. Понятно, что возникновение такого рода учебников в американской литературе дело в значительной степени вынужденное, и связано с **катастрофически** низким уровнем преподавания **всех** дисциплин, и *в особенности* математики, в американских школах. Как замечает В.И.Арнольд, именно эти учебники раскрывают **главную** тайну американского высшего образования: “как лучшим американским университетам удастся за четыре года превращать малограмотных школьников в прекрасных профессионалов”.

<sup>19</sup>Для дальнейшего удобства начинающего в томе I книги Кнута этот термин переведен как **обобщенный алгоритм Эвклида**, а в томе II — как **расширенный алгоритм Эвклида**, в то время как **обобщенным алгоритмом Эвклида** там называется нечто совершенно иное!!!

## § 2. Благодарности

Основное содержание этой книги основана на лекциях по курсам “Математические пакеты” и “Математика и компьютеры”, которые мы читали на экономическом факультете СПбГУ.

Во время работы над этой книгой мы пользовались поддержкой нескольких организаций. В первую очередь следует упомянуть следующие проекты, непосредственно посвященные развитию теоретических аспектов компьютерной алгебры:

- РФФИ 00-10-00441 Разработка методов символьных вычислений в группах;
- РФФИ 03-01-00349 Символьные вычисления в конечных и алгебраических группах;
- Минвуз 2003.10.3.03.Д Вычисления в алгебре, алгебраической геометрии и теории чисел;
- Минвуз 2004.10.1.03.Д Вычисления в алгебре, теории представлений и алгебраической К-теории.

Кроме того, наша работа была поддержана грантами INTAS 00-01-00441 и INTAS 03-51-3251 и несколькими другими проектами Минвуза и РФФИ.

## ГЛАВА I. ЧТО ТАКОЕ КОМПЬЮТЕРНАЯ АЛГЕБРА?

Умножая свои знания, умножаешь чью-то скорбь.

Соломон

— Ну скажи, почему этот “мерседес” реальный? — спросил Володин.

Несколько секунд Мария мучительно думал.

— Потому что он из железа сделан, — сказал он, — вот почему. А это железо можно подойти и потрогать.

— То есть ты хочешь сказать, что реальным его делает некая субстанция, из которой он состоит?

Мария задумался.

— В общем, да, — сказал он.

— Вот поэтому мы Аристотеля и рисуем. Потому что до него никакой субстанции не было, — сказал Володин.

— А что же было?

— Был главный небесный автомобиль, — сказал Володин, — по сравнению с которым твой шестисотый “мерседес” — говно полное. Этот небесный автомобиль был абсолютно совершенным. И все понятия и образы, относящиеся к автомобильности, содержались в нем одном. А так называемые реальные автомобили, которые ездили по дорогам Древней Греции, считались просто его несовершенными тенями. Как бы проекциями. Понял?

Виктор Пелевин, Чапаев и Пустота

Я также полагаю, что термин **интеллект** следует употреблять исключительно в связи с пониманием. Некоторые же теоретики искусственного интеллекта берутся утверждать, что их робот вполне может обладать **интеллектом** не испытывая при этом никакой необходимости в действительном **понимании** чего-либо. На мой взгляд словосочетание **интеллект без понимания** есть лишь результат неверного употребления терминов. Следует, впрочем, отметить, что иногда что-то вроде частичного моделирования подлинного интеллекта без какого бы то ни было реального понимания оказывается до определенной степени возможным. В самом деле, не так уж редко встречаются *человеческие* существа, способные на некоторое время одурачить нас демонстрацией некоторого понимания, хотя, как в конце концов выясняется, оно им в принципе не свойственно!

Роджер Пенроуз, Тени разума. Гл.1. Сознание и вычисление

Предположим, что Вы получаете анонимное письмо, содержащее лишь листок бумаги с четырьмя строчками поэтического текста. Например, это может быть стихотворение Уильяма Блейка, “Eternity”:

He who binds to himself a joy  
Does the winged life destroy;

But he who kisses the joy as it flies  
Lives in eternity's sun rise.

Предположим, что кому-то удалось воссоздать всю последовательность возбуждений нейронных цепей в мозгу поэта (в тот момент, когда он написал эти строки), в Вашем мозгу (в тот момент, когда Вы прочли эти строки) и даже в мозгу отправителя этого письма! Предположим, повторяю, что кому-то удалось понять всю эту фантастическую и прекрасную сложность биохимических реакций, обеспечивающих соответствующие возбуждения в нейронных цепях, и описать (а значит постичь) всю совокупность физических и химических актов, из которых состоят эти биохимические реакции! Какое невероятное и восхитительное **знание** можно получить при таком анализе! Какое количество Нобелевских премий можно было бы получить сразу! Но ... давайте также поймем, что все это знание не дает *ничего* для понимания смысла стихов Блейка, по крайней мере в том значении, который мы обычно вкладываем в слово **понимание**, обращаясь к поэзии. Понимание поэзии Блейка следует искать на уровне того языка, на котором они были созданы, и с учетом психологии автора и читателя. Картина возбуждения нейронных сетей никак не может помочь при этом, т.е. не дает **понимания** знания.

Роальд Хоффманн. The same and not the same

В настоящей главе мы коротко обсудим возможности компьютерной алгебры и ее место в преподавании, научной и практической работе. После этого мы приведем общую информацию о системах компьютерной алгебры общего назначения и других крупных математических пакетах, показывающую место системы *Mathematica* среди других подобных систем. Компьютерная алгебра является одной из самых мифологизированных областей. Не только большинство пользователей, но и многие профессиональные математики и программисты не имеют представления о реальной силе, возможностях и специфике имеющихся систем, не говоря уже о ближайших перспективах этой области. Мы постараемся развеять некоторые из этих мифов.

Для нас не представляет сомнения, что:

- При помощи систем компьютерной алгебры уже сегодня возможно проводить все обычные в математике и ее приложениях вычисления. Все импликации этого факта не только не осознаны, но даже не начинали еще всерьез рассматриваться.
- Основные системы компьютерной алгебры общего назначения являются в первую очередь языками программирования СВЕРХВЫСОКОГО УРОВНЯ, приближающимися по своей выразительной силе к живому языку и их следует изучать именно как языки, а не как обычные компьютерные приложения.
- Математикам свойственно недооценивать то, в какой степени развитие математики зависит от внешних обстоятельств, в первую очередь от доступных вычислительных средств. РАЗВИТИЕ КОМПЬЮТЕРНОЙ АЛГЕБРЫ УЖЕ СЕГОДНЯ ОКАЗЫВАЕТ РАДИКАЛЬНОЕ ВОЗДЕЙСТВИЕ НА ИССЛЕДОВА-

ВАНΙΑ ВО МНОГИХ ОБЛАСТЯХ ЧИСТОЙ МАТЕМАТИКИ (таких, как теория групп, комбинаторика, теория чисел, коммутативная алгебра, алгебраическая геометрия и т.д.). В самое ближайшее время это влияние распространится на **всю** математику и приведет к кардинальному пересмотру основных направлений исследований, переоценке всех ценностей и полному изменению стиля работы математиков.

- Бешеное сопротивление, которое вызывает развитие компьютерной алгебры среди методистов и многих преподавателей математики, связано с тем, что **ДАЛЬНЕЙШЕЕ РАЗВИТИЕ ЭТИХ СИСТЕМ** уже в ближайшие 10–15 лет **ПРИВЕДЕТ К ПОЛНОМУ ОБЕСЦЕНИВАНИЮ ВСЕХ ТРАДИЦИОННЫХ ВЫЧИСЛИТЕЛЬНЫХ НАВЫКОВ** и необходимости полного пересмотра преподавания математики на школьном и университетском уровне.

- Бешеное сопротивление, которое вызывает развитие компьютерной алгебры среди многих представителей **Computer Science**, связано с тем, что эти системы полностью обесценивают и подавляющую часть традиционных программистских навыков. При помощи **ЭТИХ СИСТЕМ ЛЮБОЙ ГРАМОТНЫЙ ЛЮБИТЕЛЬ МОЖЕТ ЗА НЕСКОЛЬКО МИНУТ НАПИСАТЬ ПРОГРАММУ, АНАЛОГ КОТОРОЙ НА Fortran или C ПОТРЕБОВАЛ БЫ НЕСКОЛЬКИХ ДНЕЙ РАБОТЫ ПРОФЕССИОНАЛЬНОГО ПРОГРАММИСТА.**

Мы думаем, что имеется еще одно чрезвычайно существенное обстоятельство, объясняющее неистовое эмоциональное неприятие систем компьютерной алгебры и побуждающее многих игнорировать их возможности — и даже само их существование. Дело в том, что эти системы неприужденно решают задачи, которые, как традиционно считалось, являются чисто человеческими и требуют **интеллекта** и **мышления**, задачи, на которых основано все традиционное преподавание, задачи, представляющие серьезные трудности для большинства *человеческих* существ! Опыт общения с этими системами побуждает отбросить шоры европейской рационалистической философии и заново обдумать **все**, что связано с **интеллектом** и **мышлением**, полностью разделив те уровни, на которых происходит **вычисление** и те, на которых происходит **понимание**, те, на которых функционирует **интеллект** (=мышление?) и те, на которых функционирует **сознание**.

Начиная с 1950-х годов чрезвычайно популярна дискуссия на тему “может ли компьютер мыслить?” Усилия физиков были направлены на то, чтобы доказать, что компьютер *может* мыслить, в то время как аргументы лириков каждый раз основывались на таком переопределении понятия **мышления**, которое позволяло игнорировать каждый новый успех физиков. Исследования в области компьютерной алгебры вплотную подвели нас к такой точке, где никакое дальнейшее переопределение понятия **мышления** не представляется возможным и мы вынуждены *констатировать*, что компьютер может мыслить. Тем самым, подлинный вопрос искусственного интеллекта должен теперь ставиться так: “может ли компьютер *понять*, что он может *мыслить*?” — или, по Декарту, *cogito cogitare*.

## § 1. МАТЕМАТИКА И КОМПЬЮТЕРЫ

Anyone who cannot cope with mathematics is not fully human. At best he is a tolerable subhuman who has learned to wear shoes, bathe and not make messes in the house.

Lazarus Long, "Time Enough for Love"

Проблема N 1 кибернетики: Каким местом человек думает?

Проблема N 2 кибернетики: Как он это этим местом делает?

А.Соловьев, 'Искусственный интеллект'

Прежде чем переходить к обсуждению собственно системы *Mathematica*, мы сделаем несколько общих замечаний о роли компьютера и, в особенности, о роли символьных вычислений в научном исследовании и преподавании математики как на школьном, так и на университетском уровне. Нам кажется, что этот вопрос является СЕГОДНЯ не просто важной, а **центральной** проблемой для всех, кто *всерьез* задумывается над тем, ЧЕМУ И КАК нужно учить школьников и студентов технических, экономических и естественнонаучных (а может быть и гуманитарных!!!) специальностей в курсах математики и информатики.

Сегодняшнее построение курса математики в общеобразовательной школе отягощено ДВУХТЫСЯЧЕЛЕТНЕЙ традицией и, в целом, не находится более даже на уровне потребностей XVI века!!! Чуть лучше обстоит дело в нескольких специализированных физико-математических школах, но и здесь, с нашей точки зрения, необходим дальнейший *радикальный* пересмотр всей программы, в сторону ее углубления и модернизации. Курс же информатики в значительной степени унаследовал свойственное 50-м и началу 60-х годов — и совершенно абсурдное с сегодняшней точки зрения!!! — отождествление любого серьезного использования компьютера с ТРАДИЦИОННЫМ программированием.

Высказанные в предыдущем абзаце утверждения могут показаться чересчур драматичными, и нуждаются в пояснении. С нашей точки зрения, существующий сегодня школьный курс математики ориентирован, прежде всего, на выработку вычислительных навыков и *механического* использования небольшого числа стандартных алгоритмов — кстати, в большинстве своем чрезвычайно неэффективных с вычислительной точки зрения! Не следует думать, конечно, что это является чисто Российской проблемой; насколько мы можем судить, во всех странах Западной Европы — не говоря уже про США!! — преподавание математики в школе в целом *либо* поставлено еще значительно хуже, чем в России, *либо* страдает крайним формализмом (как во Франции). Не следует думать также, что это является исключительно проблемой средней школы — почти в такой же степени архаично и ни в малейшей степени не отвечает сегодняшним потребностям и преподавание математики в университетах и технических ВУЗах. Традиционные вузовские курсы — в первую очередь уже *абсолютно отстойные* курсы математического анализа, но в значительной степени также и *архаичные* курсы линейной алгебры, дифференциальных уравнений, теории

вероятностей и дискретной математики — также направлены почти исключительно на механическое овладение *рудиментарными* вычислительными навыками, без всякого понимания подлинной структуры предмета, его современного состояния и более широкого контекста.

В прошлом подобные вычислительные навыки имели несомненную ценность, но сегодня необходимость массового обучения им более чем сомнительна. Многие разделы математики и вычислительные приемы, которые изучаются сегодня в школе, по своей функциональности в современном мире подобны добыванию огня при помощи трения. Мы не говорим, что это полностью лишает их ценности, вызывает сомнение лишь необходимость систематического упражнения в их применении. Мы провели бы границу следующим образом: появление калькуляторов не отменяет необходимость в заучивании таблицы умножения. Однако появление калькуляторов делает *абсолютно* бессмысленным СИСТЕМАТИЧЕСКОЕ УПРАЖНЕНИЕ в операциях над многозначными числами — никому из сегодняшних школьников в нормальных условиях не придется выполнять подобные операции вручную, просто потому, что любое, самое примитивное вычислительное устройство делает это БЫСТРЕЕ, ЭФФЕКТИВНЕЕ И НАДЕЖНЕЕ.

**Комментарий.** И сегодня в Японии придается чрезвычайно большое значение развитию у детей МЛАДШЕГО ВОЗРАСТА навыков устного счета, вплоть до заучивания наизусть таблицы умножения  $100 \times 100$ , что в сочетании с правильным алгоритмом умножения многозначных чисел (разбиение на блоки и замена умножений сложениями и сдвигами) позволяет им легко умножать в уме восьмизначные числа. Однако никто из японских эдукационистов не говорит, что это делается с какой-то практической целью. Единственная цель этих упражнений — тренировка памяти и лучшее кровоснабжение мозга. Американские эдукационисты придерживаются противоположной теории, в большинстве американских школ не учат даже таблицу умножения  $10 \times 10$ , а для улучшения мозгового кровообращения используются уроки физкультуры. Из личного опыта мы знаем, что рядовой американский студент не в состоянии перемножить без калькулятора 7 на 8. Однако нам трудно решить для себя, как следует относиться к этому факту.

## § 2. КОМПЬЮТЕРНАЯ АЛГЕБРА

Одной из первых областей применения компьютерной алгебры была небесная механика. Классическим примером, упоминаемым во всех обзорах, служит пересчет Депритом, Хенрардом и Ромом результатов Делоне. Введение новой техники позволило им пересчитать гамильтониан в теории Луны. Делоне потратил 20 лет, выполняя вычисления вручную. Деприту, Хенрарду и Рому понадобилось всего лишь 20 часов работы небольшой ЭВМ. Следует отдать должное аккуратности Делоне, в работах которого, опубликованных в 1867 году и содержащих результаты вплоть до 9-го порядка по малым параметрам, все было вычислено безошибочно, за исключением одного сложения в 7-м порядке. Это вычисление лунной орбиты требует тщательного рассмотрения многих физических эффектов, таких как несферичность Земли, наклон эклиптики и влияние Солнца. Гамильтониан, описывающий рассматриваемую систему, занимает несколько страниц; к каждому члену нужно применять до 600 преобразований.

Я.А.ван Хюльзен, Ж.Калме<sup>20</sup>

Появление современных систем **символьных вычислений** или, как часто говорят, **компьютерной алгебры**, сокращенно **СА** (Computer Algebra), ставит — с еще большей остротой — тот же самый вопрос в применении к большинству разделов школьного и вузовского курса математики. Название **компьютерная алгебра** хорошо отражает суть дела, но не слишком удачно с точки зрения маркетинга и рекламы. Оно создает у несведущего человека впечатление, что речь идет исключительно о проведении *алгебраических* вычислений на компьютере. На самом деле оно указывает лишь на то, что большинство используемых в этих системах *алгоритмов* основано на использовании методов современной алгебры и теории чисел, предметная же область этих систем гораздо шире, при помощи них можно анализировать ВСЕ В ОБЛАСТИ НАШЕГО ОПЫТА И УМОЗРЕНИЯ, ЧТО ПОДДАЕТСЯ ТОЧНОМУ ОПРЕДЕЛЕНИЮ. Ядром большинства современных систем компьютерной алгебры действительно являются следующие три блока вычислений:

- численные вычисления НЕОГРАНИЧЕННОЙ ТОЧНОСТИ (так называемые *безошибочные вычисления*) с целыми, рациональными, вещественными и комплексными числами;
- собственно алгебраические (алиас *символьные*) вычисления с многочленами, перестановками, векторами, матрицами etc.;
- логические и структурные манипуляции с высказываниями, последовательностями, списками, множествами etc.

Однако в действительности *кроме этого* при помощи систем компьютерной алгебры можно проводить **все** обычные в математике и ее приложениях аналитические вычисления:

- численное и символьное дифференцирование, интегрирование, решение дифференциальных уравнений и уравнений в частных производных и тому подобное;
- доказательство несложных теорем, включая доказательство **всех** теорем из школьного курса геометрии;
- логическую обработку и преобразование текстов ЛЮБОЙ ПРИРОДЫ: текстов на естественных языках, шифров, музыкальных текстов, etc.;
- анализ и редактирование изображений, все геометрические и графические построения — вплоть до создания мультфильмов;
- статистическую обработку численных, текстовых, логических и графических данных;

<sup>20</sup>Я.А.ван Хюльзен, Ж.Калме, Применения компьютерной алгебры. — В кн. Компьютерная алгебра, М., Мир, 1986, с.308–325. Речь в этом отрывке идет о работе A.Deprit, J.Henrard, A.Rom, Lunar ephemeris: Delaunay's theory revisited. — Science, 1970, vol.168, p.1569–1570. Сегодня это вычисление может быть повторено на Maple или Mathematica за несколько минут.



- создание баз данных, электронных энциклопедий, интерактивных справочников, учебников и задачников по любой области знания;
- математическое моделирование любых процессов, компьютерный эксперимент;
- разработку, тестирование и анализ алгоритмов, компьютерных программ и прикладных пакетов;
- и многое другое.

Показательно в этой связи, что в США и Западной Европе такие системы компьютерной алгебры общего назначения как *Maple* или *Mathematica* имеют даже большее распространение среди физиков, химиков, биологов, инженеров, экономистов и других представителей естественных и математических наук, чем среди собственно математиков. Кстати, профессиональные математики часто предпочитают высоко эффективные сугубо специализированные системы типа *Cayley*, *GAP*, *MAGMA*, *Singular*, *Lie*, *Chevie*, *CoCoA*, *Fermat*, *PARI*, *DELiA*, *GRep*, *Magnus*, *Macaulay*, *Schur*, *FELIX* и другие, направленные собственно на алгебраические или теоретико-числовые вычисления, численные или матричные вычисления неограниченной точности, решение дифференциальных уравнений и все такое, без всякой графики, меню, палитр, мультфильмов и прочих глупостей.

Появление систем компьютерной алгебры должно оказать и громадное влияние на преподавание курса информатики. Уже давно стало ясно, что для подавляющего большинства пользователей **компьютерная грамотность** состоит отнюдь не в навыках программирования, а в умении эффективно ориентироваться в существующих системах математического обеспечения и квалифицировано их использовать. Лучшие же системы символьных вычислений вообще *полностью* упраздняют необходимость в традиционном программировании в стиле **ПошелНа**: *If ... Then ...* и *GoTo ...*. Дело в том, что они являются не компилирующими, а интерпретирующими и сами превращают определение объекта, написанное на языке, по сути достаточно близком к реальному математическому английскому языку, но с более жесткими правилами синтаксиса (расстановка скобок, знаков препинания, прописные и строчные буквы и пр.), в программу для его вычисления.

### § 3. КОМПЬЮТЕРНАЯ АЛГЕБРА В ПРЕПОДАВАНИИ

Итак, в школах необходимо преподавать: астрологию — алхимию — метафизику — теософию — порнографию — демонологию и основы гомосексуализма. Остальное упразднить.

Венедикт Ерофеев, Из записных книжек

*The human is just a creature*<sup>21</sup> for doing slower (and unreliably) (a small part of) what we already know (or soon will know) to do faster. All pretensions of human superiority should be withdrawn if humans want to survive in the future.

<sup>21</sup>Иронический парафраз сакраментального “The computer is just an instrument...”

Shalosh B. Ekhad

С нашей точки зрения все это должно в *самые ближайшие годы* привести к **полной** перестройке школьных и университетских курсов математики и информатики в следующем направлении. Прежде всего, центр тяжести курса математики должен быть перенесен с отработки стандартных вычислительных приемов — к тому же в большинстве своем полностью устаревших, неадекватных и абсолютно неэффективных для больших чисел, многочленов больших степеней и т.д. — на развитие МАТЕМАТИЧЕСКОГО МЫШЛЕНИЯ и ФОРМИРОВАНИЕ ОБЩИХ ПОНЯТИЙ. Точно так же, курс информатики должен быть направлен не на развитие конкретных программистских навыков, а на развитие АЛГОРИТМИЧЕСКОГО МЫШЛЕНИЯ. Ценными и важными для большинства учащихся навыками являются отнюдь не умение провести сложное вычисление вручную и, тем более, не точное знание глубоких математических принципов, которые лежат в основе конкретных алгоритмов, используемых в каком либо вычислении, а понимание того, что такое ТОЧНОЕ ОПРЕДЕЛЕНИЕ ОБЪЕКТА В ОБЛАСТИ МАТЕМАТИКИ, ЕСТЕСТВЕННЫХ ИЛИ ГУМАНИТАРНЫХ НАУК, какие вопросы об этих объектах можно задать и какого рода ответы на них можно ожидать, что нужно и чего не нужно вычислять, сколько примерно времени (скажем, несколько секунд или несколько миллиардов лет) такое вычисление может занять.

Например, даже для алгебраических функций вычисление неопределенного интеграла представляет собой совершенно нетривиальное занятие, использующее *чрезвычайно* глубокую математику, развитую только в начале XIX века. Повторим, что в школьный курс математики до сих пор не включены даже достижения математики XVI века. О том, чтобы ПРИ СОХРАНЕНИИ СЕГОДНЯШНИХ ПОДХОДОВ включить туда серьезную профессиональную математику XIX века, не может быть и речи!! Даже применение простейших приемов, таких, как интегрирование по частям или замена переменных, в конкретных ситуациях представляет собой творческий процесс и основано на *угадывании* ответа — впрочем, системы символьных вычислений часто обходятся без угадывания, проделывая *тысячи* таких подстановок в секунду — или, скорее всего, вообще применяя алгоритмы совершенно другого типа!!! Возникает естественный вопрос, должны ли мы учить не то что школьников, а, скажем, инженеров технике подобных вычислений и приемам угадывания ответа? Но они должны хорошо понимать, что такое интеграл, как его можно использовать, когда его естественно вычислять — с самим же вычислением обычно *значительно* лучше справится машина.

**Комментарий.** Конечно, весь комплекс проблем, связанных с внедрением компьютеров в учебный процесс, еще значительно шире. Кстати, современные текстовые редакторы с автоматическим выделением и исправлением орфографических, синтаксических и стилистических ошибок, соединенные с электронными словарями, грамматиками и тезаурусами, резко снижают и практическую ценность грамотного письма как другой традиционной основы школьного образования — уже лет 10 назад Японские профессора жаловались нам, что молодежь совершенно разучилась писать иероглифы, потому что для их набора при помощи широко распространенных и весьма дешевых `wordprocessor`'ов больше не

требуется умения воспроизводить иероглифы, знания традиционной последовательности черт и т.д., а достаточно лишь умения узнавать их среди других иероглифов с тем же чтением.

Трудности внедрения компьютерной алгебры в учебный процесс носят как психологический, так и объективный характер. К психологическим трудностям следует отнести в первую очередь уже упоминавшуюся громадную инерционность в преподавании традиционных общеобразовательных дисциплин и неподготовленность значительной части преподавательского состава к серьезному использованию компьютерных средств в обучении. Для многих преподавателей существенным моментом является неготовность к новым критериям оценки математических знаний, не по формальной натасканности и не по способности выполнять бессмысленные формальные манипуляции, а по реальному пониманию предмета.

Объективные трудности отчасти носят экономический характер (высокая стоимость лицензированных копий современных программных систем и сопутствующей документации, высокие требования, предъявляемые ими к оборудованию и т.д.). Однако, в особенности в применении к системам компьютерной алгебры, имеются еще принципиальные трудности, в первую очередь неразработанность методических основ их использования в учебном процессе, будь то в школе или в ВУЗе, и отсутствие соответствующей документации. Нет доступной начинающим литературы, которая описывала бы не особенности работы с конкретными системами символьных вычислений, а формировала понимание математики и навыки проведения математических вычислений с использованием технических средств. Тем более нет сборников задач, которые могли бы предлагаться при таком подходе к обучению.

Дело в том, что первоначально эти системы возникли как сугубо профессиональные инструменты, использование которых было ограничено узким кругом экспертов. Даже после того как эти системы получили широкую известность и стали распространяться на коммерческой основе, сопутствующая им документация зачастую сохранила чисто технический характер и предполагает у пользователя знакомство с нетривиальными понятиями программирования, логики, комбинаторики, алгебры, теории чисел, свободное владение языком и символикой современной математики, понимание общих принципов представления и преобразования данных в компьютере и используемых при этом алгоритмов.

Ясно, что такого рода тексты (например, распространяемая вместе с лицензированной версией *Mathematica* фундаментальная книга Вольфрама) совершенно не приспособлены для использования начинающим, который должен овладевать используемыми математическими понятиями одновременно с конкретным языком программирования или вычислительной системой. К тому же, для большинства систем сколь-нибудь подробная и надежная документация доступна только на английском языке. Между тем, многие из функций и алгоритмов, реализованных в этих системах, могут быть объяснены уже на школьном уровне — и чрезвычайно полезны уже при изучении

обычной школьной математики начиная с арифметики целых чисел и действий над обыкновенными дробями!! Уже школьники младших классов могли бы получить громадную пользу перепоручив рутинные вычисления компьютеру!

#### § 4. ВЛИЯНИЕ КОМПЬЮТЕРОВ НА МАТЕМАТИЧЕСКОЕ МЫШЛЕНИЕ

Science is what we understand well enough to explain to a computer.  
Art is everything else we do.

Donald Knuth

The great advances in science usually result from new tools rather than from new doctrines.

Freeman Dyson

Mathematics is much less formally complete and precise than computer programs.

William P. Thurston

To be a scholar of mathematics you must be born with talent, insight, concentration, taste, luck, drive and the ability to visualize and guess.  
— Чтобы стать профессиональным математиком, нужно родиться с талантом, проницательностью, сосредоточенностью, стилем, удачей, настойчивостью, внутренним зрением и воображением.

Paul R. Halmos

С нашей точки зрения абсолютно не поняты теоретические основы использования систем компьютерной алгебры в преподавании и исследованиях за пределами математики и теоретической физики, а также те изменения, которые эти системы предполагают в собственно математическом мышлении пользователя. Даже многие авторы учебников по *компьютерной математике*, не говоря уже о широких кругах пользователей-неспециалистов не отличают настоящие системы компьютерной алгебры, такие как Maple или Mathematica, ни от специализированных пакетов *численных* вычислений, таких как MatLab или Statistica, ни даже от чисто учебных программ, графических калькуляторов или систем для работы с текстом, содержащим формулы, таких как Mathcad.

В большинстве случаев принятое сегодня изложение математики — не только в школе, но и на математических факультетах университетов — возникло в докомпьютерную эпоху и совершенно неудовлетворительно с алгоритмической точки зрения. Между тем, во многих случаях даже небольшое изменение определений делает их значительно более пригодными для практических вычислений. Скажем, не только в школьном, но и в университетском курсе, степень определяется рекурсивно как  $x^n = x^{n-1}x$ . Между тем, самое незначительное изменение этого определения может драматически увеличить его применимость. Например, можно определить степень указанной выше формулой в случае, когда  $n$  нечетно и формулой  $x^n = (x^{n/2})^2$  в

случае, когда  $n$  четно. Для объектов, умножение которых занимает заметное время (матрицы или многочлены высокой степени), вычисление, скажем, 1000-й степени с использованием второго определения может быть в сотни раз быстрее, чем с помощью первого (несколько минут versus нескольких часов). Замечу, что речь не идет о наиболее эффективном профессиональном алгоритме для вычисления степени — в большинстве случаев достаточно просто задумываться над подобного рода нюансами.

Еще более интересные феномены связаны с умножением матриц. Обычный алгоритм умножения матриц размера  $2 \times 2$  требует 8 умножений коэффициентов. Ф.Штрассен предложил алгоритм требующий лишь 7 умножений. Для больших степеней этот алгоритм дает еще более драматическую редукцию числа умножений. Примерно так же можно упростить и гауссово исключение. Важность этого открытия трудно переоценить — некоторые специалисты считают его одним из 10 наиболее важных математических открытий XX века. При решении серьезных вычислительных задач (например при приближенном решении дифференциальных уравнений движения, в задачах оптимизации), где приходится умножать и обращать тысячи матриц порядка 1000 и более, этот алгоритм дает громадную экономию времени, делающую возможными немислимые ранее вычисления (НАСА использует этот алгоритм при управлении в реальном времени космическими аппаратами).

Другой столь же впечатляющий пример, связанный с умножением матриц, где четко видна граница между возможным и невозможным — вычисления в конечных группах. Одним из самых замечательных достижений математики XX столетия явилась классификация конечных простых групп. Как выяснилось, кроме нескольких бесконечных серий таких групп имеется еще ровно 26 “маленьких исключений”, так называемых спорадических групп. Самая большая из этих спорадических групп, называемая “Большим Монстром” или “Дружественным Гигантом”, имеет порядок

$$2^{46} \cdot 3^{20} \cdot 5^9 \cdot 7^6 \cdot 11^2 \cdot 13^3 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 41 \cdot 47 \cdot 59 \cdot 71$$

и наименьшая степень ее точного матричного представления равна 196883. Умножение двух матриц такого порядка на лучших современных *workstations* занимает около года машинного времени — так что время, необходимое для вычислений в этой группе традиционными методами, на много порядков превосходит возраст Вселенной (кстати, уже просто хранение большого числа таких матриц и результатов промежуточных вычислений в памяти машины представляет собой достаточно серьезную проблему). В то же время умножение столбца высоты 196883 на матрицу размера  $196883 \times 196883$  требует всего нескольких минут машинного времени и вполне осуществимо, так что в последние годы многие специалисты занимаются разработкой алгоритмов “матричных вычислений без матриц”. Для матриц с полиномиальными коэффициентами подобные эффекты наступают уже значительно раньше, скажем на очень важном с точки зрения приложений в физике случае матриц размера  $248 \times 248$ .

В компьютерной алгебре значительно (бесконечно?) больше математики, чем в традиционном программировании и значительно больше программирования, чем в традиционной математике. В действительности даже для профессионального математика овладение основными принципами компьютерной алгебры на начальном этапе требует значительного интеллектуального усилия. Дело в том, что *кроме* всех обычных математических операций компьютерная алгебра включает в себя громадное количество структурных операций, связанных с манипуляцией с выражениями, и алгоритмических операций, связанных с управлением ходом вычислений (*flow of calculation*). Большинство этих структурных и алгоритмических операций не имеют стандартных названий и обозначений в традиционной математике. Более того, многие из этих новых операций и проводимых при их осуществлении различий в классической математике вообще отсутствуют, по крайней мере на сознательном уровне. Поэтому даже для профессионального математика изучение компьютерной алгебры является ценнейшим опытом, который позволяет взглянуть на уже известные ему математические явления совершенно другими глазами: THE PURPOSE OF TRAVEL IS NOT VISITING NEW PLACES, BUT HAVING NEW EYES.

## § 5. СИСТЕМЫ КОМПЬЮТЕРНОЙ АЛГЕБРЫ: ОБЩАЯ КЛАССИФИКАЦИЯ

Историческим достижением Сталина было то, что он предельно упростил картину мира и сделал ее доступной для секретарей обкомов.

Леонид Шебаршин. Заметки бывшего начальника разведки

Начиная с 1960-х годов было создано *несколько сотен систем компьютерной алгебры* (computer algebra systems, на профессиональном жаргоне CAS). Все эти системы характеризуются наличием вычислительного ядра, проводящего символьные вычисления различных типов. Некоторые из них, кроме того, включают многочисленные справочные библиотеки, пакеты расширений, более или менее развитый интерфейс, генерацию графики, форматирование ввода-вывода и т.д.

**1. Системы общего и специального назначения.** По своему назначению, структуре и функциям системы компьютерной алгебры делятся на два основных типа:

- **Системы общего назначения (general purpose)**, называемые также **универсальными CAS**, производят все обычные математические вычисления, которые обычно проводят нематематики, некоторые более специальные вычисления, а также умеют обрабатывать графику, текст и звук. Системы общего назначения обычно сопровождаются доступной документацией, имеют наглядный интерфейс и поддерживают различные форматы ввода и вывода. Эти системы перенесены на все основные платформы и операционные системы, общение с ними происходит обычно в режиме диалога.

• **Системы специального назначения** (*special purpose*) умеют проводить только определенный тип вычислений, либо с точки зрения используемой математики (скажем полиномиальные вычисления, матричные вычисления, вычисления в группах, вычисления в коммутативных кольцах, теоретико-числовые вычисления, дифференциальные уравнения и т.д.), либо с точки зрения предметной области (квантовая химия, небесная механика, астрономия, теория гравитации, физика высоких энергий, теория элементарных частиц и т.д.). Эти системы обычно работают под UNIX в текстовом режиме.

Специализированные системы часто гораздо быстрее и эффективнее систем общего назначения, так как они изначально оптимизированы под решение достаточно ограниченного круга задач и лишены *general nonsense overhead*, связанного с необходимостью интерпретировать разнообразный ввод, компилировать тысячи функций ядра в момент загрузки, поддерживать интерфейс и графику, иметь встроенный *Help* и тому подобное. В то же время использование таких систем обычно требует значительно более серьезной подготовки, мощных профессиональных компьютеров и оправдано *только* если Вы занимаетесь исследовательской работой в некоторых областях математики или теоретической физики, критическим образом зависящих от вычислений, находящихся на пределе сегодняшних возможностей.

Однако, насколько мы можем судить по публикациям, в исследовательской работе даже профессиональные математики обычно предпочитают пользоваться системами компьютерной алгебры общего назначения не только для преподавания и на этапе компьютерного эксперимента, но и для большей части собственно научных вычислений, обращаясь к специализированными системами *только* в тех случаях, когда это действительно необходимо. Для *computer aided research* в высшей степени типична ситуация, когда в одной и той же статье основная часть вычислений проведена с помощью *Mathematica*, а теоретико-групповые вычисления в группах громадных порядков — с помощью *GAP*; или, когда для основных вычислений используется *Maple*, а для алгебро-геометрических вычислений — *Singular*.

**Резюме.** КАК ВСЕ ВЫЧИСЛЕНИЯ, ПРЕДСТАВЛЕННЫЕ В МАТЕМАТИЧЕСКИХ КУРСАХ ШКОЛЬНОГО И УНИВЕРСИТЕТСКОГО УРОВНЯ, ТАК И ПОДАВЛЯЮЩЕЕ БОЛЬШИНСТВО ВЫЧИСЛЕНИЙ, КОТОРЫЕ МОГУТ ВСТРЕТИТЬСЯ ИНЖЕНЕРАМ, ЭКОНОМИСТАМ И ЛЮБЫМ ДРУГИМ ПРОФЕССИОНАЛАМ, ГОРАЗДО УДОБНЕЕ ПРОВОДИТЬ ПРИ ПОМОЩИ СИСТЕМ ОБЩЕГО НАЗНАЧЕНИЯ.

**2. Системы общего назначения.** Первые работы, приведшие к появлению компьютерной алгебры, были выполнены в самом начале 1950-х годов, но только к середине 1960-х годов они привели к созданию первых работоспособных универсальных систем компьютерной алгебры. На русском языке ранняя история таких систем детально описана в статье<sup>22</sup>. Мы не

<sup>22</sup>Я.А.ван Хюльзен, Ж.Калме, Системы компьютерной алгебры. — В кн. Компью-

будем пытаться пересказывать эту историю или хотя бы просто перечислять наиболее важные из этих систем, так как сегодня большинство из них попадает *по крайней мере* в одну из следующих категорий:

- системы, которые полностью устарели, не поддерживаются разработчиками, и, таким образом, представляют чисто исторический интерес (скажем, Macsyma, MATHLAB, Mu-Math или Reduce);
- системы, которые являются чисто учебными и имеют весьма ограниченные возможности (скажем, MathView);
- системы, которые являются экспериментальными и интересны главным образом тем, кто сам разрабатывает алгоритмы алгебраических вычислений (скажем, MAS и Risa/Asir);
- системы, которые занимают промежуточное положение между системами общего и специального назначения и не предназначены для бытового использования (скажем, JACAL или FORM).

**Резюме.** ОТБРАСЫВАНИЕ СИСТЕМ ЭТИХ КАТЕГОРИЙ ОСТАВЛЯЕТ ВСЕГО **пять** ЖИВЫХ<sup>23</sup>, РАБОТОСПОСОБНЫХ И ДОСТУПНЫХ СИСТЕМ ОБЩЕГО НАЗНАЧЕНИЯ, А ИМЕННО Axiom, Derive, Maple, Mathematica, и MuPAD.

В следующем параграфе мы обсудим эти пять систем и придем к выводу, что в действительности в качестве кандидатов для использования в учебном процессе могут рассматриваться только **две** из них, Mathematica и Maple.

---

терная алгебра, М., Мир, 1986, с.277–307.

<sup>23</sup>Существуют различные мнения по поводу того, следует ли в настоящее время (2005 год) рассматривать Axiom как живую систему. Авторы считают, что до тех пор, пока ее возможности не будут превзойдены другими системами, ответ на этот вопрос безусловно положителен.



ОПЕРАЦИОННЫЕ СИСТЕМЫ, ПОДДЕРЖИВАЕМЫЕ  
ОСНОВНЫМИ СИСТЕМАМИ СА ОБЩЕГО НАЗНАЧЕНИЯ И  
ДРУГИМИ КРУПНЫМИ МАТЕМАТИЧЕСКИМИ ПАКЕТАМИ<sup>24</sup>

	UNIX	DOS	Win	MacOS	NeXT	OS/2	VMS
Axiom	+		+				
Derive		+	+				
FORM	+	+	+	+	+		+
JACAL	+	+		+		+	+
Macsyma	+		+				
Maple	+		+	+			
MAS	+					+	+
Mathcad			+	+			
MatLab	+		+	+			+
Mathematica	+		+	+	+	+	+
MuPAD	+		+	+			
Reduce			+	+			
Risa/Asir	+	+		+			
Scilab	+		+				

**3. Другие математические программы.** Кроме того, имеется громадное количество **математических пакетов**, не являющихся системами компьютерной алгебры. Подавляющее большинство из них крайне примитивны и по существу являются либо большими научными калькуляторами, либо интерактивными учебниками или справочниками, либо редакторами формул (**formula editor**). Они не имеют собственного ядра символьных вычислений, а ищут подходящие случаю формулы в таблицах, либо, в исключительных случаях, умеют подставить в формулу другую букву. На вершине такого рода *изделий* находится Mathcad (в девичестве MathCAD). С другой стороны, имеется большое количество очень полезных специализированных пакетов (фактически, библиотек программ), проводящих численные вычисления в различных предметных областях, например, статистическую обработку данных, расчет механизмов и электрических цепей

<sup>24</sup>Указанные данные относятся к 2003–2004 годам, либо к последним официальным версиям. В этой таблице мы не пытались отразить историю. Скажем, созданная в 1965 году система **Reduce** представляет собой одну из самых старых систем компьютерной алгебры и в разные периоды своей долгой жизни заигрывала со всеми основными операционными системами, начиная с Y-MP и кончая чуть ли не **Atari**.

и т.д. Наиболее известными и продвинутыми продуктами такого рода являются **MatLab** и **Scilab**.

Специфика **MatLab**, **Scilab** и **Mathcad** по сравнению с большинством других пакетов такого рода определяется тем, что в их интерфейс встроено обращение к ядру **Maple** или каким-то его частям. Это значит, что находясь внутри этих систем можно выполнять символьные вычисления в **Maple** — причем в **MatLab Extended Symbolic Math Toolbox** и в **Scilab** все такие вычисления! Поэтому<sup>25</sup> многие — в том числе и некоторые авторы компьютерной литературы!!! — ошибочно считают, что сами эти пакеты являются системами компьютерной алгебры.

## § 6. СИСТЕМЫ ОБЩЕГО НАЗНАЧЕНИЯ: Axiom, Derive, Maple, Mathematica, MuPAD

В нужное время в нужном месте.

Девиз фирмы **Tamрах**

**1. Axiom, Maple, Mathematica, MuPAD.** В настоящее время на *любительском* рынке математического обеспечения для РС имеется громадное количество учебных и специализированных программ, и **всего четыре** универсальных (*general purpose*) системы, при помощи можно производить *серьезные* математические и научные вычисления:

- Maple 9.0,
- Mathematica 5.1,
- Axiom 2.0,
- MuPAD 2.5,

(указаны последние официальные релизы).

Системы **Maple**, **Mathematica** и **Axiom** были созданы примерно одновременно<sup>26</sup>, в 1985, 1988 и 1991 году, соответственно. К сегодняшнему дню эти системы полностью вытеснили все более ранние системы. В настоящее время все ОНИ ИМЕЮТ ПРИБЛИЗИТЕЛЬНО ОДИНАКОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ВОЗМОЖНОСТИ, но в системе **Axiom** значительно более богатая и жесткая система объектных классов, со всеми вытекающими отсюда достоинствами (которые с точки зрения неопытного пользователя представляются скорее недостатками). Все они поддерживают графику высочайшего уровня с возможностью импорта и экспорта десятках форматов.

Система **MuPAD** создана в 1997 году, учитывает все основные достижения трех предшествующих систем и в некоторых отношениях концептуально правильнее их. Тем не менее, она все еще находится **under construction**, не имеет эффективных команд генерации графики и использовать ее начинающему гораздо труднее.

<sup>25</sup> А также из-за сходства названия **MatLab** с названиями систем компьютерной алгебры **MATHLAB** и **MATHLAB-68**!

<sup>26</sup> В действительности, эти системы появились *в точности* одновременно, так как **Maple** был первоначально доступен только под **Mac**, а **Axiom** медленно вылуплялась из **Scratchpad**.

**Резюме.** Собственно вычислительные возможности этих четырех продуктов в принципе сопоставимы. Однако Maple и Mathematica проще для изучения и использования и значительно лучше приспособлены для преподавания математики.

**2. Maple и Mathematica.** В то же время выбор между системами Maple и Mathematica абсолютно нетривиален и определяется исключительно прагматическими и эстетическими соображениями. Прежде всего отметим, что Maple и Mathematica являются не просто вычислительными системами, но и — в первую очередь!!! — языками программирования **сверхвысокого** уровня, которые так же соотносятся с C++, VisualBasic, Delphi или Java, как эти последние соотносятся с языком ассемблера (или даже программированием в кодах!) Разумеется, как в том, так и в другом языке выражаются **все** конструкции, которые выражаются во всех обычных языках программирования (и много таких, которые там не выражаются!!) Если Вы писали программы на традиционных алголоподобных языках таких, как Basic, Fortran или Pascal, то Вам будет ближе язык Maple. С другой стороны, присущий системе Mathematica стиль программирования ближе к Lisp, C++ или Java но, конечно, *гораздо* гибче. Если Вы имеете опыт программирования на этих языках — или не имеете никакого!!! — то Вам будет заметно легче овладеть языком Mathematica. Многие аспекты организации языка системы Mathematica и ее интерфейса делают ее несколько более доступной, по сравнению с Maple, для человека не знакомого с алгоритмикой и программированием.

**Резюме.** Если Вы программист или имеете опыт процедурного программирования, Вам будет несколько проще овладеть языком Maple, если математик или гуманитарий — языком Mathematica.

**3. Axiom и MuPAD.** Если у Вас есть доступ к Unix'овской рабочей станции со 100 гигабайтами оперативной памяти и несколькими терабайтами на твердых дисках, то имеющийся у Вас выбор математических систем значительно богаче и Вы вряд ли нуждаетесь в наших советах. Скорее всего, Вас мало интересует *дружественный интерфейс*, палитры и другие подобные глупости и Вы пользуетесь специализированными системами. Если в этой ситуации Вам все же нужна система общего назначения, то до недавнего времени вместо Maple или Mathematica мы, скорее всего, рекомендовали бы разработанную IBM систему Axiom, которая в течение 10 лет оставалась самой мощной из всех существовавших систем общего назначения. Достоинством Axiom является четкая структура<sup>27</sup>, согласованная с образом мышления и стилем современной математики. Аксиоматически определить класс математических объектов в Axiom *существенно* проще, чем в Maple или Mathematica. Кроме того, там имплементированы мощные форсированные алгоритмы, позволяющие проводить эксперименты в области машинного доказательства теорем. Однако все в природе имеет свою цену, в данном случае чрезвычайно жесткий синтаксис и некоторые другие особенности системы затрудняют ее использование непрофессионалом.

Однако последние два года система Axiom не поддерживается разработчиками (хотя имеются планы возобновить такую поддержку и выпустить новую версию этой системы). В то же время невозможно не упомянуть еще фантастический прогресс, достигнутый за

<sup>27</sup>Структура, но не язык!!! Синтаксически и особенно лексически язык Mathematica гораздо ближе к общематематическому.

последние несколько лет разработчиками системы компьютерной алгебры **MuPAD**, построенной на тех же общих принципах, что **Axiom**, но в то же время учитывающей значительную часть достижений **Maple** и **Mathematica**. К сожалению, эта система организована в виде библиотек функций, использование которых отнюдь не является прозрачным для начинающего пользователя.

**Резюме** Если Вы профессиональный алгебраист, и Вас интересуют аксиоматические теории и машинное доказательство теорем, изучите возможности **Axiom** или **MuPAD**. Скорее всего, они лучше подойдут для Ваших целей, чем **Mathematica** или **Maple**.

**4. Derive.** Имеется чрезвычайно удачная минисистема компьютерной алгебры общего назначения **Derive 6.0**<sup>28</sup>. Первая версия **Derive** появилась в 1988 году примерно одновременно с **Mathematica**. Она была разработана специально для использования на миникомпьютерах, в том числе фирмы **Texas Instruments**. В настоящее время это единственная серьезная система компьютерной алгебры, которая может работать на DOS'овских компьютерах начала 1980-х годов, у которых нет даже твердого диска. Целесообразность ее использования на стационарных компьютерах весьма сомнительна, так как по вычислительной силе, качеству графики и другим возможностям она значительно уступает системам **Maple** и **Mathematica**. В то же время она весьма надежна, экономно обходится с вычислительным ресурсом, проста в использовании и для своего размера *необычайно* эффективна. Из всех систем общего назначения она *единственная* может работать на карманных компьютерах класса **Palm**.

## § 7. ДРУГИЕ МАТЕМАТИЧЕСКИЕ ПАКЕТЫ:

### MatLab и Mathcad

Shaw's Principle: Build a system that even a fool can use, and only a fool will want to use it.

Бессмысленно объяснять вкус дыни тому, кто всю жизнь жевал шнурки от ботинок.

Виктор Шкловский

Ну что ты, говорит, Левушка, конечно Довлатов лучше. Тут он трах ее дубиной по лбу! И с тех пор во всем полагался на ее литературное мнение.

Михаил Веллер, Ножик Сережи Довлатова

Кроме систем компьютерной алгебры имеется большое количество других специализированных программных средств для проведения вычислений или подготовки документов, содержащих математические формулы. Мы обсудим два наиболее популярных продукта этих жанров, **MatLab** и **Mathcad**. Не являясь сами системами компьютерной алгебры, они интегрированы с системой **Maple** и могут вызывать ядро **Maple** для символьных вычислений (**MatLab** в полном объеме, а **Mathcad** — несколько десятков функций).

**1. MatLab.** Весьма популярный среди инженеров, экономистов и многих других корпоративных пользователей пакет **MatLab** представляет собой не систему компьютерной алгебры в описанном выше понимании, а чрезвычайно развитую вычислительную среду, состоящую из огромной библиотеки специализированных программ, объединенных общим интерфейсом.

Изначально **MatLab** задумывался и разрабатывался как система *численных* вычислений для инженеров, система символьных вычислений (на основе **Maple**) была включена в

<sup>28</sup>B.Kutzler, V.Kokol-Voljc, Introduction to **Derive 6**. — Texas Instruments, 2003

него лишь много позже. В то же время **Maple** и **Mathematica** с самого начала представляли собой системы *символьных* вычислений, созданные профессиональными математиками для теоретических исследований в области математики и физики. Это значит, что первые версии этих систем сильно отличались между собой, причем **MatLab**, не имея выразительной мощи и гибкости **Maple** и **Mathematica**, был значительно лучше оптимизирован под стандартные вычислительные задачи (такие, как численное решение систем дифференциальных уравнений), требующие вычислений с разреженными матрицами громадных порядков. В первой половине 1990-х годов такого рода вычисления производились при помощи **MatLab** раз в 5–10 быстрее, чем при помощи его конкурентов. Однако в дальнейшем происходила конвергенция этих систем, как за счет включения в **MatLab** пакетов символьных вычислений, так и за счет оптимизации ядра систем **Maple** и **Mathematica** и включения туда быстрых алгоритмов численных и матричных вычислений. В настоящее время нет никакой статистически заметной разницы в производительности этих трех систем при проведении *численных* вычислений. Тем самым, выбор одной из трех этих систем определяется *исключительно* наличием нужных Вам функций в стандартных библиотеках, предпочтениями в области интерфейса, привычности синтаксиса и подобными субъективными факторами.

Не забывайте однако, что для проведения *символьных* вычислений в среде **MatLab** Вам все равно придется учить **Maple**. Но тогда, конечно, имеет смысл использовать **Maple** для всех вычислений и не забивать себе голову изучением **MatLab**. Чтобы облегчить выбор начинающему, отметим, что **MatLab** остается гораздо более громоздкой и тяжеловесной системой, как с точки зрения используемого ей ресурса, так и с точки зрения интерфейса. Однако все, что работает в одну сторону, часто с тем же успехом работает и в другую. Как **Maple**, так и **Mathematica** позволяют в процессе работы ядра подгружать и исполнять откомпилированный внешний код на **C**. Это значит, что после того, как Вы научились это делать, Вам нет никакого смысла переписывать те стандартные *вычислительные* функции, которые в **MatLab** реализованы хорошо. Этот принцип обильно проиллюстрирован в недавней *блистательной* книге Сэломона<sup>29</sup>. Все новые фрагменты кода там написаны в **Mathematica**, в то время как в качестве стандартных процедур оцифровки, дискретных преобразований и пр. вызываются функции **MatLab**.

**Резюме.** Если Вы не пользовались системой **MatLab** раньше, если она не используется в Вашей компании, или если в ней нет пакета, решающего нужную Вам задачу, учиться ей сегодня не имеет никакого смысла. Однако, если Вы уже умеете ей пользоваться и в ней есть нужные Вам функции, не стесняйтесь вызывать их из **Maple** или **Mathematica**.

**2. Mathcad.** Имеется еще один широко пропагандируемый продукт, **Mathcad**, который *позиционируется* как система научных вычислений. Встроенный **Help** гордо заявляет: “Mathcad is the industry standard calculation software for technical professionals, educators, and college students.” Однако с нашей точки зрения эти претензии ни на чем не основаны. **Mathcad** по своему классу является большим и красивым *научным калькулятором*, а вовсе не системой символьных вычислений. Достоинства — и недостатки!!! — **Mathcad** по сравнению с **Maple** и **Mathematica** те же самые, что достоинства и недостатки **MS-Word** по сравнению с **TeX** или фотоаппарата-мыльницы по сравнению с профессиональной камерой. В мыльнице не требуется от руки выставлять выдержку, фокус и пр., что позволяет получить *узнаваемый* результат даже ребенку. Зато при помощи мыльницы невозможно изготовить не то, что профессиональный, но и сколь-нибудь достойный и привлекательный продукт. Единственное (весьма сомнительное!) достоинство **Mathcad** состоит в интерфейсе, позволяющем *вводить* формулы в привычном школьнику виде с произвольного места в рабочем листе (*выводить* формулы в таком виде умеют все программы). Однако даже если самокат продается по цене Мерседеса и называется

- TurboPro-самоМАТНкат.v666.Ai-XXL.Billennium Edition

<sup>29</sup> Д.Сэломон, Сжатие данных, изображения и звука. — М., Техносфера, 2004.

он все равно остается самокатом, а не автомобилем. По своим реальным (а не заявленным!!!) возможностям **Mathcad 11** уступает не только версиям **Maple** и **Mathematica** 12-летней давности, но и многим крошечным программам таким, как **Derive 6.0**. Заявление о включении в **Mathcad** ядра символьных вычислений **Maple** является чисто рекламным трюком. Например, знаменитый интерфейс **Mathcad** вообще не позволяет вводить матрицы, в которых больше 100 элементов!! В то же время на самых простых бытовых компьютерах последние версии **Maple** и **Mathematica** генерируют и обращают *случайные* матрицы  $1000 \times 1000$  за 2–3 секунды, т.е. эффективно работают с массивами данных большими в десятки тысяч раз.

В действительности **Mathcad** не является ни системой компьютерной алгебры, ни даже системой научных вычислений. Это система **подготовки документов** под Windows<sup>30</sup>, содержащих математические формулы и несложную графику в режиме WYSIWYG<sup>31</sup>. К тому же она еще и немножко считает<sup>32</sup>. Однако в действительности сам по себе — **без помощи Maple** — пакет **Mathcad** считает примерно столько же, сколько любая другая система подготовки документов, скажем, **Excel** или **Scientific WorkPlace**. Кстати, в меню **Scientific WorkPlace** тоже предусмотрен прямой вызов **Maple**. В действительности, количество математических функций в **Mathcad** невелико, около 250 (в сравнении с 2000 функций, компилируемых непосредственно при загрузке ядра **Maple** или **Mathematica** и немерянными тысячами функций, определенными там в библиотеках и пакетах). Кроме того, ни математики, ни специалисты в области компьютерного набора или компьютерной графики никогда не работают в режиме WYSIWYG, так как такой режим не позволяет получать документы профессионального качества. Из-за применения неявных форматов документы, изготовленные при помощи **Mathcad**, могут быть утилизированы только под Windows и не переносятся ни на одну другую платформу (кроме, конечно, **PowerPC** от **Macintosh**, который умеет прикидываться обычным PC). В действительности, даже до последнего цеплявшаяся за принцип WYSIWYG шайка **МелкоМягких** приняла решение от него отказаться и в настоящее время разрабатывает новые движки для **Word2005** на основе **TeX**.

**Резюме.** Использовать **Mathcad** не имеет смысла ни при каких обстоятельствах, так как он неудобен, слаб, непереносим, а все положенные в его основу принципы порочны.

## § 8. СИСТЕМЫ КОМПЬЮТЕРНОЙ АЛГЕБРЫ КАК ЯЗЫКИ ПРОГРАММИРОВАНИЯ СВЕРХВЫСОКОГО УРОВНЯ

God made machine language; all the rest is the work of man.

Leopoldt Kronecker

Dislexics of the world, UNTIE!

Karl Marx

**1. Языки Axiom, Maple и Mathematica.** КАЖДАЯ ИЗ БОЛЬШИХ УНИВЕРСАЛЬНЫХ СИСТЕМ КОМПЬЮТЕРНОЙ АЛГЕБРЫ ИСПОЛЬЗУЕТ СВОЙ СОБСТВЕННЫЙ ЯЗЫК, СОСТОЯЩИЙ ИЗ НЕСКОЛЬКИХ ТЫСЯЧ СЛОВ. Имеется

<sup>30</sup>Это не только наше мнение, вот как позиционируют **Mathcad** рецензенты компьютерного журнала **Maths & Stats**, посвященного оценкам и сравнению математических пакетов. Neville Dean: “**Mathcad** is a package for creating technical documents which include mathematics and graphs”. Rob Beale: “**Mathcad** is a Windows program designed to work with formulae, text and graphics”.

<sup>31</sup>What you see is ALMOST what you get.

<sup>32</sup>“Она еще и немножко шьет.”

два случайных обстоятельства, которые мешают большинству пользователей — причем в первую очередь профессиональным программистам — осознать самостоятельное существование этих языков:

- Язык *Axiom* используется только системой *Axiom*, язык *Maple* — только системой *Maple*, а язык *Mathematica* — только системой *Mathematica*;
- Частично специфика этих языков связана с конкретными особенностями реализации ядра систем *Axiom*, *Maple* и *Mathematica*.

Тем не менее, с нашей точки зрения эти акцидентальные факторы не должны затемнять главного, а именно того, что языки *Axiom*, *Maple* и *Mathematica* существуют сами по себе и представляют **огромный** интерес независимо от соответствующих систем. Каждый, кто серьезно интересуется вычислениями, должен изучить эти языки, в первую очередь потому, что они полностью меняют представление о том, что такое программирование.

Нам представляется, что эти языки являются принципиально новым этапом в развитии программирования и соотносятся с традиционными *ALGOL*-оподобными языками так же, как эти последние соотносятся с языком ассемблера — или даже программированием в кодах. Следующие факторы побуждают считать их языками **сверхвысокого уровня** по отношению ко всем традиционным языкам программирования:

- По своему основному лексическому составу эти языки в 20–50 раз больше, чем *любой* из традиционных языков (такой как *Fortran*, *Lisp*, *Pascal*, *Delphi* или *Java*).
- По гибкости и выразительной силе эти языки превосходят все традиционные языки, так как они не только поддерживают **все** когда-либо использовавшиеся стили программирования, но и вводят громадное количество математических конструкций, никогда ранее в программировании не встречавшихся.
- Программирование в этих языках гораздо ближе к живому языку и реальной математической практике, чем программирование на любом из традиционных языков.

С другой стороны, совершенно очевидно, что в ближайшие 10–15 лет возникнут компьютерные языки еще более высокого уровня, еще более близкие к человеческому языку. В связи с этим мы предлагаем произвести терминологический даунгрейд и всюду в дальнейшем называем *Axiom*, *Maple* и *Mathematica* языками программирования **высокого уровня**, а использованный при их реализации код в *Lisp* или *C* — **низкоуровневым кодом**.

С нашей точки зрения, для *подавляющего* большинства пользователей — ученых, инженеров, экономистов, — **ЗНАНИЕ ВЫСОКОУРОВНЕВЫХ ЯЗЫКОВ КОМПЬЮТЕРНОЙ АЛГЕБРЫ, ТАКИХ КАК Maple или Mathematica ПОЛНОСТЬЮ ОТМЕНЯЕТ НЕОБХОДИМОСТЬ ИЗУЧЕНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ НИЗКОГО УРОВНЯ, ТАКИХ КАК Fortran, Pascal, Java или Delphi.**

**2. Лексический состав языка.** Чтобы понять настоящее место систем компьютерной алгебры, весьма поучительно попытаться сравнить их языки

с другими существующими языками. Вот, например, сравнение с точки зрения количества используемых слов:

- 50–80 слов — язык коллективных насекомых: муравьев, термитов, пчел;
- 80–250 слов — традиционные языки программирования и математические редакторы: Pascal, Fortran, C, C++, Lisp, Java, Delphi, Mathcad;
- 200–500 слов — язык высших человекообразных обезьян: шимпанзе и горилл;
- 300–500 слов — язык специализированных систем компьютерной алгебры GAP, Singular, MAGMA, etc.;
- 500–5000 слов — элементарный уровень овладения языком, достаточен для понимания несложных текстов и коммуникации на примитивном уровне, словарный запас маленьких детей и начинающего изучение иностранного языка;
- 2000–5000 слов — язык ядра и входящих в поставку стандартных пакетов универсальных систем компьютерной алгебры: Axiom, Maple, Mathematica;
- 5000–25000 слов — основной словарный состав языка, достаточный для понимания большинства текстов и адекватной коммуникации; словарный запас человека, хорошо владеющего *иностраным* языком;
- 5000–500000 слов — полный словарный запас носителя языка, как родного или первого, в зависимости от языка и уровня образования;
- 20000–5000000 слов — полный словарный состав национального языка (20000 — суахили и языки австралийских аборигенов, миллионы слов — литературные индоевропейские языки такие, как русский, немецкий и английский).

Таким образом, по объему и выразительной силе системы компьютерной алгебры находятся на полпути между традиционными языками программирования и живыми языками. Это значит, например, что рассматриваемые как языки программирования, эти системы поддерживают гораздо большее разнообразие стилей и приемов программирования, чем *любой* традиционный язык.

**3. Полисемантизм.** Однако, кроме номинального лексического состава языка, есть еще один чрезвычайно существенный аспект, который противопоставляет языки компьютерной алгебры традиционным языкам программирования и сближает их с живыми языками. Мало того, что они используют в десятки раз больше слов, они используют СЛОВА ГОРАЗДО БОЛЕЕ ВЫСОКОГО УРОВНЯ, имеющие несколько значений — во многих случаях несколько *десятков* значений! В отличие от традиционных языков программирования, использование слов в которых чрезвычайно жестко регламентировано, в языках компьютерной алгебры, в особенности в языке *Mathematica*, значение слова может самым существенным образом меняться в зависимости формата, в котором оно вызывается, количества и типа



аргументов, значений параметров, опций и атрибутов, контекста, и многих других обстоятельств.

Например, в языке *Mathematica* функция *Integrate* может означать, в зависимости от обстоятельств, неопределенный интеграл, определенный интеграл, кратный интеграл, интеграл, зависящий от параметра, несобственный интеграл и т.д. Реализация одной этой функции требует 600 страниц кода на C и еще 500 страниц высокоуровневого кода на языке *Mathematica*!!! Более того, в языке *Mathematica* имеются как слова еще более высокого уровня, так и слова, имеющие гораздо больше различных значений. Например, код, при помощи которого реализована команда численного решения дифференциальных уравнений *NDSolve*, занимает 1400 страниц. С другой стороны многие графические команды имеют десятки различных опций и, тем самым, фактически, используются в *десятках* различных смыслов, в том же самом духе, как значение глагола в индоевропейских языках модифицируются приставками, предлогами и послелогами. С учетом полисемантизма основные языки компьютерной алгебры уже не в 20, а по крайней мере раз в 50 богаче всех традиционных языков программирования.

## § 9. ВОЗМОЖНОСТИ СИСТЕМ КОМПЬЮТЕРНОЙ АЛГЕБРЫ.

Ленин, как известно, любил гимнастику. Но все же иногда некоторых тяжестей не осиливал. Так, бывало, схватится за бревно, а поднять от земли — слабо.

Виктор Тихомиров. Легенды о революции

Say, aliens invade the Earth and threaten to obliterate it in one year's time unless we compute  $R(5,5)$ . If we marshalled the world's best minds and fastest computers, then within a year we could probably calculate the value. If the aliens demanded  $R(6,6)$ , however, we would be better off preparing for interstellar war.

Paul Erdős

Воспроизведем из книги Вольфрама (глава “The Limits of Mathematica”) список операций, которые универсальная система компьютерной алгебры такая, как *Mathematica* в состоянии **за 3–4 секунды** выполнить на бытовом компьютере производства 2003–2004 годов работающим под Windows:

- Производить арифметические операции с целыми числами содержащими несколько сотен миллионов десятичных цифр;
- Породить миллион десятичных знаков таких чисел, как  $\pi$  или  $e$ ;
- Разложить по степеням многочлен, содержащий миллион слагаемых;
- Разложить на множители многочлен от четырех переменных, содержащий сто тысяч слагаемых;
- Решить систему квадратичных неравенств, имеющую несколько тысяч независимых компонент;
- Найти целые корни разреженного многочлена степени миллион;

- Применить рекуррентное правило миллион раз;
- Вычислить все простые до десяти миллионов;
- Найти численную обратную плотной матрицы размера  $1000 \times 1000$ ;
- Решить разреженную систему линейных уравнений с миллионом неизвестных и ста тысячами ненулевых коэффициентов;
- Вычислить определитель целочисленной матрицы размера  $250 \times 250$ ;
- Вычислить определитель символьной матрицы размера  $25 \times 25$ ;
- Найти приближенные значения корней многочлена степени 200;
- Решить разреженную задачу линейного программирования с несколькими сотнями тысяч переменных;
- Найти преобразование Фурье списка из нескольких миллионов элементов;
- Изобразить миллион графических примитивов;
- Отсортировать список из десяти миллионов элементов;
- Найти фрагмент в строке из десяти миллионов знаков;
- Загрузить несколько десятков мегабайт численных данных;
- Отформатировать несколько сотен страниц вывода в традиционной форме.

По этому поводу стоит отметить несколько обстоятельств. Прежде всего стоит указать, что производительность другой **high-end** системы, **Maple** может незначительно отличаться в ту или иную сторону, но в любом случае, время выполнения тех же задач в **Maple** также измеряется несколькими секундами. Тем самым, **Mathematica** и **Maple** могут решить *любую* задачу, которая может реально возникнуть у нематематика, за **half no time**, и выбор между ними не может определяться никакими практическими соображениями, а диктуется исключительно индивидуальными предпочтениями. Поскольку **MatLab Extended Symbolic Math Toolbox** представляет собой клон **Maple**, от него также можно ожидать сопоставимой производительности (при несколько большем, чем собственно у **Maple**, расходовании системного ресурса). Разумеется, на **Workstation**, с несколькими гигабайтами оперативной памяти, на которой, к тому же установлен **UNIX**, из всех этих систем можно выжать *гораздо* больше! С другой стороны, производительность **Mathcad** и предельный объем вычислений, которые могут быть при помощи него проведены, ниже во многие сотни тысяч, а иногда в миллионы раз. Например, наибольшая разрядность, которую допускает **Mathcad** — 255 десятичных знаков (4000 знаков без вывода на экран), самая большая матрица, с которой он в состоянии работать —  $10 \times 10$ , и т.д. Иными словами по своим возможностям **Mathcad** мало отличается от большого научного калькулятора. Именно поэтому никто из специалистов не рассматривает **Mathcad** как систему компьютерной алгебры.

От себя добавим в этот список еще один пункт, который Вольфрам не включил ввиду полной очевидности:

• Решение **всех** *вычислительных* задач по математике, которые были выполнены за 15 лет обучения математике в школе и университете.

Подчеркнем, что в этом пункте речь идет о *вычислительных* задачах — таково подавляющее большинство задач, предлагаемых школьникам и студентам. Кроме того, разумеется, учитывается только собственно время работы CPU, без ввода условий задач с клавиатуры и вывода на экран.

**Комментарий.** Интересно отметить, что решение задач “на доказательство” требует в сотни раз больше времени. Дело в том, что единственным систематическим методом доказательства геометрических фактов является введение координат, истолкование геометрических теорем как задач вещественной алгебраической геометрии и последующее применение полиномиальных алгоритмов, связанных с базисами Гребнера. Известно, что уже при нескольких десятках переменных проверка принадлежности многочлена идеалу может представлять собой достаточно серьезную задачу.

С другой стороны, легко придумать и чисто вычислительные задачи, которые поставят в тупик любой современный компьютер и любую систему компьютерной алгебры. Такова, например, задача разложения на множители целого числа со 100 цифрами. Вообще, стоит иметь в виду, что увеличив объем вычислений в задаче, на решение которой требуется несколько секунд, всего в тысячу раз, мы получим задачу, на решение которой требуется час, а в миллион раз — несколько месяцев. При росте объема вычислений всего в миллиарды раз на ее решение потребуются уже столетия! Поэтому полезно понимать, хотя бы в самых общих чертах, как при росте объема данных или значений параметров растет объем вычислений.

## § 10. ОБ “ОШИБКАХ” СИСТЕМ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

At the source of every error which is blamed on the computer you will find at least two human errors, including the error of blaming it on the computer.

The Tao of Real Programming

It seems that creating man God has grossly overestimated his abilities.  
Oscar Wilde

Не только газета, но и короткая запись где-нибудь на стенке в местах общего пользования иногда дает сильный толчок воображению.

Виктор Конецкий, Никто пути пройденного у нас не отберет

Многие наши коллеги, критически настроенные по отношению к введению компьютерной алгебры в преподавание, часто упоминают об “ошибках”, встречающихся в системах компьютерной алгебры. С нашей точки зрения, все слухи о таких ошибках основаны на устаревшей информации и носят чисто пропагандистский характер, а люди, выдвигающие подобные аргументы, либо не владеют реальной ситуацией, либо сознательно лицемерят.

Разумеется, компьютеру свойственно ошибаться, этим он мало отличается от бога. Следует отдавать себе отчет, что определяющим является не возможность наличия ошибок, а вероятность их осуществления и, в особенности, их этиология. Следует четко отличать несколько причин ошибок:

**1. Ошибки системы.** Часть ошибок относится на счет самих систем компьютерной алгебры.

- **Ошибки в математике и алгоритмах**, использованных в системах компьютерной алгебры. Такие ошибки возможны, но исключительно маловероятны. Например, было обнаружено, что первые версии систем Maple или Mathematica неправильно считают некоторые интегралы. Выяснилось однако, что ошибки в таблицах интегралов!

- **Программистские ошибки** в системах компьютерной алгебры также возможны, но для коммерческих систем, прошедших многолетнее тестирование и имеющих миллионы пользователей, крайне маловероятны.

- **Конфликты с операционной системой** и другими приложениями. Известно, что в начале и середине 1990-х годов использование систем компьютерной алгебры на бытовых компьютерах очень часто приводило к системным ошибкам и коллапсу системы. Основная причина этого состояла в том, что когда такой системе не хватало памяти для записи результатов промежуточных вычислений, она начинала писать их поверх собственного ядра и даже поверх системных файлов Windows. Однако никто не слышал про подобные явления для Unix'овских рабочих станций, так что все подобные конфликты следует рассматривать не как дефект систем компьютерной алгебры, а как изъян операционной системы Windows. Кроме того, начиная с Windows NT и Windows2000 эта проблема устранена.

Мы уверены, что если даже ошибки в таких системах, как Maple и Mathematica и имеются, то вероятность встретиться с ними настолько мала, что ей можно *полностью* пренебречь.

**2. Ошибки пользователя.** Гораздо более вероятными представляются нам ошибки, допущенные самим пользователем. Значительная часть этих ошибок ничем не отличается от ошибок, возникающих при любой попытке программирования математических задач.

- **Опечатки.** Известно, что из каждых двадцати символов, введенных человеком с клавиатуры компьютера, по крайней мере один является ошибочным. Опечатка в одном символе в большинстве случаев либо делает выражение бессмысленным, либо приводит к вычислению совершенно не того, что имелось в виду. В отличие от многих примитивных языков программирования в больших системах компьютерной алгебры прописные буквы отличаются от строчных. Таким образом, `TeXForm` значит совсем не то же самое, что `TeXForm`. Однако в подобных случаях эти системы предупреждают о возможных опечатках: `possible spelling error`.

- **Синтаксические ошибки.** Сюда относится любая попытка вычислить выражение, составленное с нарушением правил языка. Типичными

ошибками такого рода являются несбалансированность выражений, вызов в них функций с неправильным числом аргументов или аргументами неправильных форматов, и многое другое. Большая часть этих ошибок моментально обнаруживается, так как система просто откажется вычислять синтаксически неправильное выражение.

• **Программистские ошибки.** Типичные программистские ошибки, возникающие при использовании систем компьютерной алгебры, это использование переменных, которым не были присвоены значения, либо использование старых значений переменных, несогласованность форматов, выясняющаяся при вычислении (т.е. не на чисто лингвистическом, а на семантическом уровне) и многое другое. В большинстве случаев подобного рода ошибки либо порождают сообщение об ошибке (**error message**), либо приводят к бесконечной рекурсии. Однако в некоторых случаях (в особенности при использовании старых значений переменных!!) могут возникать чрезвычайно трудно отслеживаемые *невоспроизводимые* ошибки. Имеются несколько стандартных приемов: чистка и локализация переменных, явное задание начальных значений *всех* используемых итераторов и т.д., которые позволяют резко уменьшить вероятность возникновения неотслеживаемых ошибок. Кроме того, как всегда, правильно вначале тестировать любую написанную программу на совсем простых примерах. Если Вы хотите вычислить  $10000!$ , убедитесь вначале, что написанная Вами программа правильно вычисляет  $3!$  — большинство ошибок будет обнаружено уже на этом этапе.

• **Математические ошибки.** На начальном этапе программирования на языках компьютерной алгебры чрезвычайно велика вероятность возникновения *математических* ошибок. Однако такого рода ошибки характерны для начального этапа *любой* попытки уточнения понятий и их перевода с одного языка на другой. Типичными ошибками такого рода являются ошибки в определении уровней выражения, неправильное определение порядка выполнения операций, применение функции не к той части выражения, неправильная интерпретация функций и т.д. После нескольких месяцев интенсивного использования системы и выработки устойчивого навыка тестировать все программы на легко проверяемых примерах, количество подобных ошибок резко снижается.

Однако с нашей точки зрения перечисленные в этом пункте ошибки не являются специфичными для компьютерной алгебры, а возникают в любом вычислении достаточно большого объема, выполняемом человеком или компьютером.

**3. Непонимание основных принципов компьютерной алгебры.** Однако основным и с нашей точки зрения **НАИБОЛЕЕ СЕРЬЕЗНЫМ** источником *реальных* ошибок является **НЕЗНАНИЕ** и/или **НЕПОНИМАНИЕ** ОСНОВНЫХ ПРИНЦИПОВ ОРГАНИЗАЦИИ ВЫЧИСЛЕНИЙ В СИСТЕМАХ КОМПЬЮТЕРНОЙ АЛГЕБРЫ. В первую очередь это относится к следующим моментам:

• **Непонимание разницы между формой и значением выражения.** В отличие от всех традиционных вычислительных систем, системы компьютерной алгебры производят вычисление с *формой* выражения, а не только с его *значением*. Это значит, что, система тщательно различает не только сами объекты, но и их имена, имена имен, имена имен имен, и т.д. Например, с точки зрения внутреннего представления данных в системе, выражения  $(x + 1)(x - 1)$  и  $x^2 - 1$  следует рассматривать как абсолютно различные!!! Скажем, неосторожно используя условный оператор

If [(x+1)\*(x-1)==x^2-1, 1, 0],

не следует надеяться получить в ответе 1. А вычисление

If [(x+1)\*(x-1)===x^2-1, 1, 0]

и вовсе вернет 0.

• **Применение правил преобразования.** Системы компьютерной алгебры автоматически производят некоторые типы преобразований, но **не** производят других типов преобразований. В большинстве случаев у конструкторов систем были чрезвычайно серьезные принципиальные и/или практические основания для принятия подобного рода решений. В то же время не только начинающий, но даже профессиональный программист, не знакомый, однако, со спецификой символьных вычислений, скорее всего, не осознает разницу между, скажем, применением ассоциативности и применением дистрибутивности, и исходит из того, что система должна проводить вычисление так же, как это делал бы в аналогичной ситуации человек. При некотором опыте в большинстве случаев этого действительно можно добиться. Однако то, что системы компьютерной алгебры не всегда делают то, что от них ожидают, совсем не означает, что они бесполезны!!!

• **Использование приближенных вычислений.** Приближенные вычисления представляют собой меч без ручки и проводящий их вынужден держаться за лезвие. СЕРЬЕЗНЕЙШЕЙ КОНЦЕПТУАЛЬНОЙ ОШИБКОЙ, лежащей в основе большинства реально описанных случаев, когда проведение вычисления приводило к неправильному ответу, является ПРИМЕНЕНИЕ ПРИБЛИЖЕННЫХ ВЫЧИСЛЕНИЙ К ЗАДАЧАМ С ТОЧНЫМИ УСЛОВИЯМИ. Задачи с точными условиями должны обрабатываться только безошибочными алгоритмами. Никаких округлений в процессе вычисления производиться не должно, округляться может только окончательный ответ. Точно так же, применение приближенных вычислений (без контроля точности) внутри рекуррентной или итеративной процедуры в большинстве случаев абсолютно бессмысленно и с *необходимостью* приводит к абсолютно бессмысленному результату<sup>33</sup>.

• **Разбухание промежуточных выражений.** Начинающий обычно не знает, в какой форме задавать вопрос, и пытается вывести на экран то, что

<sup>33</sup>Совершенно поразительные примеры численной неустойчивости, замечательно иллюстрирующие эту мысль, приведены в статье О.А.Иванов, Современная математика в школьных задачах. — Соросовский Образ. Ж., 2000, т.6, N.6, с.1–7.

с точки зрения целей вычисления является *промежуточным* выражением. Например, в действительности его интересует *длина* некоторого списка, но он пытается вывести сам этот список. Так как форматирование ответа для его вывода на экран в большинстве случаев занимает значительно больше времени, чем само вычисление, такое поведение снижает эффективность использования систем компьютерной алгебры, и даже делает невозможным решение некоторых типов задач, которые при грамотной постановке вопроса и/или организации вычислений решаются за доли секунды.

## ГЛАВА 2. ЧТО ТАКОЕ Mathematica?

Математическое понимание представляет собой нечто, в корне отличное от вычислительных процессов; вычисления не могут *полностью* заменить понимание. Вычисление способно оказать пониманию чрезвычайно ценную помощь, однако само по себе вычисление *подлинного* понимания не дает. Однако математическое понимание часто оказывается направленно на *отыскание* алгоритмических процедур для решения тех или иных задач. В этом случае алгоритмические процедуры могут взять управление на себя, предоставив интеллекту возможность заняться чем-то другим. Приблизительно таким образом работает хорошая система обозначений — такая, например, как та, что принята в дифференциальном исчислении, или же всем известная десятичная система счисления. Овладев алгоритмом, скажем, умножения чисел, Вы можете выполнять операцию умножения совершенно бездумно, алгоритмически, при этом в процессе умножения Вам совершенно ни к чему “понимать”, почему в данной операции применяются именно эти алгоритмические правила, а не какие-то другие.

Роджер Пенроуз. Тени разума. Гл.3. О невычислимости в математическом мышлении

Тот, кто прочел Главу 1, знает, что Mathematica является системой компьютерной алгебры общего назначения, при помощи которой можно решать **любой** тип задач, в которых в той или иной форме встречается математика. При этом система Mathematica наряду с Maple является **единственной** такой **high-end**<sup>34</sup> системой, которая настолько проста в использовании, что доступна школьникам и студентам младших курсов.

### § 1. ДОСТОИНСТВА СИСТЕМЫ Mathematica

Одной из самых удивительных сторон компьютеров является то, что они становятся все лучше и лучше, в то время как все остальное становится все хуже и хуже.

Дональд Кнут. Санкт-Петербургский Университет, N.15, 1994.

По своей вычислительной эффективности система Mathematica в целом сопоставима с Axiom и Maple, в чем-то их превосходя, а в чем-то уступая. В то же время по удобству использования, продуманности интерфейса и встроенной помощи, унификации формата применяемых командных слов и конструкций, их предсказуемости и близости к реальному математическому

<sup>34</sup>Словарь дает следующие переводы компьютерного термина **high-end**: мощный, профессиональный, высококачественный; высокого класса; с широкими функциональными возможностями. Поскольку ни один из этих переводов не отражает всего пафоса и всех коннотаций оригинала, мы оставляем этот термин **as is**.



английскому языку, *Mathematica* значительно удобнее **всех** других систем, включая *Maple*. Другими принципиальными моментами, которые заставили нас сделать выбор в пользу системы *Mathematica* в нашей собственной работе, являются поддерживаемый ей более гибкий стиль программирования и более высокое качество графики.

Огромными достоинствами системы *Mathematica* являются

- Простота использования
- Высочайшая вычислительная эффективность
- Эффективная генерация графики высочайшего качества
- Близость используемого языка к реальной математической практике
- Богатство и гибкость языка
- Высочайшая степень унификации
- Высокая предсказуемость
- Неограниченная расширяемость
- Полная независимость от платформы
- Полная совместимость различных версий
- Использование явных форматов

В том, что касается трех последних пунктов, мы можем подтвердить их следующим примером из личного опыта. Один их авторов этой книги впервые познакомился с системой *Mathematica* на презентации фирмы *Wolfram Research* на Международном Математическом Конгрессе в Киото в 1990 году и начал *систематически* использовать ее с 1991 года, в 1991–1992 годах главным образом на платформе *Макинтош*, начиная с 1992 года на UNIX’овских рабочих станциях (DEC, Sun, HP) для научных вычислений, а с 1995 года, кроме того, и на PC для небольших вычислительных задач и учебных целей. При этом **все** программы, написанные начиная с 1991 года (версии 2.0 и 2.2) под *Макинтош* и UNIX оказались *полностью* работоспособными и на PC под версии 3.0, 4.0 и 5.0. Более того, в блокнотах, написанных под *Макинтош*, правильно воспроизводились даже форматирование, свойства клеток и экранная графика. Программы написанные под UNIX в чисто текстовом режиме первоначально нуждались в некотором дополнительном форматировании, чтобы стать полноценным блокнотом, однако в середине 1990-х годов с появлением версий *Mathematica* под XWin и эта проблема была решена.

Подобная политика фирмы *Wolfram Research* находится в разительном контрасте со скудоумным *МелкоМягким* продуктом, когда кодировка русского текста MS-Word в 1997 году больше не поддерживается версией той же программы 1998 года, а (английский!) текст записанный в формате *text only with line breaks*, все равно содержит неявное форматирование, так что о его переносе на WordStar или WordPerfect под *Макинтош* или UNIX без получения миллиона сообщений об инвалидных характерах вообще можно забыть. То же самое относится, конечно, к *Mathcad*. Дело в том, что когда

в качестве системы пропагандируется ее согласованность с другими системами, обычно подразумевается, что она умеет переводить свои неявные форматы в неявные форматы каких-то других систем.

**2. Недостатки системы Mathematica.** Первые версии системы Mathematica обладали рядом серьезных недостатков, начиная с невозможностью прервать вычисление. В сочетании с отсутствием комбинации из трех пальцев Ctrl-Alt-Del в тогдашней MacOS, это часто приводило к драматическим последствиям. Однако в дальнейшем разработчики системы планомерно устраняли все очевидные недостатки: *there are two ways to make a perfect piece of software. One is to make it so simple, that there obviously are no deficiencies. Another one is to make it so complicated, that there are no obvious deficiencies.* Создатели системы Mathematica пошли по второму пути.

Единственным очевидным недостатком системы является высокая стоимость легальной копии (в настоящее время 630\$). Впрочем, компания Wolfram Research предлагает широкую систему студенческих и академических скидок, позволяющих российским ученым и студентам приобрести легальную версию системы начиная со 100\$. Если говорить всерьез, то среди традиционных недостатков системы, которые не исправлены и в версии 5.0, следует упомянуть медленную и не слишком эффективную работу со звуком. Но Вы же не собираетесь *на самом деле* использовать Mathematica для того, чтобы писать музыку, или?? Еще один серьезный недостаток системы Mathematica состоит в том, что она не умеет варить кофе. Впрочем, NOBODY IS PERFECT, ведь даже emacs, который умеет все, варит очень плохой кофе.

Основными конкурентными преимуществами Maple являются несколько более низкая стоимость легальной копии, чуть меньшие требования, предъявляемыми им к аппаратуре и, в первую очередь, интегрированность Maple со многими другими программными продуктами, включая MatLab, SciLab и Scientific WorkPlace. Кроме того, многие ценят Maple за его консерватизм, делающий его гораздо более привычным для приверженцев традиционного процедурного программирования.

## § 2. СТРУКТУРА СИСТЕМЫ Mathematica

There are things on heaven and earth, Horatio, Man was not meant to know. — На свете есть много такого, Горацио, чего человеку знать не положено.

William Shakespear. Hamlet

Mathematica является одной из *самых* сложных до сих пор написанных общедоступных систем программного обеспечения. Более крупные системы как правило создавались лишь для каких-то очень специальных целей, таких как астрономические, ядерные, космические исследования, проектирование, геологоразведка, военное планирование, крупные экономические задачи и т.д.

**1. История системы.** Начиная в 1988 года было выпущено 5 основных версий системы Mathematica. Каждая из них действительно была функциональнее, эффективнее, полнее и удобней предыдущих. Особенно существенные видимые изменения произошли между версиями 1.2 и 2.0 и между версиями 2.2 и 3.0. С другой стороны, версия 4.0 мало отличалась от версии 3.0, а (огромные!) отличия версии 5.0 от предыдущих версий носят внутренний характер и связаны в первую очередь с колоссальным ростом эффективности численных вычислений и работы с графикой. Перечислим официальные релизы системы Mathematica:

- Mathematica 1.0 — 1988 год;
- Mathematica 1.2 — 1989 год;
- Mathematica 2.0 — 1991 год;
- Mathematica 2.1 — 1992 год;
- Mathematica 2.2 — 1993 год;
- Mathematica 3.0 — 1996 год;
- Mathematica 4.0 — 1999 год;
- Mathematica 4.1 — 2000 год;
- Mathematica 4.2 — 2002 год;
- Mathematica 5.0 — 2003 год;
- Mathematica 5.1 — 2004 год.

**2. Структура системы.** Полная инсталляция системы Mathematica под Windows состоит из более, чем 2200 файлов, размещенных в 280 директориях. По умолчанию она пытается установиться в

`C:\Program Files\Wolfram Research\Mathematics\5.0`

в которую вложены три большие поддиректории `SystemFiles`, `Documentation` и `AddOns` и, может быть еще какие-то, скажем, `Configuration`, `Registration` и т.д. Именно в эту директорию Mathematica по умолчанию записывает созданные Вами блокноты и другие документы.

Система Mathematica состоит из следующих основных компонентов:

- **ядро** `Kernel`;
- **интерфейс** `FrontEnd`;
- **процедуры импорта–экспорта** и программы связи `MathLink`, `JLink`, `NETLink` и т.д.
- **библиотеки и пакеты расширений** `Add-ons`, `Packages`, `Dictionaries`, `Graphics`.

В этом и следующих параграфах мы чуть подробнее опишем некоторые аспекты, связанные с реализацией и функциями ядра, а также стандартные пакеты.

**3. MathKernel.** Основной частью системы, определяющей ее вычислительные возможности, является, конечно, ядро `MathKernel`. В свою очередь ядро всех высокоуровневых систем компьютерной алгебры состоит из двух частей. Во-первых, это скомпилированный код на каком-то *низкоуровневом* языке, как правило, `C` или `Lisp`. Во-вторых, это высокоуровневый код в языке самой системы, содержащий таблицы (основные математические формулы, таблицы интегралов, интегральных преобразований и т.д.) В системах `Maple` и `Mathematica` и большинстве других новых систем низкоуровневый код написан на `C` или `C++`, в то время как в `Axiom` и большинстве более старых систем — на `Lisp`. Вот как, примерно, росло **ядро** системы `Mathematica` от версии к версии. Подчеркнем, что речь здесь идет исключительно о коде в `C`, который используется для реализации содержащихся

в ядре алгоритмов, без библиотек, таблиц и пр., написанных собственно в *Mathematica* и без *FrontEnd*, *MathLink* и пакетов расширений:

- Версия 1: 150 000 строк
- Версия 2: 350 000 строк
- Версия 3: 600 000 строк
- Версия 4: 800 000 строк
- Версия 5: 1 500 000 строк

Стоит представить себе, что 1 500 000 строк в *C* это около 50 тысяч печатных страниц, т.е. 100 книг по 500 страниц. Подчеркнем, что речь идет исключительно о коде в *C*. Кроме того, ядро содержит еще 150 000 строк, написанных собственно в *Mathematica*, т.е. языке гораздо более высокого порядка. Эти 1 650 000 строк распределены на следующие четыре примерно равные по объему части:

- Язык и системные функции,
- Численные вычисления,
- Символьные вычисления,
- Графика, звук и форматирование вывода.

В версиях 4.0 и особенно 5.0 резко выросла часть кода, связанная с реализацией численных вычислений.

**4. FrontEnd.** Реализация *FrontEnd* занимает — в зависимости от платформы — еще от 700 000 до 750 000 строк на *C*. Значительная часть этого кода посвящена поддержанию многочисленных форматов ввода-вывода. Кроме того, *FrontEnd* содержит достаточно мощный текстовый редактор (*WordProcessor*). Достаточно сказать, что *Mathematica* содержит словарь и *spell checker* на 125 000 слов, из которых около 100 000 — обычные слова английского языка, около 20 000 — математические и научные термины и около 5 000 — внутренние и системные команды, определенные в ядре *Mathematica* и ее стандартных расширениях.

Эти две части системы полностью независимы и могут работать по отдельности. В действительности, если Вы посмотрите в директорию, где установлена *Mathematica*, Вы найдете там два *exe*-файла:

- *Mathematica.exe*,
- *MathKernel.exe*

(и, быть может, еще файл *math.exe*, который вызывает *MathKernel* под DOS в чисто текстовом режиме, без поддержки графики и файл *mcc.exe* компилирующий внешний код в *C*). Файл *Mathematica.exe* запускает *FrontEnd* в то время как *MathKernel.exe* запускает собственно *ядро* системы *Mathematica*. Обратите внимание, что две эти программы работают полностью независимо. Когда Вы вызываете *FrontEnd* и начинаете редактировать документ (скажем, записную книжку), Вы фактически имеете дело со специализированным *текстовым и графическим редактором*, а вовсе не с системой символьных вычислений. Так происходит до тех пор, пока Вы первый раз не набрали **Shift-Enter**. В этот момент происходит вызов ядра,

которое может загружаться, в зависимости от конфигурации и других выполняемых системой процессов, несколько секунд. С другой стороны, когда Вы вызываете `MathKernel`, Вы обращаетесь *непосредственно* к вычислительному ядру системы, минуя `FrontEnd`, в этом случае общение с ядром происходит в текстовом режиме и графика выглядит несколько ностальгически, напоминая об удаленных терминалах начала 1990-х годов. Взаимодействие между `MathKernel` и `FrontEnd` происходит при помощи программы `MathLink`, т.е. так же, как они взаимодействуют со всеми остальными программами!

### § 3. ГЛАВНОЕ МЕНЮ

Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.

Dick Brandon

При запуске системы `Mathematica` открывается состоящее из 9 разделов **главное меню** `FrontEnd` и новый блокнот с рабочим названием `Untitled-1`. В зависимости от настроек системы, при запуске, кроме того, может открываться приветствие `Welcome Screen` и одна или несколько палитр `Palettes`. В настоящем параграфе мы опишем общую структуру главного меню. Пройдемся коротко по разделам главного меню, опуская при этом описание тех очевидных команд, которые в системе `Mathematica` действуют точно так же как в любой другой программе под `MacOS`, `Windows` или `XWin`. Тот, кто использовал одну такую программу, использовал их все и поэтому нет нужды повторять очевидные вещи. Каждый, кто сам хоть раз включал компьютер, помнит и все горячие клавиши от `Ctrl-A` до `Ctrl-Z`.

- **File** — этот раздел содержит все обычные команды: `New`, `Open`, `Open Special`, `Close`, `Save`, `SaveAs`, `Print`, `Print Selection`, `Exit`. Спецификой системы `Mathematica` является наличие здесь подменю `Save As Special`, содержащего команды, которые позволяют записать клетку или блокнот в различных специальных форматах, скажем, в форматах, согласованных с предыдущими версиями системы, а также в форматах `text only`, `TeX`, `HTML` и `XML`. Из еще одного специфического для системы `Mathematica` подменю `Palettes` открываются 9 палитр, которые описаны в § 5.

- **Edit** содержит обычные команды `Cut`, `Copy`, `Paste`, `Clear`, `SelectAll`, не нуждающиеся в комментариях и подменю `Motion`, описанные в котором команды движения курсора тоже вряд ли нуждаются в комментариях (в действительности, это подменю нужно только для того, чтобы напомнить горячие клавиши `Ctrl-Right`, `Ctrl-Left`, `Ctrl-Del`, `Ctrl-Backspace`). Кроме того, в этом меню содержатся специфические подменю `Copy As`, `Paste As`, `Save Selection As` предписывающие копировать, вклеивать или сохранять буфер в различных текстовых/графических форматах или форматах блокнотов/клеток системы `Mathematica`. Еще одним специфическим подменю является `Expression Input`, которое дублирует палитру `BasicInput`

и позволяет вводить выражения в традиционной математической форме, с верхними и нижними индексами, дробями, радикалами, матрицами и т.д.

- **Cell** содержит специфическое для интерфейса системы *Mathematica*, начиная с версии 3.0, управление клетками. Первые пять подменю **Convert To**, **Display As**, **Default Input Format Type**, **Default Output Format Type**, **Default Inline Format Type** форматируют выбранный фрагмент, либо устанавливают дефолтные форматы клеток. При этом основными возможностями являются четыре формата **FrontEnd**, два явных **InputForm** и **OutputForm**, и два неявных, **StandardForm** и **TraditionalForm**. Например, если Вы установили **Default Input Format Type** на **StandardForm**, то в ячейке инпута  $\rightarrow$  будет автоматически конвертироваться в  $\rightarrow$  и т.д. Хотя, конечно, ядро системы будет, по-прежнему, видеть то, что Вы *фактически* ввели с клавиатуры, а именно,  $\rightarrow$ . Существуют различные мнения по поводу того, удобны неявные форматы или, все же, скорее нет. По нашему мнению, явный формат позволяет гораздо легче локализовать синтаксические ошибки. Поэтому если Вы хотите видеть то, с чем *на самом деле* производится вычисление, оставьте **InputForm** и **OutputForm** в качестве установок ввода и вывода! В некоторых случаях, предлагаются и другие форматы такие, как **Postscript** или **Bitmap**. Два других подменю **Cell Properties** и **Cell Grouping** связаны со свойствами клеток и их группировкой. Команды из подменю **Cell Properties** позволяют изменить такие свойства текущей клетки, как **Open**, **Editable**, **Active**, **Initialization** и т.д. С другой стороны, команды из подменю **Cell Grouping**, позволяет группировать клетки или разбивать группы, в то время как команды **Divide Cell** и **Merge Cells** позволяют разделять и сливать клетки. Кроме того, это подменю содержит некоторые специальные команды работы с клетками, содержащими графические объекты и звук.

- **Format** содержит большое количество подменю, описывающих стили, фонты, размеры, цвет, выравнивание и пр. и пр. **Style**, **Font**, **Face**, **Size**, **Text Color**, **Background Color**, **Text Alignment**, **Text Justification**, **Word Wrapping**, **Cell Dingbat**, **Horizontal Lines**. Самое полезное подменю здесь, это **Magnification**, которое позволяет менять масштаб изображения от 50% до 300%.

- **Input** содержит несколько чрезвычайно полезных команд управления специальным вводом, в частности, графикой и звуком. Например, **3D View-Point Selector** позволяет выбирать точку просмотра трехмерной графики, **Color Selector** позволяет составлять индивидуальную палитру цветов. Команда **Record Sound** обеспечивает запись звука с внешнего источника. Команда **Create Table/Matrix/Palette** порождает таблицу, матрицу или палитру размера  $m \times n$  с предписанными элементами, ограничителями и разделителями.

**Предостережение.** Стоит иметь в виду, что создание при помощи этой команды матрицы, содержащей несколько десятков миллионов элементов, и ее вывод на экран может занимать время от нескольких десятков секунд до нескольких минут, а форматирование матриц, содержащих сотни миллионов элементов — непосильная задача для сего-

дняшних бытовых компьютеров. Остается утешаться тем, что учебные системы такие, как **Mathcad**, вообще не позволяют вводить матрицы, в которых больше 100 элементов. Таким образом, **Mathematica** использует вычислительный ресурс в *сотни тысяч раз* эффективнее, чем **Mathcad**.

- **Kernel** содержит подменю вычисления **Evaluate**, состоящее из команд, позволяющих вычислить содержание одной или нескольких клеток, блокнота и т.д., а также команды приостановки и прерывания вычисления **Interrupt Evaluation** и **Abort Evaluation**, команды управления ядром и т.д.

- **Find** также содержит обычные команды поиска **Find**, **Find Next**, **Find Previous**, **Replace**, **Replace All**, а также команды установки и поиска меток **Tags** и т.д.

- **Window** перечисляет все открытые в системе окна и управляет их расположением.

- **Help** обеспечивает доступ к различным системам информации и помощи, а также другим ресурсам, поддерживающим работу системы. Встроенная помощь более подробно описана в следующем параграфе.

## § 4. GETTING HELP

Лев Толстой очень любил играть на балалайке (и, конечно, детей), но не умел.

Даниил Хармс, Веселые ребята

В системе **Mathematica** есть несколько способов получить помощь, в частности

- возникающее при запуске **FrontEnd** **приветствие** = **Welcome Screen** алиас **Startup Palette**,

- вызываемая из меню **FrontEnd** **встроенная помощь** = **Help Browser**,

- вызываемый непосредственно из записной книжки или другого документа **MathKernel** **информационные запросы** **?** = **Query** и **??** = **Information**.

- справочные команды **Definition**, **FullDefinition**, **Names**.

Обсудим чуть подробнее, что это такое.

**1. Приветствие.** При каждом запуске системы появляется (если Вы его не отключили!!) приветствие, содержащее следующие четыре раздела:

- **курс молодого бойца** = **Ten-minute Tutorial**,
- обращение к **встроенной помощи** = **HelpBrowser**,
- **новое в версии 5** = **What's New in 5**,
- **ссылка на вебсайт** = **Website**.

Назначение двух последних разделов понятно само по себе. Кликнув на раздел **Website**, Вы обращаетесь к сайту **Wolfram Research**, где можно

получить много дальнейшей информации о системе *Mathematica* и других продуктах этой фирмы, книгах по этой системе и т.д. В частности, на этом сайте можно скачать *несколько тысяч* дополнительных пакетов, блокнотов, статей, графических объектов и других документов, содержащих определения *десятков тысяч* функций сверх примерно 5000 тысяч стандартных функций.

Раздел *What's New in 5* перечисляет отличия версии 5.0 от предыдущих версий системы. Наибольший интерес для начинающего представляет *Tutorial*. Если Вы никогда раньше не использовали систему *Mathematica* и не имеете опыта работы в других системах компьютерной алгебры, Вам стоит прочесть этот короткий учебник. В нем на 23 страницах иллюстрируются несколько простейших примеров использования системы *Mathematica* для численных и символьных вычислений, работы с графикой и текстом, анализа данных и пр. Наконец раздел *HelpBrowser* выбрасывает Вас в основной экран встроенной помощи *Mathematica Help Browser*.

**2. Встроенная помощь.** Система *Mathematica* предлагает чрезвычайно детальную интерактивную помощь, которая вызывается через рубрику *Help* главного системного меню, либо нажатием клавиш *F1* или *Shift-F1*. Меню экрана встроенной помощи состоит из навигационных клавиш, окошка поиска и восьми рубрик. Перечислим, что можно найти под ними. Конечно, начинающему имеет смысл самостоятельно обследовать хотя бы часть из них.

- **Built-in Functions** содержит иерархический список всех внутренних и системных функций, классифицированных по девяти разделам

- **Numerical Computation** — численные вычисления;
- **Algebraic Computation** — алгебраические вычисления;
- **Mathematical Functions** — математические функции;
- **Lists and Matrices** — списки и матрицы;
- **Graphics and Sound** — графика и звук;
- **Programming** — программирование;
- **Input and Output** — ввод и вывод;
- **Notebooks** — записные книжки;
- **System Interface** — системный интерфейс.

Внутри каждый из этих разделов содержит алфавитный список всех функций, попадающих в этот раздел, а также для удобства поиска их классификацию по подразделам. Например, раздел *Algebraic Computation* содержит пять подразделов *Basic Algebra*, *Formula Manipulation*, *Equation Solving*, *Calculus*, *Polynomial Functions*. Обращение к имени функции в этих разделах открывает соответствующую страницу помощи, на которой содержится основная информация о функции, примеры ее использования и ссылки на книгу Стивена Вольфрама. Кроме того, в разделе *Built-in Functions* описываются несовместимые отличия версий, дополнительные



функции, имеющиеся в ядре, но не включенные в системный контекст, и некоторые другие полезные вещи.

• **Add-ons & Links** описывает работу с дополнениями к системе **Mathematica**. Наиболее полезным для начинающего здесь является раздел **Standard Packages**, который описывает 115 стандартных пакетов, распространяемых в настоящее время вместе с системой, которые в свою очередь собраны в 11 библиотек:

- **Algebra**,
- **Calculus**,
- **DiscreteMath**,
- **Geometry**,
- **Graphics**,
- **LinearAlgebra**,
- **Miscellaneous**,
- **NumberTheory**,
- **NumericalMath**,
- **Statistics**,
- **Utilities**.

Мы несколько подробнее обсудим содержание этого раздела в § 8.

• **The Mathematica Book** содержит полный текст книги Стивена Вольфрама, состоящей из введения и пяти основных частей:

- **Tour of Mathematica** — экскурсия по системе;
- Практическое введение в **Mathematica**;
- Принципы системы **Mathematica**;
- Продвинутая математика в **Mathematica**;
- Справочный отдел **Reference Guide**.

Грубо говоря, три центральные части можно истолковать как **Mathematica** для пользователя, **Mathematica** для программиста и **Mathematica** для математика. Однако в действительности любой пользователь, который намеревается по настоящему овладеть возможностями системы, должен постараться усвоить хотя бы основные из описанных в центральной части *принципов*. Для опытного пользователя *громадную* ценность представляет справочный отдел, содержащий **ТОЧНЫЕ**, **ПОЛНЫЕ** и **НЕДВУСМЫСЛЕННЫЕ ОТВЕТЫ** на вопросы о том, как именно работают различные классы функций, как в точности система решает вопросы приоритета, и многое другое.

• **FrontEnd** содержит описание всех команд меню и многие другие вещи полезные для опытного пользователя, в частности, списки клавиатурных **short-cut**'ов, форматирование ввода и вывода, настройки опций и токенов, взаимодействие **Mathematica** с различными операционными системами и тому подобное.

• **Getting Started** еще раз объясняет, на уровне **Shift-Enter** (картинка!), что делать при первом запуске системы. Впрочем, некоторые советы в этом разделе, касающиеся работы с записными книжками и кастомизации системы будут полезными не только начинающим.

• **Tour** еще раз выводит на вводную главу книги Вольфрама **Tour of Mathematica**.

• **Demos** представляет собой если не самую полезную, то самую красивую часть **Help Browser**. Он состоит из трех очаровательных демонстрационных галерей,

- **Formula Gallery**,
- **Graphics Gallery**
- **Sound Gallery**,

и еще двух разделов,

- **Notebooks**
- **Palettes**.

Раздел **Notebooks** содержит 21 блистательно исполненную рабочую тетрадь, демонстрирующие хороший стиль программирования на языке **Mathematica** — и каждый, кто хочет действительно приобщиться к йоге системы, должен проработать по крайней мере **Programming Sampler**, чтобы понять, что такое стильная и эффективная программа. Раздел **Palettes** открывает 13 палитр, причем не только те, которые хранятся в папке **SystemFiles/FrontEnd/Palettes** и видны в рубрике **File** главного меню. Вот список представленных тут палитр:

- **Basic Input**,
- **Basic Calculations**,
- **Algebraic Manipulation**,
- **Color Palette**,
- **Polyhedron Explorer**,
- **Periodic Table**,
- **Physical Constants**,
- **Notebook Launcher**,
- **Selection Mover**,
- **Demo Maker**,
- **Basic Typesetting**,
- **Complete Character**
- **International Characters**.

Обращение к этим палитрам позволяет вводить все формулы в стиле **Mathcad**, т.е. вообще без использования клавиатуры. Мы сами обычно не пользуемся большинством из этих палитр, так как помним почти все команды и

соответствующие *short-cut*'ы. Однако, невозможно устоять против очарования и удобства Color Palette, Polyhedron Explorer и Periodic Table.

- **Master Index** является полным алфавитным указателем **всех** встроенных и системных функций, их сокращенных и операторных форм, доступных типографских знаков и т.д., а также всех функций, реализованных в стандартных и демонстрационных пакетах, в нестандартных контекстах и т.д. Поиск в указателе осуществляется либо через окно поиска, либо непосредственно в алфавитном списке.

**3. Информационные запросы.** Как и в большинстве других систем информационные запросы `?name` и `??name` дают информацию об объекте с именем `name`. А именно, ответ на запрос `?name` воспроизводит определение и описывает использование этого объекта. Ответ на запрос `??name`, кроме того, перечисляет его атрибуты и опции вместе с их *текущими* значениями. Полная форма запроса `Information[name]`. Например, если Вы хотите узнать, что делает функция `Plot`, то Вы можете напечатать `??Plot` или `Information[Plot]`

Если Вы точно не помните имя интересующего Вас объекта, то в запросах, как обычно, можно использовать знак `*`, называемый в этом случае `Wildcard` или `MetaCharacter`. Этот знак может ставиться в любое место запроса и заменяет любую конечную последовательность букв, появляющихся в этом месте имени. Например, ответом на запрос `??Plot*` является список из 10 имен `Plot`, `Plot3D`, `Plot3Matrix`, `PlotDivision`, `PlotJoined`, `PlotLabel`, `PlotPoints`, `PlotRange`, `PlotRegion`, `PlotStyle`; в то время как ответом на запрос `??*Plot` будет список из семи функций `ContourPlot`, `DensityPlot`, `ListContourPlot`, `ListDensityPlot`, `ListPlot`, `ParametricPlot`, `Plot`.

Информацию об определении объекта можно получить также при помощи команд `Definition` или `FullDefinition`. А именно, `Definition[name]` дает определение объекта `name`, а `FullDefinition[name]` — определение самого этого объекта и всех объектов, от которых зависит его определение. Для того, чтобы узнать определение встроенного объекта, нужно использовать команду `FullDefinition[name]`, так как в этом случае `Definition[name]` даст только список атрибутов и опций вместе с их текущими значениями.

Еще один народный способ увидеть список всех имен, содержащих фрагмент `blabla` состоит в том, чтобы напечатать `Names["*blabla*"]`. Например, напечатав в начале сессии `Names["*"]` Вы получите список **всех** функций, откомпилированных при загрузке ядра (скажем, в нашей версии *Mathematica 5.0* при первом вызове ядра компилируются 1929 функций). В дальнейшем в течение сессии при подгрузке дополнительных пакетов и по мере того, как Вы определяете новые объекты, список имен будет увеличиваться.

## § 5. ПАЛИТРЫ

А что нам с этих трехсот грамм будет? Мы же гипербореи.

Венедикт Ерофеев

Из подменю **Palettes** раздела **File** главного меню открываются 9 палитр. Эти палитры весьма неравноценны и по своим возможностям палитры **Basic Calculations** и **Complete Characters** неизмеримо превосходят все остальные. Фактически эти две палитры являются **виртуальными расширениями клавиатуры**, которые позволяют вводить **все** обычные символы и проводить **все** обычные математические вычисления, из тех, что могут понадобиться студенту младших курсов технических, экономических и естественнонаучных специальностей, вообще не зная языка **Mathematica**!

- **Algebraic Manipulation** вызывает полтора десятка простейших команд манипуляции с многочленами, рациональными дробями, тригонометрическими и экспоненциальными выражениями. Абсолютно бесполезно тому, кто минут пять работал с системой.

- **Basic Calculations** вызывается как одна палитра, но фактически является блоком из 17 палитр, содержащих **все** операции, которые могут понадобиться студенту. Эти 17 палитр разбиты на следующие 7 разделов:

- **Arithmetic and Numbers** — арифметические операции, возведение в степень, извлечения корней, приближенные вычисления;

- **Algebra** состоит из четырех палитр,

- ★ **Solving Equations**,
- ★ **Polynomial Manipulation**,
- ★ **Simplification**,
- ★ **Complex Numbers**,

названия которых говорят сами за себя. Палитра **Simplification** содержит также описание различных условий типа принадлежности доменам, неравенств, и т.д.

- **Lists and Matrices** состоит из двух палитр

- ★ **Creating Lists and Matrices** — создание матриц по шаблонам, либо посредством функции **Table**;

- ★ **Matrix Operations** — вычисление матричных и векторных произведений, тензорных произведений, сверток, следов, определителей, собственных чисел и векторов, решение систем линейных уравнений.

- **Trigonometric and Exponential Functions** состоит из трех палитр,

- ★ **Trigonometric**,
- ★ **Exponential and Logarithmic**,
- ★ **Hyperbolic**,

из которых вызываются основные элементарные функции и обратные к ним, а также структурные манипуляции над ними.

- **Calculus** состоит из четырех палитр:

★ **Common Operations** — дифференцирование, интегрирование, вычисление сумм и произведений, пределов и рядов;

★ **Differential Equations** — решение обыкновенных дифференциальных уравнений и дифференциальных уравнений в частных производных;

★ **Symbolic Transforms** — прямые и обратные преобразования Фурье и Лапласа, дельта функции и т.д.;

★ **Numeric Root Finding** — приближенное вычисление корней.

○ **Other Functions** состоит из двух палитр:

★ **Integer Functions** — целочисленные арифметические функции, побитовые логические операции, операции округления и т.д.;

★ **Special Functions.**

○ **Graphics** — основные команды построения графиков функций одной и двух переменных.

● **Basic Input** позволяет вводить в традиционной форме верхние и нижние индексы, дроби, радикалы, суммы и произведения, интегралы и частные производные, матрицы, греческие буквы и несколько десятков наиболее употребительных математических знаков.

● **Basic Typesetting** — примерно то же самое, что **Basic Input**, но несколько обширнее и позволяет вводить в традиционной форме не только матрицы, но и таблицы, системы уравнений и пр., около сотни специальных знаков, и некоторые другие фишки и дингбаты, символы специальных клавиш, и тому подобное.

● **International Characters** позволяет вводить буквы с диакритическими знаками, используемые в романских, славянских, германских, и некоторых других языках, символы валют и т.д.

● **Complete Characters** фактически является не одной палитрой, а блоком из 12 палитр, содержащих все символы, определяемые в палитрах **Basic Input**, **Basic Typesetting** и **International Characters**, а также несколько сотен других букв, символов и знаков. Это меню состоит из трех разделов:

○ **Letters** позволяет вводить не только греческие, но и рукописные буквы, буквы с диакритическими знаками, готические буквы и используемый в математике для обозначения уникальных объектов шрифт **BlackBoardBold**, известный также как **DoubleStruck**.

○ **Letter-like Forms** позволяет вводить все обычные технические знаки, все обычные клавиатурные знаки и символы специальных клавиш, дингбаты, некоторые специальные типографские и музыкальные знаки.

○ **Operators** позволяет вводить все обычные символы операций, отношений и стрелки.

Использование этих палитр позволяет *при желании* вводить все математические и типографские символы в традиционной форме в стиле обычном

для Mathcad т.е. вообще без использования клавиатуры. При этом греческие буквы вводятся как греческие буквы, интеграл как  $\int$ , частная производная — как  $\frac{\partial}{\partial x}$  и т.д.

Три оставшиеся палитры

- OpenAuthorTools,
- CreateSlideShow
- NotebookLauncher,

предназначены для быстрого создания документов определенных форматов.

Дополнительные палитры доступны из раздела **Demos** встроенной помощи **HelpBrowser**. Еще раз призываем Вас обратить внимание на замечательные палитры

- Polyhedron Explorer,
- Periodic Table.

При желании можно кастомизировать главное меню так, чтобы палитры автоматически вызывались при запуске системы, вызывались из любого раздела, либо даже создать свои собственные палитры.

## § 6. СЕССИИ И ВЫЧИСЛЕНИЯ

Если Вы работаете с системой Math под Windows, MacOS или XWin, то, скорее всего, основными понятиями, в терминах которых происходит Ваше взаимодействие с системой, на внутреннем уровне (**BackEnd**) являются **сессии** и **вычисления**, а на внешнем уровне (**FrontEnd**) — **блокноты** и **клетки**.

**Предостережение.** Разумеется, это не относится к случаям профессионального использования системы, скажем, в чисто текстовом режиме (в этом случае блокноту отвечал бы файл, а клетке — командная строка), в многопроцессорном или многосессионном режиме и т.д. Однако все это вряд ли представляет интерес для начинающего.

Между внутренними и внешними понятиями можно установить следующее приблизительное соответствие:

<b>Kernel</b>	<b>FrontEnd</b>
<b>Session</b> = сессия	<b>Notebook</b> = блокнот
<b>Evaluation</b> = вычисление	<b>Cell</b> = клетка

Разумеется, это соответствие, а не биекция, во время одной сессии можно открывать несколько блокнотов и, наоборот, записанный блокнот может использоваться на протяжении нескольких сессий. Точно так же одна и та же клетка может в разные моменты сессии вызываться для *различных* вычислений.

**Комментарий.** Непосвященному предыдущее заявление может показаться нелепым, как же один и тот же текст может порождать разные вычисления. Ну, во-первых, текст клетки может редактироваться. Во-вторых, РЕЗУЛЬТАТ ЛЮБОГО ВЫЧИСЛЕНИЯ ЗАВИСИТ не только от того, что мы вычисляем, но и от того, в каком состоянии находится система в момент вычисления. Например, могут измениться определения использованных в этом вычислении функций — или функций, фигурирующих в определении этих

функций, — а также значения различных переменных или параметров, опций и атрибутов. Среди этих параметров могут быть и такие, которые при нормальных условиях не видны пользователю, например, значение заправки генератора случайных чисел, зависящее от времени суток. Кроме того, поведение системы зависит от фаз луны — *You are lucky, full Moon tonight!* — и многих других факторов.

Обсудим эти ключевые понятия чуть подробнее.

С внешней точки зрения **сессия** представляет собой период непрерывной работы ядра `MathKernel` между его вызовом и прекращением его работы (безразлично, по причине сознательного выхода или самопроизвольного коллапса). Все функции сохраняют свои определения, а все переменные — свои значения на протяжении всей сессии, по крайней мере до тех пор, пока эти функции или переменные не были удалены, или их определения и значения не были вычищены или модифицированы. Все введенные во время сессии, но не сохраненные в блокнот или файл определения и все вычисленные, но не сохраненные значения теряются!!!

С внутренней точки зрения сессия представляет собой последовательность **вычислений** (*Evaluation*). Типичная сессия состоит из нескольких десятков, нескольких сотен или, в исключительных случаях, нескольких тысяч вычислений. Для того, чтобы вычислить какое-то выражение, нужно мышкой, либо посредством навигационных клавиш  $\uparrow$  и  $\downarrow$  поместить курсор в содержащую его клетку и нажать **Shift-Enter**.

**Предостережение.** Разумеется, в тот момент, когда мы начинаем *всерьез* использовать *Mathematica* как язык программирования в традиционном стиле, многие вычисления в этом смысле будут состоять в изменении установок каких-то функций, постановке меток, передаче управления, удержании или вбрасывании каких-то значений или даже в том, чтобы *отложить вычисление* какой-то функции. Тем не менее, с точки зрения ядра каждое из этих действий, в том числе удержание и откладывание, является **вычислением** некоторого специального вида и как внутренне, так и внешне оформляется по тому же регламенту, что вычисление  $1+1$ .

- Любое вычисление можно **приостановить** либо программно, командой `Interrupt[]`, либо в диалоговом режиме выбором в разделе `Kernel` главного меню команды `InterruptEvaluation`, горячий ключ **Alt-**, — это легко запомнить, если знать, что **alt = halt** представляет собой основную команду из лексикона итальянских карабинеров.

- Любое вычисление можно **прервать** программно командой `Abort[]`, либо в диалоговом режиме выбором в разделе `Kernel` главного меню команды `AbortEvaluation`, горячий ключ **Alt-**. — *ibid*.

- Первый из этих способов применяется, например, при отладке сложных программ, а второй — в случае, когда вычисление занимает неожиданно много времени и у нас возникло подозрение, постепенно переходящее в уверенность, что либо определения каких-то используемых в нем функций ошибочны, либо значения каких-то параметров слишком велики.

## § 7. Блокноты и клетки

Начиная с версии 3.0 интерфейс *Mathematica* организован в форме **блокнотов** (*Notebook*). Блокнотом называется интерактивный документ, со-

державший программу, текст, результаты вычислений, сообщения, графику, таблицы и т.д. Блокнот может являться аналогом рабочей записной книжки, законченной научной статьи или текста учебного характера. Существующий блокнот открывается при помощи команды **Open** меню **File**, команда **New** того же меню создает *новый* блокнот. Во время сессии Вы можете открывать или создавать несколько блокнотов и, наоборот, один и тот же блокнот может использоваться в большом количестве сессий. В конце сессии не забудьте записать (**Save**) изменения во всех открытых Вами блокнотах.

В свою очередь, каждый блокнот организован как иерархическая структура, состоящая из клеток (**Cell**). Каждая клетка включает одну или несколько строк, соединенных стоящей справа **скобкой** (**Bracket**). Некоторые клетки создаются пользователем, в то время как другие клетки, содержащие сообщения об ошибках, результаты вычисления, сигналы подтверждения, статус системы и другие виды вывода, создаются самой системой в процессе сессии.

Клетки объединяются в **группы** (**Groups**). Входящие в группу клетки соединяются общей скобкой, включающей скобки нескольких объединяемых клеток. Например, система *автоматически* объединяет в одну группу **клетку ввода** (**Input Cell** или **Evaluatable Cell**) и получающиеся при ее исполнении **клетки вывода** (**Output Cell**). При помощи содержащихся в меню **Cell** подменю **Cell Grouping** пользователь может вручную произвольным образом сгруппировать клетки (**Group**) или, напротив, разбить имеющиеся группы (**Ungroup**).

Чтобы начать **новую** клетку, переместите курсор в такую позицию, где он становится *горизонтальным* — внутри *существующей* клетки курсор всегда *вертикален*. Щелкнув в этот момент по *левой* кнопке<sup>35</sup> мыши, Вы создадите горизонтальную черту (**cell insertion bar**). Начав печатать, Вы создадите новую клетку. По умолчанию эта новая клетка всегда имеет формат *клетки ввода*. Чтобы изменить ее формат, щелкните по скобке, после чего найдите нужный формат в меню **Cell**, подменю **Cell Properties**. Другой народный способ начать новую клетку состоит в том, чтобы нажать **Alt-Enter**.

Для того, чтобы **вычислить** (**evaluate**) содержимое клетки ввода, поместите курсор в эту клетку и нажмите **Shift-Enter**. При этом клетке автоматически будет присвоен **промпт ввода** (**Input Prompt**) формата **In[n] :=**, а результат вычисления *через какое-то время* появится в клетке вывода, имеющей заголовок **Out[n]=** с тем же номером. В дальнейшем Вы можете ссылаться на *n*-й ввод как **In[n]**, а на результат *n*-го вывода как **Out[n]**. Обратите внимание на разницу в формате: **In[n]** представляет собой *отложенное* значение и на протяжении сессии его вычисление может приводить к различным результатам, в то время как **Out[n]** представляет

<sup>35</sup> Или, как элегантно выражается шайка **МелкоМягких**, почке.



собой фактическое значение  $n$ -го вывода.

Enter	начать новую строку в текущей клетке
Shift-Enter	вычислить содержание текущей клетки
Alt-Enter	начать новую клетку

На начальном этапе работы придерживайтесь правила

ОДИН ПРОМПТ — ОДНА ФУНКЦИЯ — ОДНА КЛЕТКА

Иными словами, это значит, что в большинстве случаев, когда Вам хочется просто нажать **Enter**, то, что Вам в действительности нужно — это **Alt-Enter!!!**

**Предостережение.** При интерпретации инпута *Mathematica* имеет обыкновение игнорировать большую часть пробелов, табулирования и перенос строки!!! Как показывает наш опыт преподавания, именно непонимание этого фундаментального обстоятельства приводит к половине всех возникающих у студентов ошибок, некоторые из которых трудно отслеживаются. А именно, определив одну функцию, они два раза нажимают на **Enter**, после чего начинают **в той же клетке** определять то, что они считают другой функцией. Но *Mathematica* продолжает интерпретировать все дальнейшее содержание той же клетки, не отделенное точкой с запятой, частью определения первой функции!!! Это значит, что при обращении к этому определению либо (как правило!) обнаружится синтаксическая ошибка, либо начнутся другие увлекательные явления типа бесконечной рекурсии, либо произойдет что-либо еще более драматическое. Довольно часто единственный способ выпутаться из получающегося положения состоит в том, чтобы закончить сессию и перезапустить ядро.

## § 8. ПАКЕТЫ РАСШИРЕНИЙ

Предметы появляются тогда, когда становятся известны их названия.

Виктор Пелевин, Водонапорная башня

Сразу при загрузке ядра системы *Mathematica* компилируется около 2000 функций. Однако фактически это лишь часть общего количества функций, имеющих в системе. Дело в том, что многие функции, к которым обращается лишь часть пользователей — например, несколько сотен комбинаторных функций, несколько сотен статистических функций, несколько сотен функций численной математики, большая часть объектов трехмерной графики и т.д. — не компилируются при запуске по чисто техническим причинам. Основной из этих причин является, конечно, экономия памяти. Кроме того, многие функции пакетов (в особенности это относится к контекстам *Algebra* и *NumberTheory*) используют гораздо более совершенные и мощные, чем это нужно подавляющему большинству пользователей, — но и гораздо более медленные алгоритмы, — скажем, детерминированные,

а не вероятностные тесты проверки простоты. В настоящее время вместе с системой поставляются 115 **стандартных пакетов**, в которых определено около 2500 дополнительных функций. По-существу, различие между функциями ядра и функциями, определенными в *стандартных* пакетах, является чисто техническим, так как эти пакеты являются неотъемлемой частью системы и обращение к содержащимся в них функциям ничуть не сложнее обращения к функциям ядра. В фабричной конфигурации системы эти пакеты разбиты на 11 директорий или, с внутренней точки зрения языка системы, **контекстов**, каждый из которых содержит от 2 до 22 пакетов:

- Algebra содержит 10 пакетов:
  - ★ AlgebraicInequalities,
  - ★ FiniteFields,
  - ★ Horner,
  - ★ InequalitySolve,
  - ★ PolynomialExtendedGCD,
  - ★ PolynomialPowerMod,
  - ★ Quaternions,
  - ★ ReIm,
  - ★ RootIsolation,
  - ★ SymmetricPolynomials;
- Calculus содержит 6 пакетов:
  - ★ DSolveIntegrals,
  - ★ FourierTransform,
  - ★ Integration,
  - ★ Pade,
  - ★ VariationalMethods,
  - ★ VectorAnalysis;
- DiscreteMath содержит следующие 6 пакетов:
  - ★ CombinatorialFunctions,
  - ★ Combinatorica,
  - ★ ComputationalGeometry,
  - ★ DiscreteStep,
  - ★ RSolve, Tree;
- Geometry содержит 2 пакета:
  - ★ Polytopes,
  - ★ Rotations;
- Graphics содержит 22 пакета:
  - ★ Animation,

- ★ ArgColors,
- ★ Arrow,
- ★ Colors,
- ★ ComplexMaps,
- ★ ContourPlot3D,
- ★ FilledPlot,
- ★ Graphics,
- ★ Graphics3D,
- ★ ImplicitPlot,
- ★ InequalityGraphics,
- ★ Legend,
- ★ MultipleListPlot,
- ★ ParametricPlot3D,
- ★ PlotField,
- ★ PlotField3D,
- ★ Polyhedra,
- ★ Shapes,
- ★ Spline,
- ★ SurfaceOfRevolution,
- ★ ThreeScript,
- ★ Common;
- LinearAlgebra содержит 4 пакета:
  - ★ FourierTrig,
  - ★ MatrixManipulation,
  - ★ Orthogonalization,
  - ★ Tridiagonal;
- Miscellaneous содержит 15 пакетов:
  - ★ Audio,
  - ★ BlackBodyRadiation,
  - ★ Calendar,
  - ★ ChemicalElements,
  - ★ CityData,
  - ★ Geodesy,
  - ★ Music,
  - ★ PhysicalConstants,
  - ★ RealOnly,
  - ★ ResonanceAbsorptionLines,

- ★ StandardAtmosphere,
- ★ Units,
- ★ WorldData,
- ★ WorldNames,
- ★ WorldPlot;
- NumberTheory содержит 10 пакетов:
  - ★ AlgebraicNumberFields,
  - ★ ContinuedFractions,
  - ★ FactorIntegerECM,
  - ★ NumberTheoryFunctions,
  - ★ PrimeQ,
  - ★ Primitive Element,
  - ★ Ramanujan,
  - ★ Rationalize,
  - ★ Recognize,
  - ★ SiegelTheta
- NumericalMath содержит 19 пакетов:
  - ★ Approximations,
  - ★ BesselZeros,
  - ★ Butcher,
  - ★ CauchyPrincipalValue,
  - ★ ComputerArithmetic,
  - ★ GaussianQuadrature,
  - ★ InterpolateRoot,
  - ★ IntervalRoots,
  - ★ ListIntegrate,
  - ★ Microscope,
  - ★ NintegrateInterpolatingFunct,
  - ★ NLimit,
  - ★ NResidue,
  - ★ NSeries,
  - ★ NewtonCotes,
  - ★ OrderStar,
  - ★ PolynomialFit,
  - ★ SplineFit,
  - ★ TrigFit;
- Statistics содержит 16 пакетов:

- ★ ANOVA,
- ★ ConfidenceIntervals,
- ★ ContinuousDistributions,
- ★ DataManipulation,
- ★ DataSmoothing,
- ★ Descriptive Statistics,
- ★ DiscreteDistributions,
- ★ HypothesisTests,
- ★ LinearRegression,
- ★ MultiDescriptiveStatistics,
- ★ MultiDiscreteDistributions,
- ★ MultinormalDistribution,
- ★ NonlinearFit,
- ★ NormalDistribution,
- ★ StatisticsPlots,
- ★ Common;
- Utilities содержит 5 пакетов:
  - ★ BinaryFiles,
  - ★ FilterOptions,
  - ★ MemoryConserve,
  - ★ Packages,
  - ★ ShowTime.

Дополнительные функции из пакетов подгружаются посредством команды `Get`, в операторной записи `<<`.

Например, обычные названия цветов не компилируются при запуске ядра. Если Вы хотите вызывать цвета их обычными английскими именами, а не как комбинации `red - green - blue` или `hue - saturation - brightness`, то Вы должны подгрузить их определения. Определения 193 цветов (начиная с `AliceBlue` `AlizarinCrimson`, `<<189>>` вплоть до `YellowOchre` и `Zinc`) содержатся в пакете `Graphics`Colors``. Чтобы загрузить этот пакет, нужно напечатать *с новой строки* `<<Graphics`Colors`` или, что то же самое `Get[Graphics`Colors`]` и нажать `Shift-Enter`. В действительности, если Вы часто используете графику, имеет смысл в самом начале работы загрузить *все* графические пакеты посредством `<<Graphics`` или, если Вы не уверены не были ли уже какие-то из этих пакетов загружены ранее, посредством `Needs["Graphics`"]`. Если Вы в течение длительного времени работаете с каким-то блокнотом, использующим эти дополнительные функции, то удобно объявить клетку с таким содержанием в качестве `Initialization Cell`, так что загрузка всех нужных пакетов будет автоматически производиться при попытке вычислить любую функцию в этом блокноте.

*Несколько сотен* других дополнительных пакетов доступны на сайте [www.wolfram.com/products/applications](http://www.wolfram.com/products/applications)

компании Wolfram Research, некоторые из них являются коммерческими пакетами, но многие другие распространяются бесплатно. Большое количество дополнительных документов доступно также на Mathematica Information Center по адресу [library.wolfram.com](http://library.wolfram.com).

## § 9. КОНЦЕПТУАЛЬНОЕ ПРОГРАММИРОВАНИЕ ВМЕСТО ПРОЦЕДУРНОГО И РЕКУРСИВНОГО

A language that doesn't affect the way you think about programming  
is not worth knowing.

Dennis M. Ritchie

ОСНОВНАЯ ДОГМА традиционного процедурного программирования в стиле *ПошелНа* (GoTo) состоит в том, что компилируемая программа **всегда** выполняется быстрее, чем интерпретируемая. Язык Mathematica поддерживает все стили программирования, включая, конечно, и *процедурное* программирование, хотя в гораздо большей степени ему свойственны *функциональное* программирование, основанное на рекурсии и *концептуальное* программирование, состоящее в том, что мы даем прямое *математическое* определение того, что хотим вычислить.

Проиллюстрируем различные стили программирования на примере вычисления  $n!$ . Конечно, в ядре системы Mathematica есть функция `Factorial`, вычисляющая факториал, поэтому этот пример приводится исключительно с тем, чтобы сравнить на простейшем материале разные стили программирования. Вот как, примерно, могла бы выглядеть в языке Mathematica программа для определения факториала в стиле *ПошелНа*:

```
In[1]:=factor1[n_]:=Block[{m=1},For[i=1,i<=n,i++,m=m*i];m]
```

Однако гораздо быстрее написать рекуррентное определение факториала в стиле функционального программирования

```
In[2]:=factor2[0]=1; factor2[n_]:=factor2[n-1]*n
```

Конечно, более опытный программист, который экономит свое время, а не время компьютера, именно так и поступит. Однако одной из самых сильных сторон языка Mathematica является обилие мощных встроенных функций для работы с выражениями, в частности, списками и применения функций к различным их уровням. Поэтому тот, кто не является программистом, но приобщился к *йогe* системы Mathematica определит  $n!$  либо как произведение элементов списка  $\{1, \dots, n\}$ :

```
In[3]:=factor3[n_]:=Apply[Times,Range[n]]
```

либо просто как произведение чисел от 1 до  $n$ :

```
In[4]:=factor4[n_]:=Product[i,{i,1,n}]
```

А теперь сравним, сколько времени занимает вычисление  $50000!$  при помощи четырех этих программ. Мы думаем, что результат окажется ошеломляющим для сторонников традиционного программирования. Вот как

выглядит фактический результат (для получения которого в зависимости от того, как сконфигурировано ядро Вашей версии, может понадобится изменить глубину рекурсии, сделав ее *достаточно* большой, например, бесконечной `$RecursionLimit=Infinity`):

```
In[5]:=Timing[factor1[50000];]
Out[5]={2.534Second,Null}
In[6]:=Timing[factor2[50000];]
Out[6]={2.493Second,Null}
In[7]:=Timing[factor3[50000];]
Out[7]={0.251Second,Null}
In[8]:=Timing[factor4[50000];]
Out[8]={0.231Second,Null}
```

Производит впечатление, не так ли? Рекурсивная программа работает не медленнее, а даже чуть быстрее, чем процедурная, а работа со списком **в десять раз быстрее**, чем каждая из них!!! А вот с какой, примерно, скоростью вычисляется оптимизированная внутренняя функция `Factorial`:

```
In[9]:=Timing[Factorial[50000];]
Out[9]={0.121Second,Null}
```

Мы видим, что это еще примерно в два раза быстрее, чем работа со списком, что, впрочем, неудивительно, так как код, описывающий встроенные команды, использует быстрые алгоритмы и оптимизирован по скорости.

Философский вывод из этого состоит в следующем: ЧЕМ ПРОЩЕ НАПИСАНА ПРОГРАММА, ТЕМ БЫСТРЕЕ ОНА РАБОТАЕТ! Быстрее всего работают внутренние функции системы `Mathematica`, потом математические определения, которые Вы даете в терминах этих функций. Программы в традиционном стиле — притом как процедурном, так и функциональном!!! — работают значительно медленнее. УСТОЙЧИВЫЕ НАВЫКИ ТРАДИЦИОННОГО ПРОГРАММИРОВАНИЯ ЯВЛЯЮТСЯ СКОРЕЕ ПОМЕХОЙ, ЧЕМ ПОМОЩЬЮ ДЛЯ ЭФФЕКТИВНОГО ПРОГРАММИРОВАНИЯ на языке `Mathematica`, зато понимание смысла и *математической* структуры используемых объектов решающим образом ускоряет не только процесс написания программ, но их работу.

## § 10. ПРОСТЕЙШИЕ ПРАВИЛА И ТИПИЧНЫЕ ОШИБКИ

Nessun effetto è in natura senza ragione; intendi la ragione e non ti bisogna sperienza. — В природе ничто не происходит без причины; пойми эту причину и тебе не будут нужны никакие эксперименты.

Leonardo da Vinci

`Mathematica` is a complex piece of software, and coming to terms with this complexity can impose a steep learning curve upon a student whose primary interest is in learning some mathematics. In my experience of teaching with `Mathematica`, it seems to be a product whic

can too easily divide students very quickly into lovers and haters — not least I'm sure because of its rich but strict syntax.

Philip Kent

В настоящем параграфе мы опишем несколько простейших правил синтаксиса системы *Mathematica*. Все эти правила детально обсуждаются и иллюстрируются в дальнейшем, но по нашему опыту по крайней мере 90–95% всех ошибок, совершаемых начинающими, связано именно с нарушением этих простейших правил. Всякий, кто научился писать `Sin[x*y]^2` вместо `sin^2(xy)`, может решить при помощи системы *Mathematica* *любую* задачу из вузовского курса математики за первые три курса.

- **ВСЕ ИМЕНА ВСТРОЕННЫХ ФУНКЦИЙ** состоят из полных английских слов или общепринятых сокращений и **НАЧИНАЮТСЯ С ЗАГЛАВНОЙ БУКВЫ**, например, экспонента обозначается `Exp`, а не `exp`, логарифм — `Log`, а не `log` и т.д. Если имя функции состоит из нескольких слов, они пишутся слитно, без знаков препинания, причем каждое из них начинается с заглавной буквы.

- **АРГУМЕНТЫ ФУНКЦИЙ ВСЕГДА ПИШУТСЯ В КВАДРАТНЫХ СКОБКАХ**, `cos(x)` обозначается `Cos[x]`, а не `cos(x)`. Различные аргументы разделяются запятой, скажем, `f(x,y,z)` вводится как `f[x,y,z]`. **КРУГЛЫЕ СКОБКИ** используются только для группировки.

- **ФИГУРНЫЕ СКОБКИ** используются для обозначения списков, наборов и множеств. Векторы и матрицы в *Mathematica* интерпретируются как списки. Например, вектор  $c(x,y,z)$  вводится как `{x,y,z}`, а матрица  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$  как `{{a,b},{c,d}}`.

- **ПРОИЗВЕДЕНИЕ** обозначается через `*`. Выражение `xu` интерпретируется не как произведение `x` и `y`, а как новая переменная. Правильное обозначение произведения в операторной форме `x*y`, а в полной форме `Times[x,y]`.

**Предостережение.** В действительности *иногда* — но далеко не всегда!!! — пробел тоже интерпретируется как умножение, так что `x y` будет истолковано точно так же, как `x*y`. Единственный способ узнать, как ядро на самом деле понимает Ваш ввод, состоит, конечно, в том, чтобы взглянуть на полную форму того, что Вы печатаете. Так вот,

```
FullForm[x y]===FullForm[x*y]===Times[x,y],
```

но начинающему лучше не знать об этом и всюду ставить `*`. Есть несколько особых случаев, когда отсутствие пробела после цифры, после специального знака или перед ним тоже интерпретируется как умножение. Например, `2x` значит то же самое, что `2*x`, а `x(y+z)` — то же самое, что `x*(y+z)`, но начинающему лучше вообще не пользоваться подобными сокращениями, тем более, что `x2` значит совсем не то же самое, что `x*2`. Иными словами, `FullForm[x2]===x2`, а `FullForm[x 2]===FullForm[x*2]===Times[2,x]`.

- **ПОТЕНЦИРОВАНИЕ** записывается как `x^y`, или, в полной форме как `Power[x,y]`. Конечно, Вы можете вызвать потенцирование в традиционной форме  $x^y$  через любую из палитр *BasicCalculations*, *BasicInput* или



**BasicTypesetting**, однако через полчаса работы с системой Вы убедитесь, что гораздо быстрее и удобнее набирать его в форме  $x^y$ .

- **НЕ ЖАЛЕЙТЕ СКОБОК.** Представления системы **Mathematica** о приоритете выполнения арифметических операций могут не совпадать с Вашим замыслом. Поэтому за исключением тех случаев, когда Вы абсолютно точно знаете, что происходит, никогда не применяйте два оператора подряд, не указав, в каком порядке они выполняются. Например, вместо  $x/y*z$  пишите  $(x/y)*z$  или  $x/(y*z)$ , as appropriate.

- **МАЛО НАПЕЧАТАТЬ ТЕКСТ, НУЖНО ВВЕСТИ ЕГО В ЯДРО.** До тех пор, пока Вы фактически не предложили системе произвести вычисление, т.е. не нажали **Shift-Enter** на текущей клетке, содержимое этой клетки является достоянием интерфейса, но не ядра, блокнота но не сессии! Тем самым, все введенные в этой клетке функции остаются с точки зрения программы неопределенными, а все переменные, которым в этой клетке присвоено значение — символами.

- По умолчанию одна клетка должна содержать одно вычисляемое выражение. Все пробелы, табулирования и переносы игнорируются. Для ОТДЕЛЕНИЯ ДВУХ ВЫРАЖЕНИЙ ОНИ ДОЛЖНЫ НАХОДИТЬСЯ не в разных строках, а в РАЗНЫХ КЛЕТКАХ. Для создания новой клетки нажмите **Alt-Enter**, либо переведите курсор на одну позицию вниз посредством  $\downarrow$  и начните новый ввод.

- Если Вы все же хотите поместить два или несколько вычисляемых выражений в одну клетку, то они ДОЛЖНЫ РАЗДЕЛЯТЬСЯ ТОЧКОЙ С ЗАПЯТОЙ, скажем  $f[x]; g[x]$ . Однако в этом случае будет отображен только результат последнего вычисления. Если Вы хотите увидеть оба результата, необходимо ОФОРМЛЯТЬ ВЫЧИСЛЕНИЕ КАК СПИСОК, в этом случае все вычисляемые выражения заключаются в фигурные скобки и разделяются запятой  $\{f[x], g[x]\}$ .

## § 11. НЕСКОЛЬКО ОБЩИХ СОВЕТОВ

Experience is that marvelous thing that enables you recognize a mistake when you make it again.

F. P. Jones

Experience is what causes a person to make new mistakes instead of old ones.

Oscar Wilde

- Пишите простые программы. Экономьте свое время, а не время компьютера. Во всех случаях ЯСНОСТЬ И ПРАВИЛЬНОСТЬ ПРОГРАММЫ ВАЖНЕЕ ЕЕ ЭФФЕКТИВНОСТИ.

- Проверяйте, что Вы правильно понимаете использование внутренних функций и ТЕСТИРУЙТЕ РАБОТУ КАЖДОЙ написанной Вами функции на

СОВСЕМ МАЛЕНЬКИХ ПРИМЕРАХ, когда Вы можете проверить правильность ответа в уме, и на примерах с известным ответом.

- Главное правило оптимизации программ: DON'T DO IT!!! Если программа работает, не пытайтесь ее улучшить.

- ПРИСВАИВАЙТЕ ГЛОБАЛЬНЫМ И РЕДКО ВСТРЕЧАЮЩИМСЯ ПЕРЕМЕННЫМ ДЛИННЫЕ ИМЕНА, А ЛОКАЛЬНЫМ И ЧАСТО ВСТРЕЧАЮЩИМСЯ — КОРОТКИЕ ИМЕНА.

- Объясняйте системе то, что Вы хотите посчитать, а не то, как она должна это сделать. По возможности используйте встроенные функции. Не пытайтесь предписывать системе конкретные процедуры. Если Вы не являетесь профессиональным вычислителем, то ЛЮБОЙ ВСТРОЕННЫЙ АЛГОРИТМ ЛУЧШЕ **любого** АЛГОРИТМА, КОТОРЫЙ ВЫ МОЖЕТЕ ПРИДУМАТЬ.

- РАЗБИВАЙТЕ ЛЮБОЙ АЛГОРИТМ НА ПОСЛЕДОВАТЕЛЬНОСТИ ОПРЕДЕЛЕНИЙ. Любая программа, в которой больше, чем две или три строчки, неправильно написана и должна быть разбита на последовательность более коротких программ.

- ПРИМЕНЯЙТЕ К ПРОМЕЖУТОЧНЫМ РЕЗУЛЬТАТАМ ФУНКЦИИ УПРОЩЕНИЯ `Simplify`, `FullSimplify`, `Refine` и т.д. Система, как правило, не производит упрощений автоматически.

- Команды решения уравнений такие как `Solve` как правило дают не все, а лишь *какие-то* решения уравнений. В особенности это относится к трансцендентным уравнениям. ПРОВЕРЯЙТЕ ПОЛУЧЕННЫЕ РЕШЕНИЯ. В случае, если Вам нужно настоящее решение, применяйте команды `Reduce`, `Eliminate` и т.д.

- В РЕКУРРЕНТНЫХ ПРОЦЕДУРАХ ЗАПОМИНАЙТЕ ПРОМЕЖУТОЧНЫЕ РЕЗУЛЬТАТЫ при помощи конструкции `Remember f[n_]:=f[n]=...`

- ЗАМЕНЯЙТЕ ЦИКЛЫ РАБОТОЙ СО СПИСКАМИ или ВСТРОЕННЫМИ ИТЕРАЦИОННЫМИ ПРОЦЕДУРАМИ такими как `Do`, `Sum`, `Product` и т.д.

- НИКОГДА НЕ ПРИМЕНЯЙТЕ ПРИБЛИЖЕННЫЕ ВЫЧИСЛЕНИЯ к ЗАДАЧАМ с ТОЧНЫМИ УСЛОВИЯМИ. Задачи в целых или рациональных числах должны обрабатываться только алгоритмами бесконечной точности. Приближенные вычисления могут применяться только к задачам с приближенными условиями.

- НЕ ПРОИЗВОДИТЕ НИКАКИХ ОКРУГЛЕНИЙ в ПРОЦЕССЕ ВЫЧИСЛЕНИЯ. Проводите все вычисления с бесконечной точностью и переходите к численным значениям только на последнем шаге

- НИКОГДА НЕ ИСПОЛЬЗУЙТЕ ПРИБЛИЖЕННЫЕ ВЫЧИСЛЕНИЯ в РЕКУРРЕНТНЫХ или ИТЕРАЦИОННЫХ ПРОЦЕДУРАХ!! Результат итерационной процедуры, выполненной без контроля сходимости и устойчивости, без изучения обусловленности и оценок ошибок, не может быть ничем иным, кроме артефакта.

### ГЛАВА 3. ПРАКТИЧЕСКОЕ ВВЕДЕНИЕ В СИСТЕМУ Mathematica

Das sieht schon besser aus! Man sieht doch wo un wie! — А вот это уже лучше! По крайней мере видно, куда и как!

Johann Wolfgang von Goethe, Faust I

Математика может быть определена как единственная наука, в которой мы

- ЗНАЕМ, О ЧЕМ МЫ ГОВОРИМ;
- ЗНАЕМ, ЧТО МЫ ГОВОРИМ;
- ЗНАЕМ, ВЕРНО ЛИ ТО, ЧТО МЫ ГОВОРИМ<sup>36</sup>.

Bertran Russell

— Вибрационализм, — сказал Никсим Сколповский, обращаясь к нескольким пожилым женщинам, по виду — работницам фабрики “Буревестник”, непонятно как оказавшимся на авангардной выставке, — это направление в искусстве, исходящее из того, что мы живем в колеблющемся мире и сами являемся совокупностью колебаний.

Женщины испуганно притихли. Никсим поправил непрозрачные очки с узкими прорезями и продолжил: — Но простое отражение этой концепции в артефакте еще не приведет к появлению произведения вибрационалистического искусства. Чистая фиксация идей неминуемо отбросит нас на исхоженный пустырь концептуализма. С другой стороны, возможность вибрационалистической интерпретации любого художественного объекта приводит к тому, что границы вибрационализма оказываются размытыми и как бы несуществующими. Поэтому задача художника-вибрационалиста — проскочить между Сциллой концептуализма и Харибдой теоретизирования постфактум.

Виктор Пелевин. Встроенный напоминатель

В связи с экспансией митьковского движения на север, юг, запад и восток давно пора подготовиться к массовому паломничеству иностранных туристов в места скопления митьков.

Халатное, поверхностное знакомство с митьковской лексикой приводит к быстрому искажению и, в конечном счете, вырождению смысла цитат и выражений.

Отсюда видна необходимость дополнить список употребляемых выражений и адаптировать его до уровня разумения иностранца.

---

<sup>36</sup>Именно в таком порядке: мы только потому знаем, верно ли то, что мы говорим, что мы точно знаем, что именно мы говорим. С другой стороны, мы только потому знаем, что мы говорим, что мы абсолютно точно знаем, о чем именно мы говорим. В ответ на весь слабоумный бред, который по этому поводу несли философы от Гегеля до Виттгенштейна, в этой главе мы констатируем, что у математики есть субстрат. Более того, из этого субстрата она и произрастает.

Итак, что должен знать тот же мистер Майер, американский миллионер, если он является начинающим митьком?

Ниже приводятся новые митьковские слова, выражения и цитаты, которые м-ру Майеру необходимо выучить для полноценного общения с окружающими. В общении с согражданами м-р Майер может произносить большинство выражений по-английски. Александр Флоренский любезно сделал перевод на то, что он считает английским языком. Хотя этот перевод даже увеличивает примитивистскую мощь митьковских выражений, он заставил меня прибегать к обратному переводу.

Владимир Шинкарев. Митьки

В настоящей главе мы описываем работу с системой *Mathematica* как с продвинутым научным калькулятором. Мы объясняем, как проводятся все обычные вычисления с числами, многочленами, рациональными дробями, элементарными функциями, векторами, матрицами, и т.д. Основные излагаемые в этой главе темы включают:

- Решение всех обычных вычислительных задач школьной математики: арифметические операции, алгебраические преобразования, решение уравнений и неравенств, а также систем уравнений и неравенств, исследование функций.

- Построение графиков функций одной и двух переменных, параметрических графиков и графиков неявных функций, графическое решение неравенств и простейшая двумерная графика.

- Решение всех обычно излагаемых на младших курсах университетов задач так называемой “высшей математики”: ряды, произведения и пределы, дифференцирование и интегрирование функций одной и нескольких переменных, решение дифференциальных уравнений и уравнений в частных производных.

- Решение всех обычно излагаемых на младших курсах университетов задач линейной алгебры: решение систем линейных уравнений, умножение и обращение матриц, вычисление определителей, собственных чисел и собственных векторов, канонические формы и т.д.

ПРИМЕРЫ ПОДОБРАНЫ ТАК, ЧТОБЫ после беглого знакомства с содержанием этой главы — даже не вникая в детали — ШКОЛЬНИК ИЛИ СТУДЕНТ МЛАДШИХ КУРСОВ БЫЛ В СОСТОЯНИИ *самостоятельно* РЕШАТЬ ПРИ ПОМОЩИ СИСТЕМЫ *Mathematica* **все** ВСТРЕЧАЮЩИЕСЯ ЕМУ В КУРСЕ МАТЕМАТИКИ *вычислительные* ЗАДАЧИ.

С другой стороны, мы приложили все усилия к тому, чтобы большинство приводимых нами примеров были МАТЕМАТИЧЕСКИ ОСМЫСЛЕННЫМИ И СОДЕРЖАТЕЛЬНЫМИ. Они призваны либо проиллюстрировать уже знакомые читателю явления, либо познакомить его с такими классическими объектами и фактами, которые с большой вероятностью встретятся ему при проведении самостоятельных вычислений.

В отличие от многих наших коллег, настойчиво изгоняющих из преподавания математики всякую конкретику, мы не видим абсолютно **никакого**

**вреда** в том, чтобы начинающий *как можно раньше* услышал о золотом сечении, биномиальных коэффициентах, числах Стирлинга, числах Бернулли, числах Фибоначчи, числах Каталана, гармонических числах, многочленах Бернулли, гауссовых многочленах, многочленах Чебышева, циклотомических многочленах, гамма-функции Эйлера, дзета-функции Римана, фигурах Лиссажу, эллиптических кривых, плоскости Фано, кватернионах и октавах, формуле Фаа ди Бруно, функциях Бесселя, функциях Эйри, интегральном логарифме, интегральной экспоненте, интегральных синусе и косинусе, полилогарифме, гипергеометрических функциях, интегралах Френеля, стандартных матричных единицах, матрицах Картана, матрице Гильберта, якобиевых матрицах, определителе Вандермонда, определителе Коши, и *десяток* других важнейших вещей. Все это классические объекты, которые возникают в самой математике и ее приложениях в *сотнях* самых различных контекстов.

Наша собственная позиция прямо противоположна: мы считаем, что знание определителя Вандермонда *гораздо* важнее знакопеременной формулы для определителя в общем случае. Более того, мы выскажем уже совершенно крамольную мысль, что не только для физика, но и для любого математика-неспециалиста, знание полутора десятков классических дифференциальных уравнений, структуры и поведения их решений, *гораздо* важнее всех теорем существования и единственности решений в общем случае, вместе взятых. МАТЕМАТИКА — ЧРЕЗВЫЧАЙНО КОНКРЕТНАЯ НАУКА и любые **общие факты** только тогда становятся *общими* фактами, когда они опираются на глубокое понимание **примеров**, конкретных частных случаев, притом на *такое* понимание, которое может быть в целом и во всех деталях доведено до исчерпывающего ответа и объяснено школьнику шестого класса. Кроме всего прочего, большинство этих конкретных объектов с необходимостью появляются уже на самых первых шагах в компьютерной математике и анализе алгоритмов — и каждый, кто хочет *профессионально* использовать вычисления, должен быть готов к тому, что мир *настоящей* математики намного богаче и увлекательнее того скуд[оум]ного набора общих фраз, к которым сводятся курсы “элементарной” и “высшей” математики.

Несколько слов о том, чего нет в этой главе. Мы не обсуждаем компьютерные доказательства геометрических теорем. Дело в том, что хотя с технической точки зрения это совсем несложно, при этом требуется принципиально другой уровень алгебраической культуры (знакомство хотя бы с рудиментами коммутативной алгебры и алгебраической геометрии, базисами Гребнера, etc.) Кроме того, мы вообще не упоминаем ряд важнейших общих тем, традиционно вообще не входящих в курсы математики для нематематиков, таких, как комбинаторика, теория чисел и алгебра (вычисления в группах, кольцах, etc.) и ряд более специальных тем, излагаемых на старших курсах: теория вероятностей, оптимизация, etc. Впрочем, многие темы из алгебры, комбинаторики и теории чисел ненавязчиво звучат в дальнейших выпусках — по крайней мере в той степени, в которой это абсо-

лотно необходимо при обсуждении собственно программистских вопросов.

## § 1. АРИФМЕТИКА

Arithmetic is being able to count up to twenty without taking off your shoes.

Mickey Mouse

Системы компьютерной алгебры обесценивают большинство традиционных вычислительных *навыков*, которым обучают в школьном курсе математики. Все задачи школьной математики, вычисления с целыми, рациональными, вещественными и комплексными числами, алгебраические манипуляции, решение уравнений и неравенств, а также систем уравнений и неравенств, построение графиков функций, тригонометрические преобразования, геометрические построения выполняются этими системами за тысячные доли секунды, а доказательство теорем элементарной геометрии — за несколько секунд.

• **Вычисления с целыми числами.** На самом примитивном уровне *Mathematica* является очень мощным калькулятором, работающим с числами неограниченной разрядности, притом работающим с ними гораздо эффективнее, чем подавляющее большинство специализированных численных приложений!!! Сейчас мы предложим *Mathematica* вычислить  $180(2^{127} - 1)^2 + 1$  (это самое большое простое число известное к началу 1952 года, оно было открыто Миллером и Уиллером на компьютере EDSAC1):

```
In[1]:=180*(2^127-1)^2+1
```

```
Out[1]=521064401567922879406069432539095585333589
      8483908056458352183851018372555735221
```

В арифметических вычислениях знаки +, - и / имеют обычный смысл, однако ОБРАТИТЕ ВНИМАНИЕ НА ИСПОЛЬЗОВАНИЕ \* и ^ ДЛЯ ОБОЗНАЧЕНИЯ УМНОЖЕНИЯ И ВОЗВЕДЕНИЯ В СТЕПЕНЬ. Фактически разрядность арифметических вычислений лимитируется *только* объемом памяти, так как даже на пределе этого объема вычисления занимают доли секунды. Например, возведение 2 в степень 1 000 000 000 — дающее число с 301 029 996 знаками — занимает на нашем компьютере четверть секунды и половину оперативной памяти.

А вот еще один очень поучительный диалог, в котором мы спрашиваем, равны ли два числа и получаем ответ, что они действительно равны:

```
In[2]:=2682440^4+15365639^4+18796760^4==20615673^4
```

```
Out[2]=True
```

Иными словами, действительно

$$2682440^4 + 15365639^4 + 18796760^4 = 20615673^4.$$

Эти четыре числа, найденные в 1988 году Ноамом Элкисом<sup>37</sup>, представляют собой контрпример к гипотезе Эйлера, который по аналогии с гипотезой

<sup>37</sup>N.D.Elkie, On  $A^4 + B^4 + C^4 = D^4$ . — Math. Comput., 1988, vol.51, p.825–835.

Ферма предположил, что уравнение  $x^4 + y^4 + z^4 = w^4$  не имеет решений в ненулевых целых числах. Обратите внимание на использование двойного знака равенства `==`, который трактуется как уравнение, в данном случае вопрос о том, равна левая часть правой, или нет. А *parte*: довольно удивительно, что Эйлер задал столь наивный вопрос, ведь ему было известно не только наименьшее решение уравнения  $x^4 + y^4 = z^4 + w^4$  в целых числах:

$$59^4 + 158^4 = 635318657 = 133^4 + 134^4,$$

но и параметрические семейства таких решений. Интересно, что уравнение  $u^5 + v^5 + x^5 + y^5 = w^5$  тоже имеет решение в совсем маленьких числах:

$$27^5 + 84^5 + 110^5 + 133^5 = 61917364224 = 144^5,$$

хотя, по-видимому, найти вручную даже такое совсем маленькое решение довольно трудно.

• **Вычисления с рациональными числами.** Разумеется, вычисления с рациональными числами ничуть не сложнее вычислений с целыми числами — в действительности, это и есть вычисления с парами целых чисел. Вот, например, таблица первых 20 чисел Бернулли  $B_n$ :

```
In[3]:=Table[BernoulliB[n],{n,1,20}]
```

```
Out[3]={-1/2,1/6,0,-1/30,0,1/42,0,-1/30,0,5/66,0,-691/2730,0,
7/6,0,-3617/510,0,43867/798,0,-174611/330}
```

А вот их сумма и произведение чисел с четными номерами

```
In[4]:=Sum[BernoulliB[n],{n,1,20}]
```

```
Out[4]=-932396477/1939938
```

```
In[5]:=Product[BernoulliB[n],{n,2,20,2}]
```

```
Out[5]=-19144150084038739/940848823474560000
```

Обратите внимание на естественные названия операций: таблица называется `Table`, сумма — `Sum`, произведение — `Product`. При некотором навыке и небольшом знании английского языка в большинстве случаев Вы будете в состоянии *угадать* название любой команды *Mathematica* с одной, двух, максимум трех попыток. Если команда, осуществляющая интегрирование, не называется `Integral`, то она просто обязана называться `Integrate`. Обратите внимание также на естественную форму итератора: `{n,1,20}` — сумма по  $n$  от 1 до 20; `{n,2,20,2}` — произведение по  $n$  от 2 до 20 с шагом 2. Эта форма знакома Вам из языка C. Для разнообразия *Maple* использует форму итератора, принятую в *Pascal*, а именно, `n=1..20`.

• **Вычисления с вещественными числами.** Особенностью системы *Mathematica* по сравнению с обычными системами численных вычислений является использование вычислений *бесконечной точности*. Это значит, что НИКАКИХ ОКРУГЛЕНИЙ ПРИ ВЫЧИСЛЕНИЯХ С ТОЧНЫМИ ВЕЩЕСТВЕННЫМИ ЧИСЛАМИ такими, как  $\sqrt{2}$ ,  $e$  и  $\pi$  — или, на языке *Mathematica*,

`Sqrt[2]`, `E`, `Pi` — НЕ ПРОИЗВОДИТСЯ. Если нам нужно десятичное приближение числа  $x$  с точностью до  $n$  значащих цифр, мы можем просто спросить `N[x,n]`. При помощи `Mathematica` мы можем проделать любое вычисление с вещественными числами, которое можно было бы проделать при помощи научного калькулятора с неограниченным количеством разрядов — и много более того. У многих школьников, когда они впервые знакомятся с числами  $\pi$  и  $e$ , возникает вопрос, что больше,  $e^\pi$  или  $\pi^e$ ? Как мы только что объяснили, попытка непосредственно вычислить `E^Pi` и `Pi^E` и посмотреть на то, что получится, не приведет к успеху, так как `Mathematica` трактует эти числа как *точные* вещественные числа, а какие там у точных чисел десятичные знаки? Однако `Mathematica` умеет сравнивать точные вещественные числа. Ввод `E^Pi>Pi^E` дает значение `True`. Конечно, мы можем посмотреть и на любое количество десятичных разрядов, ну, скажем, для начала на 20:

```
In[6]:={N[E^Pi,20],N[Pi^E,20]}
```

```
Out[6]={23.140692632779269006,22.459157718361045473}
```

Мы оформили это вычисление в виде списка, так как хотели увидеть оба ответа вместе.

Бесконечная точность совершенно необходима для того, чтобы контролировать явление, известное в компьютерной алгебре как **high-precision fraud**<sup>38</sup>. Дело в том, что недостаточная точность часто приводит к появлению **артефактов**<sup>39</sup>. Известно много поразительных примеров, когда значения двух естественно заданных функций совпадают с точностью до тысяч и даже миллионов знаков после запятой, так что никакие приближенные вычисления не позволяют сказать, равны эти значения или нет. Вот один из самых знаменитых примеров — число  $e^{\pi\sqrt{163}}$ , настолько близко к целому, что даже вычисление 12 знаков после запятой не позволяет сказать, что это число не целое:

```
In[7]:=NumberForm[N[Exp[Pi*Sqrt[163]]],30],
```

```
ExponentFunction->(Null&)]
```

```
Out[7]=262537412640768743.999999999999
```

Для профессионалов заметим, что этот факт ошеломителен лишь на первый взгляд, после секундного раздумья каждый компетентный математик должен увидеть его связь с тем фактом, что кольцо  $\mathbb{Z}[\sqrt{-163}]$  является кольцом главных идеалов!!! И действительно, числа  $e^{\pi\sqrt{67}}$  и  $e^{\pi\sqrt{43}}$  тоже чрезвычайно близки к целым, хотя, конечно, и не с такой изумительной точностью. Мы уже знаем, что `Sqrt[163]` обозначает  $\sqrt{163}$ , где `Sqrt` — стандартное сокращение от `Square Root`, принятое в большинстве языков программирования,

<sup>38</sup>J.M.Borwein, P.B. Borwein, Strange series and high precision fraud. — Amer. Math. Monthly, 1992, vol.99, N.7, p.622–640.

<sup>39</sup>**Артефактом** в компьютерной алгебре называется ошибочный результат или наблюдаемое явление, не отвечающее существу рассматриваемой задачи, порожденные недостаточной точностью вычисления и/или использованием в процессе вычисления плохих алгоритмов (ошибочных, неустойчивых или очень медленно сходящихся).



`Exp[x]`, естественно, обозначает экспоненту  $e^x$ , а функция `N[x,30]` показывает нам первые тридцать десятичных знаков числа  $x$ . Использование функции `NumberForm` и опции `ExponentFunction->(Null&)` нужно только для того, чтобы *Mathematica* не пыталась выражать эти числа в научной форме, с разделением мантииссы и порядка. Мы могли бы с таким же успехом напечатать просто `N[Exp[Pi*Sqrt[163]],30]`, но тогда, конечно, ответ имел бы менее наглядную форму,  $2.62537412640768743999999999999 \times 10^{17}$ , в которой нужно было бы еще мысленно сдвинуть десятичную точку на 17 позиций вправо. Однако небольшое дальнейшее увеличение точности показывает, что это число, все-таки, не целое:

```
In[8]:=NumberForm[N[Exp[Pi*Sqrt[163]],40],ExponentFunction->(Null&)]
```

```
Out[8]=262537412640768743.9999999999992500725972
```

В приведенных выше примерах мы ограничивались небольшим количеством десятичных знаков. В действительности *Mathematica* может работать с сотнями тысяч или миллионами десятичных знаков. Время, нужное для вычисления первого миллиона знаков  $\pi$ ,  $e$  и  $\sqrt{2}$  или первых ста тысяч знаков  $e^\pi$ ,  $\pi^e$  и  $\sqrt{2}^{\sqrt{3}}$ , исчисляется несколькими секундами или, в худшем случае (профессиональным вычислителям предлагается угадать, какой случай худший!), десятком секунд. Но не пытайтесь выводить все эти знаки на экран, так как форматирование вывода занимает гораздо больше времени, чем само вычисление!!! Таким образом, единственным реальным ограничением для дальнейшего увеличения разрядности становится оперативная память используемого Вами компьютера.

• **Вычисления с комплексными числами.** Ясно, что вычисления с комплексными числами ничуть не сложнее — а на самом деле часто проще! — чем с вещественными. Комплексное число  $z \in \mathbb{C}$  можно задавать, например, в алгебраической форме  $z = x + iy$ , где  $x, y \in \mathbb{R}$  суть его вещественная и мнимая части, соответственно, а  $i$  — мнимая единица,  $i^2 = -1$ . Нас не должно удивлять, что в *Mathematica* следует печатать `z=x+I*y` — мы уже привыкли, что ИМЕНА ВСЕХ ВНУТРЕННИХ ОБЪЕКТОВ НАЧИНАЮТСЯ С ЗАГЛАВНОЙ БУКВЫ. Если *Mathematica* уверена, что  $x$  и  $y$  вещественные числа (например, если она *знает*, что это вещественные числа или если мы специфицировали домен, которому принадлежат  $x$  и  $y$ , и этот домен содержится в домене вещественных чисел), то она истолковывает их как вещественную и мнимую часть  $z$ :

```
In[9]:=z=E+I*Pi; {Re[z],Im[z],Abs[z],Arg[z],Conjugate[z]}
```

```
Out[9]={E,Pi,Sqrt[E^2+Pi^2],ArcTan[Pi/E],E+I*Pi}
```

Снова мы видим, что в *Mathematica* все функции называются так, как они НА САМОМ ДЕЛЕ называются:

- `Re[z]` — вещественная часть  $z$ ,
- `Im[z]` — мнимая часть  $z$ ,
- `Abs[z]` — модуль  $z$  (абсолютная величина),

◦ `Arg[z]` — аргумент  $z$ ,

◦ `Conjugate[z]` — сопряженное к  $z$  комплексное число,

и т.д. Именно высочайшая степень предсказуемости и согласованности языка системы **Mathematica** с обычным математическим языком облегчают ее использование по сравнению со всеми остальными системами сопоставимой силы и относятся к числу ее главных достоинств. Ясно, однако, что **Mathematica** ориентируется на англоязычную терминологию и типографские традиции. Так, например, в этом примере мы видим, что  $\arctg(x)$  обозначается через  $\arctan(x)$  или, на языке системы `ArcTan[x]` — каждый корень, входящий в состав многосложного слова, пишется с заглавной буквы.

В школьной математике принято заучивать значения основных тригонометрических функций углов  $60^\circ$ ,  $45^\circ$  и  $30^\circ$ . Между тем есть еще один столь же замечательный угол, а именно,  $36^\circ$ , значения основных тригонометрических функций в котором являются квадратичными иррациональностями. Иными словами, как знали еще древние греки, круг можно разделить на 5 — или, что то же самое, на 10 — равных частей при помощи циркуля и линейки. Определим комплексное число  $\eta$ , вещественная часть которого равна  $\cos(\pi/5)$ , а мнимая —  $\sin(\pi/5)$ :

```
In[10]:=eta=Cos[Pi/5]+I*Sin[Pi/5]
```

$$\text{Out}[10]=\frac{1}{2}*I*\sqrt{\frac{1}{2}*(5-\sqrt{5})}+\frac{1}{4}*(1+\sqrt{5})$$

Попробовав вычислить  $\eta^2$  мы не получим ничего интересного:

```
In[11]:=eta^2
```

$$\text{Out}[11]=\left(\frac{1}{2}*I*\sqrt{\frac{1}{2}*(5-\sqrt{5})}+\frac{1}{4}*(1+\sqrt{5})\right)^2$$

Дело в том, что — и мы многократно столкнемся с этим в дальнейшем! — ни одна интеллигентная система компьютерной алгебры не проводит преобразований, если она не уверена, что их применение упростит форму исходного выражения. В частности, она автоматически не применяет дистрибутивность ни в ту ни в другую сторону или, пользуясь школьным жаргоном, не раскрывает скобок и не выносит общие множители. В тот момент, когда Вы поймете, *почему* системы компьютерной алгебры устроены таким образом, Вы сделаете решающий шаг к пониманию того, как добиться от системы ответа на любой вопрос в интересующей Вас форме. В частности, для комплексных чисел — и комплексных функций! — в системе имеется функция `ComplexExpand`, которая заставляет искать вещественную и мнимую часть. Поэтому если Вы хотите увидеть  $\eta^2$  в алгебраической форме, нужно просить чуть настойчивее:

```
In[12]:=ComplexExpand[eta^2]
```

$$\text{Out}[12]=-\frac{1}{4}+\frac{\sqrt{5}}{4}+I*\left(\frac{1}{4}*\sqrt{\frac{1}{2}*(5-\sqrt{5})}+\frac{1}{4}*\sqrt{\frac{5}{2}*(5-\sqrt{5})}\right)$$

Теперь Вас, вероятно, уже не удивит, что спросив `ComplexExpand[eta^5]` мы получим  $-1$ .

## § 2. МНОГОЧЛЕНЫ И РАЦИОНАЛЬНЫЕ ДРОБИ

Чуда не вижу я тут. Генерал-лейтенант Захаржевский  
В урне той дно просверлив, воду провел чрез нее.

Алексей Константинович Толстой, Царскосельская статуя

Каждый, кто заглядывал в ее внутреннее устройство, знает, что СЕРДЦЕВИНУ СИСТЕМЫ *Mathematica* СОСТАВЛЯЮТ ВЫЧИСЛЕНИЯ С МНОГОЧЛЕНАМИ ОТ НЕСКОЛЬКИХ ПЕРЕМЕННЫХ. Здесь *Mathematica* в своей стихии и заведомо превосходит остальные системы общего назначения — *nobody can beat me in the kitchen*. Начнем с чего-нибудь совсем простенького.

• **Вычисления с многочленами.** Любая *новая* переменная, которой до сих пор не присваивалось значения, рассматривается системой как **независимая переменная**. Например, если мы до сих пор не присваивали значений  $x, y, z, u, v, w$ , то мы можем использовать их как независимые полиномиальные переменные. В действительности, конечно, в предыдущем параграфе мы уже присвоили  $z$  значение, так что если мы хотим снова использовать ее как переменную, мы должны удалить ее значения и определение посредством `ClearAll[z]` или, может быть, даже посредством радикального `Remove[z]`. Если мы этого не сделаем, система будет подставлять вместо  $z$  ее старое значение  $E+I\pi$ , а это, видимо, совсем не то, что мы хотели. Самая страшная тайна компьютерной алгебры состоит в том, что ИСПОЛЬЗОВАНИЕ СТАРЫХ ЗНАЧЕНИЙ ПЕРЕМЕННЫХ ЯВЛЯЕТСЯ ОСНОВНЫМ ИСТОЧНИКОМ ОШИБОК!!! Если нам не хватает букв латинского алфавита, мы можем использовать в качестве имени переменной любое слово, скажем, `xx, xxx, xy, strength, dexterity, constitution, wisdom, intelligence, charisma, hitpoints, armourclass, experience, bandwidth`, и т.д. Иногда, например, если Вы обзовете переменную `length, depth` или `power`, система будет *слегка встревожена* (`slightly alarmed`), так как будет считать, что Вы хотели обратиться к какой-то из ее встроенных функций, но сделали опечатку, набрав строчную букву вместо заглавной. Однако после того, как Вы один раз проигнорируете ее жалобу, она внесет это новое имя в глобальный контекст и в дальнейшем уже не будет проявлять по этому поводу никаких признаков беспокойства. Основное правило грамотного программирования состоит в том, чтобы ПРИСВАИВАТЬ ЛОКАЛЬНЫМ И ЧАСТО ВСТРЕЧАЮЩИМСЯ ПЕРЕМЕННЫМ КОРОТКИЕ ИМЕНА, А ГЛОБАЛЬНЫМ И РЕДКО ВСТРЕЧАЮЩИМСЯ ПЕРЕМЕННЫМ — ДЛИННЫЕ ИМЕНА. Если Вам не хватает букв, Вы можете использовать цифры, например, `x1, x2, x3, x4` могут обозначать  $x_1, x_2, x_3, x_4$ . Единственное правило, которого при этом нужно придерживаться, состоит в том, что имя переменной не может *начинаться* с цифры. Однако имейте в виду, что если Вам нужно несколько десятков, сотен или тысяч однородных переменных, то их следует оформлять в виде массива `Array` или списка `List, Table` — мы сами *обычно* так и поступаем уже с тремя или четырьмя переменными!

С многочленами можно проделывать все обычные операции:

- о арифметические операции,

- деление с остатком `PolynomialQuotient`, `PolynomialRemainder`, `PolynomialMod`, `PolynomialReduce`,

- композицию (подстановку многочлена в многочлен),

- отыскание наибольшего общего делителя `PolynomialGCD` и наименьшего общего кратного `PolynomialLCM`,

и т.д. Кроме того, с многочленами можно проделывать различные структурные манипуляции такие как

- разложение по степеням какой-то из переменных `Expand`,

- разложение на множители `Factor`,

и т.д. Приведем несколько совсем простых примеров.

• **Операции над многочленами.** Проиллюстрируем операции над многочленами на примере **круговых многочленов**  $\Phi_n(x)$ . Напомним, что

$$\Phi_n(x) = \prod (x - \varepsilon_i),$$

где произведение берется по всем **первообразным** корням  $\varepsilon_i$  из 1 степени  $n$ . Эти многочлены естественно возникают в сотнях различных вычислений, как в самой алгебре и теории чисел, так и в многочисленных приложениях, связанных с теорией конечных полей, включая `Computer Science`, теорию передачи информации и криптографию. В `Mathematica`  $n$ -й круговой многочлен  $\Phi_n(x)$  называется `Cyclotomic[n,x]`. Взглянем на несколько — для круглого счета 16 — первых круговых многочленов:

```
In[13]:=Do[Print[Cyclotomic[n,x]],{n,0,15}]
```

```
Out[13]=1
```

```
-1+x
```

```
1+x
```

```
1+x+x^2
```

```
1+x^2
```

```
1+x+x^2+x^3+x^4
```

```
1-x+x^2
```

```
1+x+x^2+x^3+x^4+x^5+x^6
```

```
1+x^4
```

```
1+x^3+x^6
```

```
1-x+x^2-x^3+x^4
```

```
1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^10
```

```
1-x^2+x^4
```

```
1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^10+x^11+x^12
```

```
1-x+x^2-x^3+x^4-x^5+x^6
```

```
1-x+x^3-x^4+x^5-x^7+x^8
```

Команда `Print` выводит результат каждого индивидуального вычисления в отдельной строке — иными словами, вставляет `\newline` после каждого выражения. Она применяется, если мы хотим посмотреть на результаты нескольких вычислений в одной клетке, но не хотим при этом явно структурировать их в виде списка. Обратите внимание на организацию цикла при помощи команды `Do` с обычной в языке `C` формой итератора  $\{n, 0, 15\}$  — по  $n$  от 0 до 15 (с шагом 1). Мы могли бы достичь того же результата следующим образом: вначале *породить список* циклотомических многочленов при помощи обсуждаемой в § 10 команды `Table`, а потом **проскан(д)ировать** (`Scan`) элементы этого списка командой `Print`:

```
In[14]:=Scan[Print,Table[Cyclotomic[n,x],{n,0,15}]]
```

Попробуем теперь перемножить два круговых многочлена. Беззастенчивое вычисление `Cyclotomic[5,x]*Cyclotomic[6,x]` ничего не даст, так как `Mathematica` НЕ РАСКРЫВАЕТ СКОБОК АВТОМАТИЧЕСКИ. Правильная форма вопроса, если мы хотим получить ответ, разложенный по степеням  $x$ , должна включать структурную команду `Expand`:

```
In[15]:=Expand[Cyclotomic[5,x]*Cyclotomic[6,x]]
```

```
Out[15]=1+x^2+x^3+x^4+x^6
```

Деление многочленов от одной переменной осуществляется при помощи команд `PolynomialQuotient` — неполное частное и `PolynomialRemainder` — остаток. Остаток можно найти и при помощи более общих команд `PolynomialMod` или `PolynomialReduce`, предназначенных для нахождения остатка многочленов от нескольких переменных по модулю нескольких многочленов. Вот пример деления с остатком:

```
In[16]:=PolynomialQuotient[Cyclotomic[11,x],Cyclotomic[5,x],x]
```

```
Out[16]=x+x^6
```

```
In[17]:=PolynomialRemainder[Cyclotomic[11,x],Cyclotomic[5,x],x]
```

```
Out[17]=1
```

Обратите внимание на синтаксис: команды деления многочленов от одной переменной `PolynomialQuotient` и `PolynomialRemainder` вызываются с *тремя* аргументами, в формате `PolynomialQuotient[f,g,x]`, где  $f$  — делимое,  $g$  — делитель, а  $x$  — переменная, по которой производится деление.

• **Структурные манипуляции.** Из школы все помнят формулу для  $(x+y)^n$  известную под кодовым названием “бином Ньютона”. Но кто, кроме профессиональных математиков, видел **мультиномиальную формулу Лейбница** для  $(x_1 + \dots + x_m)^n$ , за исключением, может быть, случая  $(x+y+z)^3$ ? Вот один из первых интересных примеров. Как мы уже знаем, попытка вычислить  $(w+x+y+z)^4$  не приведет к успеху, так как `Mathematica` автоматически не раскрывает скобок. Это значит, что мы должны попросить ее сделать это:

```
In[18]:=Expand[(w+x+y+z)^4]
```

```
Out[18]=w^4+4*w^3*x+6*w^2*x^2+4*w*x^3+x^4+
```

$$\begin{aligned}
& 4w^3y + 12w^2xy + 12wx^2y + 4x^3y + 6w^2y^2 + \\
& 12wx^2y^2 + 6x^2y^2 + 4wy^3 + 4x^2y^3 + y^4 + \\
& 4w^3z + 12w^2xz + 12wx^2z + 4x^3z + \\
& 12w^2yz + 24wx^2yz + 12x^2yz + 12wy^2z + \\
& 12x^2yz + 4y^3z + 6w^2z^2 + 12wx^2z^2 + \\
& 6x^2z^2 + 12wy^2z^2 + 12x^2yz^2 + 6y^2z^2 + \\
& 4wz^3 + 4xz^3 + 4yz^3 + z^4
\end{aligned}$$

А сейчас мы предложим системе проделать эту операцию в обратную сторону. Из школы все помнят “формулы сокращенного умножения”  $x^2 - y^2 = (x - y)(x + y)$  и  $x^3 - y^3 = (x - y)(x^2 + xy + y^2)$ , но как раскладывается на множители что-нибудь чуть большей степени, ну хотя бы  $x^{100} - y^{100}$ ? Нет ничего проще:

```

In[19]:=Factor[x^100-y^100]
Out[19]=(x-y)(x+y)(x^2+y^2)
          (x^4-x^3y+x^2y^2-xy^3+y^4)
          (x^4+x^3y+x^2y^2+xy^3+y^4)
          (x^8-x^6y^2+x^4y^4-x^2y^6+y^8)
          (x^20-x^15y^5+x^10y^10-x^5y^15+y^20)
          (x^20+x^15y^5+x^10y^10+x^5y^15+y^20)
          (x^40-x^30y^10+x^20y^20-x^10y^30+y^40)

```

• **Многочлены от нескольких переменных.** Эйлер заметил, что произведение двух сумм четырех квадратов снова является суммой четырех квадратов:

$$\begin{aligned}
& (x_1^2 + x_2^2 + x_3^2 + x_4^2)(y_1^2 + y_2^2 + y_3^2 + y_4^2) = \\
& (x_1y_1 - x_2y_2 - x_3y_3 - x_4y_4)^2 + (x_1y_2 + x_2y_1 + x_3y_4 - x_4y_3)^2 + \\
& (x_1y_3 + x_3y_1 - x_2y_4 + x_4y_2)^2 + (x_1x_4 + x_4y_1 + x_2y_3 - x_3y_2)^2
\end{aligned}$$

В 1842 году Гамильтон осознал, что это замечательное тождество, известное как **тождество Эйлера**, можно принять за определение умножения четверок вещественных чисел, которое превращает  $\mathbb{R}^4$  в алгебру с делением  $\mathbb{H}$ , известную как **тело кватернионов**. Попробуем проверить тождество Эйлера с помощью системы Mathematica. Для этого совершенно бесхитростно предложим ей провести следующее вычисление:

```

In[20]:=z1=x1*y1-x2*y2-x3*y3-x4*y4; z2=x1*y2+x2*y1+x3*y4-x4*y3;
          z3=x1*y3+x3*y1-x2*y4+x4*y2; z4=x1*y4+x4*y1+x2*y3-x3*y2;
          z1^2+z2^2+z3^2+z4^2
Out[20]=(x4*y1-x3*y2+x2*y3+x1*y4)^2+(x3*y1+x4*y2+x1*y3-x2*y4)^2+
          (x2*y1+x1*y2-x4*y3+x3*y4)^2+(x1*y1-x2*y2-x3*y3-x4*y4)^2

```

Постараемся понять, что произошло. Во-первых, мы определили новые переменные  $z_1, z_2, z_3, z_4$ , стоящие в скобках в правой части тождества Эйлера, и задали их выражение через исходные переменные  $x_1, x_2, x_3, x_4$  и

$y_1, y_2, y_3, y_4$ . После этого мы предложили системе *вычислить*  $z_1^2 + z_2^2 + z_3^2 + z_4^2$ , но она просто подставила сюда выражения  $z_i$  через  $x_i$  и  $y_i$ . Мы уже встречались с этим явлением. Дело в том, что система не уверена, приведет ли раскрытие скобок к более короткому выражению, и ждет нашего явного указания сделать это. Обратите внимание, что РАЗЛИЧНЫЕ ВЫРАЖЕНИЯ внутри одного ввода ДОЛЖНЫ РАЗДЕЛЯТЬСЯ ТОЧКОЙ С ЗАПЯТОЙ. Конечно, мы можем заставить систему раскрыть скобки при помощи команды **Expand**, но тогда она оставит результат в виде суммы одночленов. После этого мы можем применить к этой сумме одночленов команду **Factor** и она попытается разложить их на целочисленные множители. Однако проще всего довериться ее собственному эстетическому чувству и посредством какой-либо из команд **Simplify**, **FullSimplify** или **Refine** предложить ей *упростить* выражение  $z_1^2 + z_2^2 + z_3^2 + z_4^2$ . В этом случае она будет пытаться применить к этому выражению известные ей преобразования и искать среди получающихся результатов самый простой. Первая же попытка приводит к тождеству Эйлера:

```
In[21]:=Simplify[z1^2+z2^2+z3^2+z4^2]
```

```
Out[21]=(x1^2+x2^2+x3^2+x4^2)*(y1^2+y2^2+y3^2+y4^2)
```

• **Многочлены Чебышева.** В школьном курсе тригонометрии встречаются формулы  $\cos(2\phi) = 2\cos(\phi)^2 - 1$  и  $\cos(3\phi) = 4\cos(\phi)^3 - 3\cos(\phi)$ . Одним из самых важных классических объектов математики являются **многочлены Чебышева** первого рода  $T_n$ , при помощи которых  $\cos(n\phi)$  выражается через  $\cos(\phi)$ . По определению,  $\cos(n\phi) = T_n(\cos(\phi))$ . Таким образом,  $T_0(x) = 1$ ,  $T_1(x) = x$  и, как мы только что вспомнили,  $T_2(x) = 2x^2 - 1$  и  $T_3(x) = 4x^3 - 3x$ . Естественно, многочлены Чебышева описаны в ядре системы **Mathematica** и, по аналогии с **BernoulliB**, мы можем ожидать, что они называются **ChebyshevT**. Посмотрим на многочлены Чебышева  $T_5(x)$  и  $T_7(x)$ , чтобы немного уменьшить количество печатаемых символов, введем для **ChebyshevT[x,n]** сокращение **T[x,n]**:

```
In[22]:=T[n_,x_]:=ChebyshevT[n,x]; {T[5,x],T[7,x]}
```

```
Out[22]={5*x-20*x^3+16*x^5,7*x+56*x^3-112*x^5+64*x^7}
```

В этом тексте встречаются несколько ключевых синтаксических моментов, которые подробнейшим образом обсуждаются в дальнейшем, тем не менее сразу поясним их. В первой строчке мы определяем **T[n,x]**, при этом **бланк** — после неизвестной означает, что это **фиктивная неизвестная**, вместо которой можно подставить произвольное значение или любую другую неизвестную. Если вместо **T[n,x]** мы написали бы просто **T[n,x]**, система знала бы, чему равно **T[n,x]**, но не чему равно **T[m,y]**. С другой стороны, отложенное присваивание **:=** означает, что правая часть **ChebyshevT[n,x]** вычисляется не в тот момент, когда выполнено присваивание, а каждый раз при вызове очередного значения **T[n,x]**. После этого мы видим уже знакомую нам точку с запятой **;** разделяющую выражения или команды. После этого мы вызываем **список** **{T[5,x],T[7,x]}** состоящий из многочленов Чебышева  $T_5(x)$  и  $T_7(x)$ . Мы не могли бы напечатать просто **T[5,x]; T[7,x]**,

так как при этом увидели бы только многочлен  $T_7(x)$ . Подставим теперь  $T_7(x)$  в  $T_5(x)$ :

```
In[23]:=T[5,T[7,x]]
Out[23]=5*(-7*x+56*x^3-112*x^5+64*x^7)-
          20*(-7*x+56*x^3-112*x^5+64*x^7)^3+
          16*(-7*x+56*x^3-112*x^5+64*x^7)^5
```

Как всегда, Mathematica не раскрывает скобки! Ну что же, попросим ее сделать это:

```
In[24]:=Expand[T[5,T[7,x]]]
Out[24]=-35*x+7140*x^3-434112*x^5+12403200*x^7-
          202585600*x^9+2106890240*x^11-14910300160*x^13+
          74977509376*x^15-275652608000*x^17+754417664000*x^19-
          1551944908800*x^21+2404594483200*x^23-2789329600512*x^25+
          2384042393600*x^27-1456262348800*x^29+601295421440*x^31-
          150323855360*x^33+17179869184*x^35
```

Если то, что получилось, удивительно похоже на многочлен Чебышева  $T_{35}(x)$ , то это потому, что это и есть многочлен Чебышева  $T_{35}(x)$  — IT LOOKS LIKE A CHURCH, IT SMELLS LIKE A CHURCH, IT IS A CHURCH. Иными словами, если мы подставим в только что полученное выражение  $\cos(\phi)$  вместо  $x$ , то мы получим  $\cos(35\phi)$ . Теперь Вас уже наверное, не особенно удивит, если и `Expand[T[7,T[5,x]]]` даст тот же результат.

В этом можно убедиться также применив к многочлену  $T[35,x]$  функцию `Decompose`, раскладывающую многочлен в композицию неразложимых многочленов:

```
In[25]:=Decompose[T[35,x],x]
Out[25]={7*x-56*x^3+112*x^5-64*x^7,-5*x+20*x^3-16*x^5}
```

Вообще то, в данном случае нам просто крупно повезло. Попробуйте образовать композицию  $T_2$  и  $T_3$ , а потом разложить ее при помощи `Decompose` и посмотрите, что получится!

Вообще,  $T_m(T_n(x)) = T_n(T_m(x))$  причем в смысле, который легко уточнить<sup>40,41,42,43</sup>, это единственная *нетривиальная* система многочленов над  $\mathbb{C}$  с таким свойством. Только не говорите нам про  $f_n(x) = x^n$ , мы же сказали, *нетривиальная*!

• **Вычисления с рациональными дробями.** Известно, что любое рациональное число представимо в виде суммы трех кубов рациональных

<sup>40</sup>J.F.Ritt, Prime and composite polynomials. — Trans. Amer. Math. Soc., 1922, vol.23, p.51–66.

<sup>41</sup>J.F.Ritt, Permutable rational functions. — Trans. Amer. Math. Soc., 1923, vol.25, p.399–448.

<sup>42</sup>В.О.Бугаенко, Коммутирующие многочлены. — Математическое Просвещение, Сер.3, N.1, с.140–163.

<sup>43</sup>В.В.Прасолов, О.В.Шварцман, Алгебра римановых поверхностей. — М., Фазис, 1999, с.1–142; стр.75–85.



чисел. Это вытекает, например, из следующего тождества, независимо открытого в 1825 году Райли и в 1930 году Ричмондом<sup>44</sup>:

$$x = \left( \frac{x^3 - 3^6}{3^2 x^2 + 3^4 x + 3^6} \right)^3 + \left( \frac{-x^3 + 3^5 x + 3^6}{3^2 x^2 + 3^4 x + 3^6} \right)^3 + \left( \frac{3^3 x^2 + 3^5 x}{3^2 x^2 + 3^4 x + 3^6} \right)^3$$

Попытка доказать это тождество просто напечатав

```
In[26]:=((x^3-3^6)/(3^2x^2+3^4x+3^6))^3+
          ((-x^3+3^5x+3^6)/(3^2x^2+3^4x+3^6))^3+
          ((3^3x^2+3^5x)/(3^2x^2+3^4x+3^6))^3
```

не приведет к успеху, так как Mathematica автоматически не раскрывает скобок — в Главе 5 мы объясняем, почему ни одна разумная система компьютерной алгебры этого не делает! Однако в Mathematica имеется несколько функций таких, как Expand, ExpandNumerator, ExpandDenominator, ExpandAll, Factor, Cancel, Together, Apart и т.д., которые позволяют проделывать все обычные структурные манипуляции над дробями. Попробовав, например, привести эти дроби к общему знаменателю

```
In[27]:=Together[((x^3-3^6)/(3^2x^2+3^4x+3^6))^3+
                  ((-x^3+3^5x+3^6)/(3^2x^2+3^4x+3^6))^3+
                  ((3^3x^2+3^5x)/(3^2x^2+3^4x+3^6))^3]
```

мы сразу получим ответ  $x$ . Разумеется, к тому же результату приведет и упрощение этого выражения при помощи Simplify, FullSimplify или Refine. Интересно, что хотя это вычисление и может быть проведено человеком — оно и было первые два раза проведено человеком!!! — тем не менее, вряд ли многие математики захотят проводить подобное вычисление по своей воле без какой-то великой цели. Косвенным подтверждением этого является тот факт, что в книге<sup>45</sup>, специально посвященной суммам кубов, эта формула приведена с ошибкой (в последнем слагаемом пропущен множитель  $3^3$  при  $x^2$ , а в качестве множителя при  $x$  там же напечатано  $3^4$  вместо  $3^5$ ), причем эта ошибка воспроизведена в обзоре<sup>46</sup>. Тем самым, авторы сами эту формулу не проверяли, что, впрочем, неудивительно, если учесть, что при этом вычислении получаются коэффициенты наподобие 387420489, 129140163 или 28697814.

### § 3. РЕШЕНИЕ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Alles vergangliches ist nur ein Gleichnis. — Все преходящее есть всего лишь уравнение<sup>47</sup>.

<sup>44</sup>L.E.Dickson, History of the theory of numbers, vol.II. — Chelsea, 1952, p.1–802.

<sup>45</sup>Ю.И.Манин, Кубические формы. — М., Наука, 1972, с.1–304.

<sup>46</sup>Ю.И.Манин, А.А.Панчишкин, Введение в теорию чисел. — Соврем. Проблема Математики, т.49, М., ВИНТИ, 1990, с.1–348; стр.191.

<sup>47</sup>В кольце  $\mathbb{Z}/m\mathbb{Z}$ . Как всегда, канонический русский перевод “Все преходящее есть только символ” полностью извращает смысл сказанного. На самом деле Gleichnis может быть чем угодно: сравнением, подобием, несовершенной тенью, иносказанием — но никак не символом. Наоборот, символ есть как то, как бы проекцией чего является Gleichnis.

Johann Wolfgang von Goethe, Faust I

Решение уравнений и неравенств, а также систем уравнений и неравенств является *sanctum sanctorum* школьной математики. *Mathematica* может успешно справиться с **любой** задачей, с которой могут справиться школьник, школьный учитель математики, преподаватель педвуза и любой репетитор над репетиторами.

• **Алгебраические уравнения от одной неизвестной.** Основной командой для решения алгебраических и сводящихся к ним уравнений в системе *Mathematica* является *Solve*. Эта команда допускает вызов в разных форматах и настройку большого числа опций, а ее реализация занимает около 500 страниц кода, поэтому здесь мы изложим только простейшие примеры ее использования. В случае решения одного уравнения  $f(x) = g(x)$  относительно одной неизвестной  $x$  команда *Solve* вызывается в следующем формате

```
Solve[f[x]==g[x],x].
```

Вот простейший пример использования этой команды.

```
In[28]:=Solve[Sqrt[x^2+1]==x+2,x]
```

```
Out[28]={ {x->-3/4} }
```

Как всегда, начнем с обсуждения синтаксиса:

о Обратите внимание на то, что ПРИ ЗАПИСИ УРАВНЕНИЯ ВСЕГДА ИСПОЛЬЗУЕТСЯ ТОЛЬКО ПРЕДИКАТ `== Equal`, но ни в коем случае не предикат `=== SameQ` и, тем более, не оператор `= Set!!!` Вычисление

```
In[29]:=Solve[Sqrt[x^2+1]==x+2,x]
```

даст ответ `{}`. В самом деле, спрашивая `Solve[f[x]==g[x],x]` мы интересуемся, при каких  $x$  значения функций  $f$  и  $g$  в точке  $x$  совпадают. В то же время спрашивая `Solve[f[x]==g[x],x]` мы интересуемся, при каких  $x$  внутренние представления функций  $f$  и  $g$  совпадают в языке системы. Ну и при каких  $x$ , по Вашему `Sqrt[x^2+1]` и `x+2` могут совпадать как выражения? Да ни при каких! С другой стороны, попытка вычислить `Solve[Sqrt[x^2+1]=x+2,x]` сразу приведет к сообщению об ошибке, так как здесь мы пытаемся присвоить новое значение внутренней (защищенной!!!) функции `Sqrt`. Итак, уравнение всегда записывается в виде `f[x]==g[x]`.

о В школьной математике существует устойчивая традиция обозначать **параметры** буквами  $a, b, c, d, \dots$ , а **неизвестные** буквами  $x, y, z, u, v, w$ . Однако было бы нелепо ожидать, чтобы система *Mathematica* использовала подобное соглашение — СИМВОЛ ЕСТЬ СИМВОЛ, ЕСТЬ СИМВОЛ, ХОТЬ СИМВОЛОМ ЕГО НАЗОВИ, ХОТЬ КАК. Это значит, что если уравнение содержит *единственный* символ, то этот символ и будет с необходимостью истолкован как неизвестная, так что в этом случае команду *Solve* можно вызывать с одним аргументом, в формате `Solve[f[x]==f[y]]`:

```
In[30]:=Solve[x^3+3*x^2-3*x+1==0]
```

```
Out[30]={ {x->-1-2^(1/3)-2^(2/3)},
```

$$\{x \rightarrow -1 + (1 - I \sqrt{3}) / 2^{(1/3)} + (1 + I \sqrt{3}) / 2^{(2/3)}\},$$

$$\{x \rightarrow -1 + (1 - I \sqrt{3}) / 2^{(2/3)} + (1 + I \sqrt{3}) / 2^{(1/3)}\}$$

о В то же время, если уравнение содержит несколько символов, то указание на то, который из них рассматривается как переменная, абсолютно необходимо. Если это не сделать, то система будет решать уравнение относительно того символа, который — с ее точки зрения — идет первым в *записи* этого уравнения. Это значит, что (в зависимости от истории сессии и настроек ядра) в качестве неизвестной в уравнении  $ax^2 + bx + c = 0$  обычно будет рассматриваться  $a$  или  $c$ , но отнюдь не  $x$ ! При попытке провести вычисление

```
In[31]:=Solve[a*x^2+b*x+c==0]
```

результатом будет сообщение об возможной ошибке:

```
Solve::svars: Equations may not give solutions for all
"solve" variables
```

сообщающее нам, что мы пытаемся найти *четыре* неизвестных из единственного уравнения, и следующий аутпут:

```
Out[31]={ {c->-b*x-a*x^2} }
```

Таким образом, чтобы решить уравнение  $ax^2 + bx + c = 0$  относительно  $x$ , нужно задавать системе вопрос в форме `Solve[a*x^2+b*x+c==0,x]`, где  $x$  явно декларируется в качестве неизвестной.

о Кроме того, обратите внимание на формат ответа! Найденные корни выражаются в форме **правил подстановки**  $x \rightarrow c$ , с тем, чтобы их можно было подставлять в другие выражения, не модифицируя при этом значения самого  $x$  (которое остается независимой переменной).

По этому поводу стоит заметить, что в *Mathematica* имеется еще одна команда решения алгебраических уравнений *от одной неизвестной*, а именно **Roots**, которая выражает *набор* корней уравнения как объединение наборов корней более простых уравнений. При этом ответ записывается как дизъюнкция этих более простых уравнений — и, тем самым, в большинстве случаев, которые могут реально встретиться начинающему, как дизъюнкция линейных уравнений. Использование команды **Roots** по своему синтаксису ничем не отличается от использования команды **Solve** для *одной неизвестной*:

```
Roots[f[x]==g[x],x].
```

Следующий пример иллюстрирует использование команды **Roots** для решения алгебраического уравнения:

```
In[32]:=Roots[x^2+x+1/x+1/x^2==4,x]
```

```
Out[32]=x==1/2(-3-Sqrt[5])||x==1/2(-3+Sqrt[5])||x==1||x==1
```

Чтобы перевести этот ответ в форму списка правил подстановки, порождаемую командой **Solve**, необходимо применить к нему форматную команду **ToRules**. Несмотря на более привычную форму ответа, возвращаемого командой **Roots**, при решении уравнений *в вещественных или комплексных*

*числа* в большинстве ситуаций предпочтительно пользоваться более общими командами `Solve`, при помощи которой можно решать уравнения от нескольких неизвестных, и `Reduce`, решающей или упрощающей в том же стиле, что `Roots`, любые уравнения, как алгебраические, так и трансцендентные, а также неравенства, логические суждения и т.д. С другой стороны, команда `Roots` использует чисто алгебраические алгоритмы (те же, что `Factor` и `Decompose`) и оказывается *незаменимой* во многих ситуациях, когда `Solve` бессильна (конечные поля, модулярная арифметика, вычисления в кольцах и т.д.).

Увидев пустой ответ, не забудьте внимательно пересчитать скобки!!! А именно:

◦ Ответ в форме `{}` означает, что у уравнения нет решений. Например, мы получим такой ответ, предложив системе проделать следующее упражнение: `Solve[1==0,x]`.

◦ Ответ в форме `{{}}` означает, что *любое* допустимое значение аргумента служит решением уравнения. Мы получим такой ответ предложив системе вычислить `Solve[(x-1)(x+1)==x^2-1,x]`.

• **Уравнения степени  $\leq 4$ .** Уравнения степени  $\leq 4$  команды `Solve` и `Roots` решают в школьном стиле — т.е. в радикалах, возвращая формулы в духе Кардано и Феррари. Вот, скажем, как выглядят корни многочлена Тэйлора  $1 + x + x^2/2 + x^3/6$  порядка 3 для экспоненты:

```
In[33]:=Solve[Sum[x^i/i!,{i,0,3}]==0,x]
Out[33]={{x->-1-1/(-1+Sqrt[2])^(1/3)+(-1+Sqrt[2])^(1/3)},
          {x->-1-1/2*(-1+Sqrt[2])^(1/3)*(1-I*Sqrt[3])
           +(1+I*Sqrt[3])/(2*(-1+Sqrt[2])^(1/3))},
          {x->-1+(1-I*Sqrt[3])/(2*(-1+Sqrt[2])^(1/3))
           -1/2*(-1+Sqrt[2])^(1/3)*(1+I*Sqrt[3])}}
```

В тех случаях, когда системе удастся решить в таком же стиле уравнения более высоких степеней, она делает это. Посмотрим, как система борется с уравнением  $x^5 + x - c = 0$ , которое, вообще говоря, в радикалах не решается. Более того, как хорошо известно, при помощи **преобразования Чирнгаузена** решение любого алгебраического уравнения степени 5 сводится к уравнению такого вида, называемому **уравнением Бринга—Джерарда**<sup>48</sup>. Для того, чтобы чуть упростить вид входящих в решение радикалов, вынести общие множители и пр., поверх команды `Solve` рекомендуется применять команду `Simplify` — но, как будет объяснено ниже, вообще говоря, не команду `FullSimplify`:

```
In[34]:=Simplify[Solve[x^5+x-1==0,x]]
Out[34]={{x->(-1)^(1/3)}, {x->-(-1)^(2/3)},
          {x->1/6*(-2+2^(2/3)*(25-3*Sqrt[69])^(1/3)+
           2^(2/3)*(25+3*Sqrt[69])^(1/3))},
```

<sup>48</sup>В.В.Прасолов, Ю.П.Соловьев, 'Эллиптические кривые и алгебраические уравнения. — М., Факториал, 1997, с.1–288; стр.222–225.

```
{x->-1/3-1/6*(1+I*Sqrt[3])*(1/2*(25-3*Sqrt[69]))^(1/3)
+1/6*I*(1+Sqrt[3])*(1/2*(25+3*Sqrt[69]))^(1/3)},
{x->-1/3+1/6*I*(1+I*Sqrt[3])*(1/2*(25-3*Sqrt[69]))^(1/3)
-1/6*(1+Sqrt[3])*(1/2*(25+3*Sqrt[69]))^(1/3)}
```

Заметим, что формулы с радикалами легко отключить, задав в теле команд опции Cubics->False или Quartics->False (по умолчанию обе эти опции поставлены на True).

Еще одной чрезвычайно интересной и полезной опцией, которую допускает команда Roots при решении алгебраических уравнений с рациональными коэффициентами, является Modulus. Значением модуля может быть любое целое число (по умолчанию Modulus->0). Это значит, что включив в тело команды опцию Modulus->m, мы ищем решение уравнения в кольце  $\mathbb{Z}/m\mathbb{Z}$  классов вычетов по модулю  $m$ .

```
In[35]:=Roots[3*x^2+5==0,x,Modulus->17]
```

```
Out[35]=x==2 | x==15
```

Стоит подчеркнуть, что при помощи команды Solve без довольно деликатной перенастройки проделать подобное упражнение не просто.

• **Корни алгебраических уравнений.** Почему мы не рекомендуем применять команду Full Simplify? Дело в том, что система пытается привести выражение к наиболее простому виду, а никакого *более простого* описания корня  $s$  алгебраического уравнения, чем само это — или какое-то другое алгебраическое уравнение, корнем которого является  $s$  — в общем случае не существует. Иными словами, утверждение, что  $f(c) = 0$  для некоторого многочлена  $f$ , возможно в сочетании с какими-то утверждениями о локализации корня  $s$  и/или изоляции корней, является в подавляющем большинстве случаев *гораздо* лучшим описанием числа  $s$  с вычислительной точки зрения, чем любое другое его описание!!! Например, вычислив Solve[x^5+x-2==0,x], даже после упрощения ответа при помощи Simplify, Вы натолкнетесь на полторы-две страницы радикалов. Но полностью упростив это выражение, Вы увидите следующее:

```
In[36]:=FullSimplify[Solve[x^5+x-2==0,x]]
```

```
Out[36]={{x->1},{x->Root[2+#1+#1^2+#1^3+#1^4&,3]},
{x->Root[2+#1+#1^2+#1^3+#1^4&,4]},
{x->Root[2+#1+#1^2+#1^3+#1^4&,1]},
{x->Root[2+#1+#1^2+#1^3+#1^4&,2]}}
```

Прокомментируем вначале *форму* ответа. Команда Root[f,i] возвращает  $i$ -й корень *алгебраического* уравнения  $f(x) = 0$ . Нам нет нужды сейчас *точно* описывать порядок, в котором Mathematica учитывает корни. В первом приближении можно считать, что она руководствуется следующими правилами:

о Вещественные корни предшествуют комплексным и упорядочиваются естественным образом;

◦ Сопряженные комплексные корни приводятся *парами* и упорядочиваются лексикографически: вначале по вещественной части, а потом по мнимой части того корня из пары, который лежит в верхней полуплоскости.

При этом если все коэффициенты исходного уравнения были числами, полученные при помощи применения команды `Root` объекты тоже рассматриваются как числа, иными словами, к ним можно применять все обычные операции над числами, сравнивать их, вычислять приближенные значения и т.д.

Новым и весьма необычным для начинающих моментом в использовании команды `Root` является то, что многочлен  $f$  задается в формате **чистой** или **анонимной функции**. Вызов  $i$ -го корня многочлена  $f = a_n x^n + \dots + a_1 x + a_0$  может быть произведен в одном из следующих эквивалентных форматов:

- `Root[Function[x, a_n*x^n+...+a_1*x+a_0], i]`
- `Root[a_n*#^n+...+a_1*#+a_0&, i]`

Первый из этих форматов (функциональный, формат чистой функции) рассматривается как основной способ внутреннего представления выражения, включающего объекты типа `Root`, в языке системы. Мы настоятельно рекомендуем начинающему пользоваться именно этим форматом, несмотря на чуть большую длину получающихся при этом текстов.

Второй формат (операторный, формат анонимной функции) рассматривается системой как сокращение первого. Его назначение состоит только в том, чтобы слегка уменьшить длину ввода за счет использования следующих операторов:

- Оператор `#` или, в полной форме `Slot`, обозначает *аргумент* чистой функции, которому мы не хотим присваивать никакого индивидуального имени (отсюда название **анонимная** функция — в этом случае не только сама функция, но и ее аргументы не имеют индивидуальных имен);
- Если у анонимной функции несколько аргументов, то они будут вызываться как `#1`, `#2`, `#3`, и так далее, по мере появления;
- Оператор `&` есть просто сокращение для `Function`, и обозначает *применение* чистой функции.

Мы еще раз встретимся с анонимными функциями при дифференцировании и решении дифференциальных уравнений, но, как нам кажется, для тех кто раньше не имел дела с математической логикой и функциональным программированием, правильное применение этого формата может представлять известные трудности и, поэтому мы отложим более подробное обсуждение всех возникающих здесь нюансов до Выпусков 2 и 3.

Вернемся теперь к *смыслу* последнего результата. Он означает, что система уверена в том, что никакого “более простого” описания корней  $c \neq 1$  уравнения  $x^5 + x - 1 = 0$ , чем как корни уравнения  $x^4 + x^3 + x^2 + x + 2 = 0$ , *не существует*. Иными словами, она предпочитает использовать именно это описание (а не полустраничное выражение каждого корня в виде комбинации радикалов) в вычислениях и советует нам делать то же самое.

Во многих случаях она с самого начала считает, что у каких-то чисел нет вообще никакого более простого описания, чем как корни того уравнения, которое мы пытаемся решить! В этом случае она выражает корни этого уравнения как корни этого уравнения:

```
In[37]:=FullSimplify[Solve[x^5+x-3==0,x]]
Out[37]={{x->Root[-3+#1+#1^5&,1]}, {x->Root[-3+#1+#1^5&,2]},
          {x->Root[-3+#1+#1^5&,3]}, {x->Root[-3+#1+#1^5&,4]},
          {x->Root[-3+#1+#1^5&,5]}}
```

Что, конечно, ничуть не мешает ей знать приближенные численные значения этих корней и использовать их в других вычислениях по нашей просьбе:

```
In[38]:=Table[N[Root[-3+#1+#1^5&,i]],{i,1,5}]
Out[38]={1.133,-1.04188-0.82287*I,-1.04188+0.82287*I,
          0.475381-1.1297*I, 0.475381+1.1297*I}
```

С другой стороны, мы можем всегда (когда это возможно!) заставить систему записать формулу в радикалах, применив к объектам типа `Root` команду `ToRadicals`. Более того, система сама делает это для уравнений степени  $\leq 2$ :

```
In[39]:=f:=Function[x,a*x^2+b*x+c]; {Root[f,1],Root[f,2]}
Out[39]={-b/(2*a)-1/2*Sqrt[(b^2-4*a*c)/a^2],
          -b/(2*a)+1/2*Sqrt[(b^2-4*a*c)/a^2]}
```

Для вычислений с алгебраическими числами чрезвычайно полезна команда `RootReduce`, которая пытается переписать сложное выражение, содержащее корни уравнения, в терминах *единственного* корня этого (или какого-то другого!) уравнения.

• **Что значит решить уравнение?** Тому, кто не задумывался над тем, что значит **решить уравнение**, этот вопрос, скорее всего, покажется чисто схоластическим. Между тем, для каждого, кто *реально* проводит вычисления, этот вопрос становится абсолютно конкретным и неотвратимым.

В самом деле, с младших классов школы нас приучили никуда особенно не вникая писать, что корни уравнения  $x^2 - 2 = 0$  равны  $x = \pm\sqrt{2}$  и называть это **решением уравнения**. Да, но что такое  $\sqrt{2}$ ? Ведь если вернуться к истокам, то  $\sqrt{2}$  как раз и *определяется* как (единственный) положительный корень уравнения  $x^2 - 2 = 0$ . Да, но где живет этот корень? Коэффициенты нашего исходного уравнения рациональны, но  $\sqrt{2}$  не является рациональным числом. Любое фактическое вычисление с  $\sqrt{2}$  либо является приближенным, либо сводится к тому, что мы возводим его в квадрат и проводим вычисление с рациональными числами!! Тем самым, любое точное вычисление с  $\sqrt{2}$  основано непосредственно на том факте, что  $\sqrt{2}^2 - 2 = 0$ , т.е. на том, что  $\sqrt{2}$  является корнем уравнения  $x^2 - 2 = 0$ . Никакого более простого описания у числа  $\sqrt{2}$  нет. Вся школьная премудрость о решении квадратного уравнения  $x^2 - c = 0$  сводится к тому, что мы заявляем, что корнем этого уравнения является корень уравнения  $x^2 - c = 0$ . Иными словами, под влиянием школьной математики нам *мнится*, что мы умеем решать квадратные уравнения и *не умеем* решать уравнений пятой степени, ну, хотя бы,  $x^5 + x - 3 = 0$ .

Однако с нашей точки зрения статус уравнений  $x^2 - c = 0$  и  $x^5 + x - c = 0$  абсолютно одинаков и никакой концептуальной разницы между ними нет. Для каждого из них — да и вообще для любого алгебраического уравнения разумной степени — мы можем за доли секунды приближенно найти корни с любой точностью. В то же время в смысле арифметики ни одно из этих уравнений — без введения дополнительных функций — точно

не решается. С алгоритмической точки зрения вычисление значения квадратного корня основано на разложении в ряд и ничем не отличается от вычисления значения любой другой аналитической функции. Единственное отличие носит чисто психологический характер и состоит в том, что мы ввели специальный значок  $\sqrt{c}$  для положительного корня первого из этих уравнений

$$\sqrt{c} = 1 + \frac{1}{2}(c-1) - \frac{1}{8}(c-1)^2 + \frac{1}{16}(c-1)^3 - \frac{5}{128}(c-1)^4 + \dots$$

и долго учились им манипулировать. Если бы мы ввели специальный значок, ну хотя бы  $\P(c)$ , для **функции Ламберта—Эйзенштейна**, выражающей положительный корень второго из этих уравнений

$$\P(c) = c - c^5 + 10 \frac{c^9}{2!} - 15 \cdot 14 \frac{c^{13}}{3!} + 20 \cdot 19 \cdot 18 \frac{c^{17}}{4!} - \dots$$

и в течение такого же времени упражнялись в манипуляциях с этим значком<sup>49,50</sup>, а потом с преобразованием Чирнгаузена, позволяющим свести решение любого уравнений пятой степени к последовательности арифметических операций и применения  $\sqrt{\phantom{x}}$ ,  $\sqrt[3]{\phantom{x}}$  и  $\P$ , то у нас сформировалось бы отчетливое — и, в целом, вполне обоснованное — убеждение, что мы *умеем* решать уравнения пятой степени.

Тот же вопрос, но, конечно, с еще большей остротой, встает при попытке решения трансцендентных уравнений. Что такое  $\pi$ ? По определению это наименьший положительный корень уравнения  $\sin(x) = 0$ . Пользуясь соотношениями между основными тригонометрическими функциями и теоремами сложения несложно доказать, что тогда  $\pi$  будет и корнем уравнений  $\cos(x) = -1$ ,  $\cos(x/2) = 0$ ,  $\sin(x/2) = 1$  и тому подобные факты. Но все равно в конце дня (in the final analysis, at the end of the day) выясняется, что  $\pi$  никогда не было ничем, кроме наименьшего положительного корня уравнения  $\sin(x) = 0$ . В своем знаменитом учебнике<sup>51</sup> Эдмунд Ландау *определяет*  $\pi$  как наименьшее положительное число такое, что  $\cos(\pi/2) = 0$  (определение 61) и *доказывает*, что тогда  $\sin(\pi) = 0$  (теорема 267), но мы же слышали о логической эквивалентности?

Из сказанного выше ясно, что в общем случае трансцендентные уравнения не решаются. У нас просто нет такого количества букв, чтобы присвоить индивидуальное имя каждому корню всех встречающихся в природе трансцендентных уравнений. Более того, в большинстве случаев очень трудно даже связать решения различных трансцендентных уравнений или вообще доказать какое-то индивидуальное суждение об этих решениях. Самый знаменитый пример, когда мы не в состоянии решить простейшее трансцендентное уравнение — это **гипотеза Римана**. При  $\operatorname{re}(s) > 1$  определим **дзета-функцию Римана** сходящимся рядом

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Эта функция допускает аналитическое продолжение на всю комплексную плоскость, с полюсом в  $s = 1$  (гармонический ряд расходится!) Так вот, многие факты в теории чисел зависят от того, что все нули дзета-функции в критической полосе  $0 < \operatorname{re}(s) < 1$  расположены на прямой  $\operatorname{re}(s) = 1/2$ . Однако на протяжении полутора веков решение этой проблемы — которую Давид Гильберт называл главной задачей не только математики, но и всей жизни! — ускользает от всех усилий специалистов. Иными словами, мы не можем решить уравнение  $\zeta(s) = 0$  даже при дополнительных предположениях о локализации корней!

<sup>49</sup>S.J.Patterson, Eisenstein and the quintic equation. — Historia Math, vol.17, 1990, p.132–140.

<sup>50</sup>J.Stillwell, Eisenstein's footnote. — Math. Intelligencer, v.17, N.2, 1995, p.58–62.

<sup>51</sup>Э.Ландау, Введение в дифференциальное и интегральное исчисление. — 1948, ИЛ, М., с.1–458.



• **Решение трансцендентных уравнений.** Команда `Solve` может служить и для решения трансцендентных уравнений, в тех случаях, когда система может легко сделать это известными ей методами. Однако в действительности правильной командой для решения трансцендентных уравнений в подавляющем большинстве случаев является `Reduce`, а отнюдь не `Solve`!! Разумеется, так как мы *не умеем* решать трансцендентных уравнений, то даже `Reduce` не может в общем случае дать правильного ответа. Однако применение `Reduce` по крайней мере гарантирует от получения неправильного или неполного ответа. Сравним эти команды на конкретных примерах. Скажем, попытка провести вычисление

```
In[40]:=Solve[Exp[2*x]+2*Exp[x]+1==x,x]
```

приводит к следующему сообщению об ошибке:

```
Solve::ifun: The equations appear to involve the variables
              to be solved for in an essentially
              non-algebraic way.
```

В результате система просто оставляет наше исходное выражение неизвлеченным. К сожалению, в данном случае и применение `Reduce` не приводит к большому успеху, в ответ на попытку вычислить

```
In[40]:=Reduce[Exp[2*x]+2*Exp[x]+1==x,x]
```

мы получаем такое оптимистическое сообщение:

```
Reduce::nsmet: This system cannot be solved with the methods
              available to Reduce.
```

Так что не нарисовав графики левой и правой части при помощи команды `Plot`, Вы так и не сможете узнать, имеет это уравнение вещественные корни, или нет. С другой стороны при попытке вычислить

```
In[40]:=Solve[Sqrt[Log[x]]==Log[Sqrt[x]],x]
```

система выдает сообщение о *возможной* ошибке:

```
Solve::ifun: Inverse functions are being used by Solve,
              so some solutions may not be found;
              use Reduce for complete solution information.
```

Однако в данном случае, несмотря на свою озабоченность, система возвращает правильный ответ:

```
Out[40]={ {x->1}, {x->E^4} }
```

В то же время в выпуске 2 нам встречаются примеры, когда система действительно находит лишь *часть* решений, а не все решения, так что обычно к подобному предупреждению следует относиться с полной серьезностью. Вот еще один похожий пример, в котором мы игнорируем `error message` и в результате получаем лишь частичный ответ:

```
In[41]:=Solve[Sin[x]==Cos[x],x]
```

```
Out[41]={ {x->-3*Pi/4}, {x->Pi/4} }
```

Понятно, что это не совсем то, что мы хотели. В то же время применение здесь команды `Reduce` дает уже абсолютно осмысленный ответ:

```
In[42]:=Reduce[Sin[x]==Cos[x],x]
Out[42]=C[1]∈Integers&&(x== -2*ArcTan[1+Sqrt[2]]+2*Pi*C[1]||
x== -2*ArcTan[1-Sqrt[2]]+2*Pi*C[1])
```

Дальнейшие примеры трансцендентных уравнений рассмотрены в § 5.

• **Численное решение уравнений.** Основной командой для численного решения алгебраических уравнений является команда `NSolve`. Дело в том, что точное решение алгебраических уравнений с последующим округлением может приводить к чрезвычайно забавным эффектам. В качестве типичного примера можно рассмотреть характеристическое уравнение кубической матрицы золотого сечения  $1 - 5x + 6x^2 - x^3 = 0$ . Решая это уравнение при помощи формулы Кардано, мы получим точные корни:

```
In[43]:=Solve[1-5*x+6*x^2-x^3==0,x]
Out[43]={ {x->2+7^(2/3)/(3/2*(9+I*Sqrt[3]))^(1/3)+
(7/2*(9+I*Sqrt[3]))^(1/3)/3^(2/3)},
{x->2-(7/2)^(2/3)*(1+I*Sqrt[3])/(3*(9+I*Sqrt[3]))^(1/3)-
(1-I*Sqrt[3])*(7/2*(9+I*Sqrt[3]))^(1/3)/2*3^(2/3)},
{x->2-(7/2)^(2/3)*(1-I*Sqrt[3])/(3*(9+I*Sqrt[3]))^(1/3)-
(1+I*Sqrt[3])*(7/2*(9+I*Sqrt[3]))^(1/3)/2*3^(2/3)},
```

Как было известно итальянским математикам XVI века, все мнимости в этих формулах сокращаются, так что все три корня вещественные (Кардано назвал это явление *casus irreducibilis*). Однако попытка найти приближенные значения корней приводит к следующему удивительному ответу, еще раз иллюстрирующему полную абсурдность применения численных методов к задачам с точными условиями:

```
In[44]:=Map[N,Solve[1-5*x+6*x^2-x^3==0,x]]
Out[44]={ {x->5.04892+2.77556*10^-17*I},
{x->0.307979+2.22045*10^-16*I},
{x->0.643104-2.22045*10^-16*I}}
```

Понятно, что здесь происходит? Система находит значения  $9 + i\sqrt{3}$  и т.д. с машинной точностью (около 16 знаков после запятой), после чего начинает манипулировать с этими приближенными значениями. Но при приближенных вычислениях неизбежно возникают артефакты наподобие 2.77556, 2.22045 и т.д. В подобных случаях изначальная трактовка условий как приближенных приводит к лучшим результатам:

```
In[45]:=NSolve[1-5*x+6*x^2-x^3==0,x,20]]
Out[45]={ {x->0.3079785283699041304},
{x->0.6431041321077905561}, {x->5.048917339522305314}}
```

#### § 4. СИСТЕМЫ УРАВНЕНИЙ И НЕРАВЕНСТВ

Nowadays we can do computer experiments using *Mathematica*, and even solve a system of 42 equations. This offers another route to knowledge, rather than mere ideas.

John F. Nash, Jr.

Конечно, преимущества системы **Mathematica** становятся полностью понятны только в тот момент, когда нам нужно решить систему 42 уравнений от 47 неизвестных. Для наглядности и из типографских соображений мы проиллюстрируем способности системы на чисто учебных примерах систем от *трех* неизвестных, но она сравнительно легко решает в реальном времени системы от десятка неизвестных, а, если дать ей немного подумать — то и от нескольких десятков неизвестных (на UNIX'овских рабочих станциях, большинство бытовых компьютеров, работающих под Windows, не обладают необходимой для этого памятью). В действительности в этом отношении **Mathematica** уступает только самым продвинутым *специализированным* системам полиномиальных вычислений, вроде **Singular**, которые, с другой стороны, не умеют делать абсолютно ничего, *кроме* полиномиальных преобразований и решения систем алгебраических уравнений. Решение систем алгебраических уравнений от нескольких сотен неизвестных сегодня все еще рассматривается как *очень* трудная задача для систем компьютерной алгебры.

• **Решение систем алгебраических уравнений.** Для решения системы алгебраических уравнений

$$f_1(x, y, z) = g_1(x, y, z), \dots, f_n(x, y, z) = g_n(x, y, z)$$

команда **Solve** вызывается в одном из следующих основных форматов:

◦ Как функция *двух* аргументов, первым из которых является *список* уравнений, а вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему:

`Solve[{f1[x,y,z]==g1[x,y,z],...,fn[x,y,z]==gn[x,y,z]},{x,y}]`

остальные неизвестные при этом рассматриваются как **параметры**.

◦ Как функция *двух* аргументов, первым из которых является *конъюнкция* уравнений, а вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему:

`Solve[f1[x,y,z]==g1[x,y,z]&&...&&fn[x,y,z]==gn[x,y,z]},{x,y}]`

◦ Как функцию *двух* аргументов, первым из которых является *векторное* уравнение,

$$(f_1(x, y, z), \dots, f_n(x, y, z)) = (g_1(x, y, z), \dots, g_n(x, y, z)),$$

а вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему:

`Solve[{f1[x,y,z],...,fn[x,y,z]}=={g1[x,y,z],...,gn[x,y,z]},{x,y}]`

◦ Как функция *одного* аргумента, который при этом оформляется либо как список, либо как конъюнкция уравнений, либо, наконец, как векторное уравнение:

```
Solve[{f1[x,y,z]==g1[x,y,z],...,fn[x,y,z]==gn[x,y,z]}]
Solve[f1[x,y,z]==g1[x,y,z]&&...&&fn[x,y,z]==gn[x,y,z]]
Solve[{f1[x,y,z],...,fn[x,y,z]}=={g1[x,y,z],...,gn[x,y,z]}]
```

В этом случае система из всех сил пытается найти значения **всех** входящих в эти уравнения символов, что, вообще говоря, не всегда получается, если количество уравнений меньше, чем количество неизвестных и параметров.

о Как функция *трех* аргументов, первым из которых является система уравнений (представленная в любой из трех описанных выше эквивалентных форм), вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему, а третьим — та неизвестная или список тех неизвестных, которые мы при этом при этом полностью **элиминировать** (= исключить) из ответа, как в качестве неизвестных, так и в качестве параметров:

```
Solve[f1[x,y,...]==g1[x,y,...]&&...&&fn[x,y,...]==gn[x,y,...],
      {x,y},{u,v}]
```

Поскольку понять (а тем более объяснить!!) не только все детали, но даже азы того, что происходит при вызове команды **Solve** с тремя аргументами, довольно трудно, мы настоятельно рекомендуем начинающему использовать команду **Eliminate** для исключения неизвестных, а потом уже решать получающуюся систему обычным образом.

• **Enough, or too little.** Проиллюстрируем решение систем алгебраических уравнений на простейших примерах. Заметим, что в связи с используемой процедурой элиминации неизвестных система может несколько раз порождать одно и то же решение, поэтому, если нас — в школьном духе — интересует *множество* решений, для сокращения ответа мы обычно применяем поверх команды **Solve** команду **Union**.

Мы уже видели, что уже в случае одного уравнения **Mathematica** не знает, какие из входящих в него символов следует рассматривать как неизвестные, а какие как параметры. Для правильного решения *систем* уравнений эта проблема становится абсолютно критической. Как всегда, **Mathematica** может посчитать некоторые некоторые параметры неизвестными — но это сравнительно мелкая неприятность (или, на компьютерном языке, **small beer**). В действительности, здесь может произойти *значительно* более серьезное несчастье: **Mathematica** может решить, что некоторые неизвестные являются параметрами!!! А это, как правило, приводит к неверному ответу, причем получающиеся ошибки очень трудно отслеживаются (особенно если решение уравнений бесконтрольно вызывается в составе другого вычисления). Например, если Вы постараетесь решить систему

$$x - y = z^2, \quad x^3 = 1, \quad y^3 = 1$$

относительно  $x$  и  $y$  посредством беззастенчивого

```
In[46]:=Solve[{x-y==z^2,x^3==1,y^3==1},{x,y}]
```

то ответом будет оглушительное {}, хотя совершенно ясно, что эта система имеет решение, ну хотя бы  $(x, y, z) = (1, 1, 0)$ . Ясно, что произошло? Дело в том, что у этой системы нет *общих* решений, в которых  $z$  могло бы выступать в качестве параметра. Для любых допустимых значений  $x$  и  $y$  мы получаем явное уравнение на  $z$ . А попросив разрешить систему уравнений относительно  $x$  и  $y$  мы предложили Mathematica найти такие решения, в которых  $z$  выступает в качестве параметра!!! Это значит, что мы окажемся в гораздо лучшем положении, если попросим больше. И действительно, вычисление

```
In[47]:=Union[Solve[{x-y==z^2,x^3==1,y^3==1},{x,y,z}]]
```

дает все 15 решений системы. Итак, решая систему уравнений, всегда задавайте себе вопрос, что Вас интересует: нахождение *всех* решений системы или нахождение *общих* решений, в которых какие-то неизвестные выступают в качестве параметров?

**Мораль:** Чтобы избежать грубых ошибок, ВСЕГДА ПРОСИТЕ У СИСТЕМЫ БОЛЬШЕ, ЧЕМ ХОЧЕТСЯ.

• **Примеры решения систем.** Приведем несколько чисто учебных примеров решения систем алгебраических уравнений.

◦ В первом примере мы пытаемся решить систему<sup>52</sup>

$$x^2 + y + z = 1, \quad x + y^2 + z = 1, \quad x + y + z^2 = 1$$

относительно всех входящих в нее неизвестных:

```
In[48]:=Union[Solve[{x^2+y+z==1,x+y^2+z==1,x+y+z^2==1}]]
```

```
Out[48]={ {x->0,y->0,z->1}, {x->0,y->1,z->0}, {x->1,y->0,z->0},
  {x->-1-Sqrt[2],y->-1-Sqrt[2],z->-1-Sqrt[2]},
  {x->-1+Sqrt[2],y->-1+Sqrt[2],z->-1+Sqrt[2]} }
```

◦ Решение уже простейших систем, зависящих от параметров, представляет собой довольно серьезную задачу. Дело в том, что при последовательном исключении неизвестных степень уравнений относительно остающихся неизвестных быстро растет. Например, исключая неизвестные  $y$  и  $z$  из системы

$$x^2 + y + z = a, \quad x + y^2 + z = b, \quad x + y + z^2 = c$$

мы получаем уравнение степени 8 относительно  $x$ , коэффициенты которого довольно тягостным образом выражаются через  $a, b, c$ . Это значит, что попытавшись решить систему относительно уравнений  $x, y, z$  беззастенчивым

```
In[49]:=Solve[{x^2+y+z==a,x+y^2+z==b,x+y+z^2==c},{x,y,z}]
```

<sup>52</sup>Д.Кокс, Дж.Литтл, Д.О'Ши, Идеалы, многообразия и алгоритмы. — Москва, Мир, 2000, с.1–687; стр.151–152.

Вы *конечно*, увидите на экране решение. Вот только что именно Вы собираетесь делать с таким решением, координаты которого выражены в терминах объектов типа `Root` от многочлена степени 8, каждый из коэффициентов которого в свою очередь является многочленом изрядной степени от  $a, b$  и  $c$ ? Такой ответ легко может занять сотни или тысячи строк на экране. Поэтому в тех случаях, когда мы не знаем, что произойдет, мы часто вначале интересуемся не самим ответом, а *длиной* ответа, в данном случае *количеством* решений:

```
In[50]:=Length[Union[Solve[{x^2+y+z==a,x+y^2+z==b,x+y+z^2==c},
                           {x,y,z}]]]
```

Ну в данном-то случае у нашей системы 8 решений. А если их несколько десятков или несколько сотен? В этом случае мы обычно смотрим на *вид* решений посредством команды `Short`:

```
In[51]:=Short[Union[Solve[{x^2+y+z==a,x+y^2+z==b,x+y+z^2==c},
                           {x,y,z}]],10]
```

Вызванная в формате

```
Short[expression,d]
```

команда `Short` приводит к тому, что на экране отображается не все выражение `expression`, которое может занимать несколько сотен страниц и форматирование которого может требовать весьма значительного времени, а только его часть, *приблизительно*  $d$  строчек. *Приблизительно* потому, что система все же пытается вывести осмысленную часть выражения, по которой можно составить хотя бы самое общее впечатление о виде всего остального.

В общем случае (например, если количество уравнений меньше, чем количество неизвестных) решить систему относительно всех неизвестных не удастся, в этом случае следует явно указывать те неизвестные, которые мы хотим выразить через остальные.

◦ В следующей задаче мы просим `Mathematica` выразить в системе

$$x + y + z = 1, \quad x^2 + y^2 + z^2 = 1,$$

неизвестные  $x, y$  через неизвестную  $z$ , которая в этом случае трактуется как параметр:

```
In[52]:=Solve[{x+y+z==1,x^2+y^2+z^2==1},{x,y}]
Out[52]={ {x->1/2*(1-z-Sqrt[1+2*z-3*z^2]),
           y->1/2*(1-z+Sqrt[1+2*z-3*z^2])},
          {x->1/2*(1-z+Sqrt[1+2*z-3*z^2]),
           y->1/2*(1-z-Sqrt[1+2*z-3*z^2])} }
```

Если попытаться решить систему относительно всех трех неизвестных, скажем, так:

```
In[53]:=Solve[{x+y+z==1,x^2+y^2+z^2==1},{x,y,z}]
```

то результатом будет уже знакомое нам сообщение об ошибке:

Solve::svars: Equations may not give solutions for all  
"solve" variables.

Тем не менее, Mathematica снова вернет те же самые формулы, что и в предыдущем случае. Кстати, как Вы думаете, почему она и в этом случае выражает  $x$  и  $y$  через  $z$ ? Ну это, как раз, совершенно понятно. Она пытается решить систему в первую очередь относительно тех переменных, которые перечислены в списке первыми. В ответ на

```
In[54]:=Solve[{x+y+z==1,x^2+y^2+z^2==1},{z,y,x}]
```

она выдаст то же сообщение об ошибке и выразит  $z$  и  $y$  (в таком порядке!) через  $x$ .

• **Исключение неизвестных.** В тех случаях, когда уравнений недостаточно, чтобы найти все неизвестные, часто полезно просто свести исходную систему уравнений к системе от меньшего количества неизвестных. В случае алгебраических уравнений основной командой для этого в языке Mathematica является `Eliminate`. Для исключения неизвестной  $z$  из системы алгебраических уравнений

$$f_1(x, y, z) = g_1(x, y, z), \dots, f_n(x, y, z) = g_n(x, y, z)$$

команда `Eliminate` вызывается в формате

```
Eliminate[{f1[x,y,z]==g1[x,y,z],...,fn[x,y,z]==gn[x,y,z]},z]
```

функции двух аргументов, первым из которых является *список* уравнений, а вторым — исключаемая неизвестная (или *список* неизвестных, если их несколько). Для получения более симметричного ответа, как всегда, рекомендуется применять поверх команды `Eliminate` команду `Simplify`. Вот пара простейших примеров:

```
In[55]:=Simplify[Eliminate[{x+y+z==1,x*y*z==1},y]]
```

```
Out[55]=x*z*(-1+x+z)==-1
```

```
In[56]:=Simplify[Eliminate[{x+y+z==1,x^2+y^2+z^2==1},y]]
```

```
Out[56]=x^2+x*z+z^2==x+z
```

Если нам нужно одновременно исключить несколько неизвестных, то эти неизвестные тоже должны задаваться в виде списка. Вот пример, где мы ищем соотношение между суммами степеней, для чего требуется исключить сразу две неизвестных:

```
In[57]:=Simplify[Eliminate[{u==x^5+y^5,v==x^3+y^3,w==x^2+y^2},
                                                                    {x,y}]]
```

```
Out[57]=2*u^3+30*u*v^2*w^2+5*v*w^6==
v^5+15*u^2*v*w+15*v^3*w^3+6*u*w^5
```

Ясно, что уже в подобном примере выполнение исключения вручную требует известного присутствия духа и уверенности в своих технических возможностях.

При применении команды `Eliminate` система пытается известными ей методами исключать и неизвестные из трансцендентных уравнений, конечно, не всегда одинаково успешно.

• **Решение неравенств.** Естественно, `Mathematica` может решать не только уравнения, но и неравенства. В § 1 мы уже видели, что неравенство записывается обычным образом:

- Строгое неравенство  $x > y$  `Greater` или  $x < y$  `Less` ;
- Нестрогое неравенство  $x \geq y$  `GreaterEqual` или  $x \leq y$  `LessEqual` .

Обычно система считает, что связанные неравенством величины представляют собой *вещественные числа*, но она производит и некоторые упрощения частей неравенства и в том случае, когда они не являются числами, хотя и не может в этом случае приписать неравенству значение истинности `True` или `False`.

Основной командой для решения неравенств и систем неравенств в языке `Mathematica` является `Reduce`. В простейшем варианте команда `Reduce` вызывается в формате

`Reduce[expression, {x,y}]`

где  $x, y$  это переменные, относительно которых мы хотим разрешить систему уравнений, неравенств и логических высказываний, а `expression` представляет собой произвольную логическую комбинацию следующих вещей:

- уравнений  $f[x, y, \dots] == f[x, y, \dots]$ ,
- неравенств  $f[x, y, \dots] != f[x, y, \dots]$ ,
- строгих неравенств  $f[x, y, \dots] < f[x, y, \dots]$ ,
- нестрогих неравенств  $f[x, y, \dots] \leq f[x, y, \dots]$ ,
- спецификаций доменов `Element[x, domain]`, утверждающих, что  $x$  принадлежит домену `domain`.
- кванторов всеобщности `ForAll[x, condition[x], test[x]]`, возвращающих значение `True`, если тест `test[x]` принимает значение `True` для всех значений  $x$ , для которых выполняется условие `condition[x]`.
- кванторов существования `Exists[x, condition[x], test[x]]`, возвращающих значение `True`, если *существует* значение  $x$ , удовлетворяющее условию `condition[x]`, для которого тест `test[x]` принимает значение `True`.

По умолчанию команда `Reduce` считает, что все входящие в нее неизвестные принимают *комплексные* значения, причем все неизвестные, которые явным образом входят в строгие и нестрогие неравенства — *вещественные*. Поэтому для решения неравенств в *вещественных* числах никаких спецификаций доменов задавать не нужно. Если мы хотим решить систему неравенств в *целых* числах, то проще всего тоже не задавать спецификацию доменов для каждой переменной по отдельности, а сразу вызывать команду `Reduce` в формате:



`Reduce[expression,{x,y},Integers]`

В случае одного неравенства использование команды `Reduce` и вид получающегося при этом ответа ничем не отличается от того, что мы уже видели для команды `Roots` в случае одного уравнения:

`In[58]:=Reduce[x+(x-1)/(x+1)>0,x]`

`Out[58]=-1-Sqrt[2]<x<-1||x>-1+Sqrt[2]`

Разумеется, в общем случае, сводящемся к решению уравнений степени  $\geq 3$  решение будет выражено в терминах объектов типа `Root`:

`In[59]:=Reduce[x^2+(x-1)/(x^2+x+1)>1,x]`

`Out[59]=x<Root[2+2*#1+2*#1^2+#1^3&,1]||x>1`

Для неравенств, сводящихся к уравнениям степени  $\geq 4$ , мы всегда можем, *при желании* выразить ответ в радикалах, установив в теле функции опцию `Cubics->True` и/или `Quartics->True`, в следующем формате:

`In[60]:=Reduce[x^2+(x-1)/(x^2+x+1)>1,x,Cubics->True]`

`Out[60]=x<1/3*(-2-2/(-17+3*Sqrt[33]))^(1/3)+  
(-17+3*Sqrt[33]))^(1/3)||x>1`

Разумеется, неравенство совсем не обязано с самого начала иметь полиномиальный вид. С тем же успехом система решает любые неравенства, *сводящиеся* к алгебраическим, скажем, неравенства, содержащие абсолютные величины, квадратные корни и пр. Вот пример, основанный на задаче, фактически предлагавшейся на вступительном экзамене на экономический факультет МГУ:

`In[61]:=Simplify[Reduce[Abs[y]+x-Sqrt[x^2+y^2-1]>=1,{x,y}]]`

`Out[61]=Element[y,Reals]&&  
(x==0&&(y==-1||y==1)||  
0<x<1&&(-1<=y<=-Sqrt[1-x^2]||Sqrt[1-x^2]<=y<=1)||  
x==1||  
x>1&&(y<=-1||y>=1))`

Обратите внимание, что так как  $y$  входит в неравенство не в явном виде, а в аргументе функций `Abs` и `Sqrt`, в ответе система считает нужным подчеркнуть, что она считает  $y$  *вещественным* числом. При решении неравенств с абсолютными величинами нужно быть чрезвычайно аккуратным. Дело в том, что если все переменные входят в неравенства только под знаком абсолютной величины, то система будет считать их комплексными. Поэтому, скажем, `Reduce[Abs[x]+Abs[y]<1,{x,y}]` вернет гораздо больше решений, чем Вы ожидали! Если вы хотите увидеть только вещественные решения, нужно явно специфицировать домен, вызывая эту команду в следующем формате:

`In[62]:=Reduce[Abs[x]+Abs[y]<1,{x,y},Reals]`

`Out[62]=-1<x<=0&&-1-x<=y<1+x||0<x<1&&-1+x<=y<1-x`

Разумеется, система может решить и неравенства, в которые входят трансцендентные функции, по крайней мере в тех случаях, когда она мо-

жет решить соответствующее уравнение! Более того, даже для трансцендентных *уравнений* заведомо предпочтительно использовать именно команду `Reduce`, так как она всегда пытается определить все множество решений, а не просто *какие-то* решения. Однако во многих случаях при попытке найти *точные* решения неравенства ответом будет сакраментальное `Reduce::nsmet: This system cannot be solved with the methods available to Reduce.`

Модификации, которые необходимо внести в использование команды `Reduce` для решения систем неравенств относительно нескольких неизвестных, точно такие же, как при использовании команды `Solve` для решения системы уравнений. А именно, в качестве первого аргумента команды `Reduce` в этом случае можно задавать список неравенств, а в качестве второго — список неизвестных, приблизительно в таком формате:

```
Reduce[{f1[x,y]>g1[x,y],f2[x,y]>g2[x,y],...},{x,y}]
```

С другой стороны, эту функцию можно вызывать и в следующем формате:

```
Reduce[f1[x,y]>g1[x,y]&&f2[x,y]>g2[x,y]&&...},{x,y}]
```

Однако следует иметь в виду, что уже решение *простейших* систем неравенств может состоять из нескольких компонент — иногда из огромного числа компонент:

```
In[63]:=Reduce[Abs[x]+Abs[y]>1&&x^2+x*y+y^2<1,{x,y}]
```

```
Out[63]=-2/Sqrt[3]<x<=-1&&
      -x/2-1/2*Sqrt[4-3*x^2]<y<-x/2+1/2*Sqrt[4-3*x^2]||
      -1<x<0&&(-x/2-1/2*Sqrt[4-3*x^2]<y<-1-x||
      1+x<y<-x/2+1/2*Sqrt[4-3*x^2])||
      0<x<=1&&(-x/2-1/2*Sqrt[4-3*x^2]<y<-1+x||
      1-x<y<-x/2+1/2*Sqrt[4-3*x^2])||
      1<x<2/Sqrt[3]&&
      -x/2-1/2*Sqrt[4-3*x^2]<y<-x/2+1/2*Sqrt[4-3*x^2]
```

Вы действительно в состоянии *сразу* увидеть все четыре компоненты решения по этим формулам? Мы уверены, что в подобных случаях — а также для трансцендентных уравнений — гораздо удобнее пользоваться *графическим* представлением ответа при помощи функции `InequalityPlot`, содержащейся в пакете `Graphics`InequalityGraphics``. Мы совсем коротко описываем использование этой функции в § 6.

## § 5. ЭЛЕМЕНТАРНЫЕ ФУНКЦИИ

Я стал немного забывать теорию функций. Ну, это восстановится. Врач обещал ... врет, наверно.

Владимир Высоцкий. 'Дельфины и психи (записки сумасшедшего)'

Система `Mathematica` знает определения большого количества элементарных и специальных функций, и громадное количество фактов об их значениях, их поведении, дифференциальные и функциональные уравнения,

которым они удовлетворяют, etc., etc., etc. Эти знания в сочетании с мощью ее интеллекта позволяют без труда решать любую задачу об этих функциях, которая может встретиться школьнику, студенту и вообще любому нематематику.

• **Экспоненты и логарифмы.** Как мы уже упоминали, именами всех обычных функций в языке *Mathematica* являются либо их *полные* английские названия, либо *стандартные* сокращения этих названий. Неспециалист должен иметь в виду, что — особенно в области элементарной математики — многие английские сокращения, в частности, имена большинства тригонометрических и гиперболических функций, отличаются от континентальных. В то же время русская педагогическая традиция основана на латинских и французских сокращениях!

В соответствии с этим общим принципом экспонента и логарифм называются *Exp* и *Log*:

- *Exp*[*x*] возвращает  $e^x$ ;
- *Log*[*x*] возвращает натуральный логарифм  $x$ ;
- *Log*[*b*,*x*] возвращает логарифм  $x$  по основанию  $b$ . Обратите внимание на весьма необычный (для языка *Mathematica*!) порядок аргумента и параметра: в записи большинства функций аргументы предшествуют параметрам.

Стоит предупредить, что в языке *Mathematica* есть и функция *Exponent*, но по-английски слово *exponent* означает как собственно экспоненту, т.е. степень, так и показатель степени. Так вот *Exponent*[*f*,*x*] обозначает наибольший *показатель* степени, с которым  $x$  входит в  $f$ .

Экспонента и логарифм наряду с круговыми (“тригонометрическими”) и гиперболическими функциями являются примерами числовых функций. В выпуске 2 мы достаточно подробно обсуждаем специфику **числовых функций**, поэтому ограничимся пока констатацией того, что в тех случаях, когда их аргументы принимают численные значения, эти функции всегда, когда это возможно, пытаются вернуть *точные* численные значения.

В природных условиях *Mathematica* работает с экспонентой и логарифмом как *комплексными* функциями, считая при этом, что для логарифма разрез произведен по лучу  $(0, -\infty)$ . Вот, например, как *Mathematica* понимает **формулу Эйлера**:

```
In[64]:=ComplexExpand[Exp[x+I*y]]
```

```
Out[64]=E^x*Cos[y]+I*E^x*Sin[y]
```

А вот что такое логарифм отрицательного числа:

```
In[65]:=Refine[Log[x],x<0]
```

```
Out[65]=I*Pi+Log[-x]
```

Использованная здесь команда *Refine* является одной из самых полезных команд системы при работе с числовыми функциями. Вызванная в формате

```
Refine[expression,assumptions]
```

она упрощает выражение `expression` так как она сделала бы это при условии, что входящие в него символы заменены *явными числовыми значениями*, удовлетворяющими предположениям `assumptions`. В действительности во многих ситуациях, например, когда мы пытаемся упростить несколько различных функций при одних и тех же предположениях, удобно вызывать `Refine` внутри вспомогательной команды `Assuming`, в следующем формате:

```
Assuming[assumptions,Refine[expression]]
```

Стоит подчеркнуть, что команда `Refine` не конкурирует с командами `Simplify` и `FullSimplify`, а дополняет их. Дело в том, что `Simplify` опирается главным образом на общие полиномиальные алгоритмы, в то время как `Refine` использует несколько сотен *конкретных* типов преобразований известных системе числовых функций. Например, не только `Simplify`, но даже `FullSimplify` не выполняет произведенного выше упрощения:

```
In[66]:=FullSimplify[Log[x],x<0]
```

```
Out[66]=Log[x]
```

Это значит, что для упрощения формул, в которые входят значения числовых функций, полезно применить к ним как команду `Simplify`, так и команду `Refine`.

Используемые командой `Refine` предположения могут состоять из произвольных логических комбинаций неравенств, уравнений и спецификаций доменов. Непосредственно в языке ядра описаны всего *семь* доменов:

- $\mathbb{C}$  = `Complexes` — комплексные числа,
- $\mathbb{R}$  = `Reals` — вещественные числа,
- $\overline{\mathbb{Q}}$  = `Algebraics` — алгебраические числа (в документации фирмы `Wolfram Research` этот домен обозначается через  $\mathbb{A}$ , но большинство алгебраистов использует обозначение  $\mathbb{A}$  не для поля алгебраических чисел, а для кольца *целых* алгебраических чисел),
- $\mathbb{Q}$  = `Rationals` — рациональные числа,
- $\mathbb{Z}$  = `Integers` — целые числа,
- $\mathbb{P}$  = `Primes` — простые числа,
- `{True,False}` = `Booleans` — значения истинности.

При этом условие принадлежности элемента  $x$  домену `Domain` записывается в виде

```
Element[x,Domain]
```

Вот пример упрощения, использующего условие  $y \in \mathbb{Z}$ :

```
In[67]:=Refine[Exp[x+I*Pi*y],Element[y,Integers]]
```

```
Out[67]=(-1)^y*E^x
```

• **Круговые и гиперболические функции.** Естественно, в `Mathematica` имплементированы и все остальные основные элементарные функции, в частности, круговые, гиперболические, обратные круговые и обратные

гиперболические. Все они рассматриваются как числовые функции (комплексного аргумента) и к ним относится все, что было сказано выше об экспоненте и логарифме.

Работая с комплексными числами мы уже имели возможность убедиться, что косинус и синус называются *Cos* и *Sin*. Аргумент круговых функций выражается в радианах, причем по умолчанию всегда возвращается *точное* значение функции. Впрочем, как и для любой числовой функции, точное значение всегда можно превратить в приближенное, применяя функцию *N*:

```
In[68]:={Sin[Pi/12],Sin[Pi/8],Sin[1],N[Sin[1]]}
```

```
Out[68]={(-1+Sqrt[3])/(2*Sqrt[2]),Sin[Pi/8],Sin[1],0.841471}
```

Однако при желании аргумент круговых функций можно выражать и в градусах, для этого его величину в градусах нужно умножить на константу *Degree*, равную *численному значению*  $\pi/180$ . Приведем первые 50 разрядов этой константы:

```
In[69]:=N[Degree,50]
```

```
Out[69]=0.017453292519943295769236907684886127134428718885417
```

Подчеркнем, что в отличие от *Pi* константа *Degree* имеет формат *приближенного* вещественного числа (хотя и неопределенной разрядности). Это значит, что *Mathematica* не может ответить на вопрос  $\text{Pi}/180 == \text{Degree}$ . В то же время тест  $\text{N}[\text{Pi}/180, d] == \text{Degree}$  даст значение *True* при любом количестве разрядов *d*. Тем не менее, к нашему большому удивлению, вычисление значений круговых функций от углов, выраженных в градусах, дает *точные* значения:

```
In[70]:={Cos[15*Degree],Sin[15*Degree]}
```

```
Out[70]={(1+Sqrt[3])/(2*Sqrt[2]),(-1+Sqrt[3])/(2*Sqrt[2])}
```

Команда *Refine* работает обычным образом:

```
In[71]:=Refine[Cos[x+Pi/2],Sin[Pi/2-x]]
```

```
Out[71]={-Sin[x],Cos[x]}
```

В соответствии с английской традицией тангенс и котангенс называются *Tan* и *Cot*:

```
In[72]:={Sin[x]/Cos[x],Cos[x]/Sin[x]}
```

```
Out[72]={Tan[x],Cot[x]}
```

Названия основных гиперболических функций получаются из названий соответствующих круговых функций дописыванием буквы *h*. Таким образом, например, гиперболический косинус и гиперболический синус называются *Cosh* и *Sinh*, соответственно. Названия обратных функций получаются из исходных функций приписыванием приставки *Arc*, при этом имя исходной функции не меняется (в частности, продолжает писаться с большой буквы). Таким образом, например, арккосинус и арксинус называются *ArcCos* и *ArcSin*, а гиперболический арккосинус и гиперболический арксинус — *ArcCosh* и *ArcSinh*, соответственно.

• **Значения элементарных функций.** Одной из самых мощных и полезных команд системы Mathematica для работы со значениями *числовых функций* является команда `FunctionExpand`, которая пытается привести выражение к пусть и более длинному, но более “элементарному” виду. Таким образом, действие `FunctionExpand` в определенном смысле *противоположно* действию команды `Simplify`, которая пытается представить выражения в самом *коротком* виде, хотя бы как значения высших трансцендентных функций. При этом последовательное применение команд `FunctionExpand` и `Simplify` обычно не возвращает исходное выражение, а преобразует его к какому-то совсем другому виду! Команда `FunctionExpand` *пытается* сделать следующее:

- избавиться от явного применения операций анализа (дифференцирование, интегрирование и т.д.);
- выразить значения специальных функций в терминах элементарных функций и констант;
- выразить значения элементарных функций в терминах известных констант и арифметических операций;
- упростить вид аргументов.

Мы уже видели, что Mathematica не преобразует  $\sin(\pi/8)$  к какому-либо другому виду, так как не считает, что, скажем, выражение этого значения в радикалах *проще*, чем  $\sin(\pi/8)$ . Однако выражение этого значения в радикалах “элементарнее”:

```
In[73]:=FunctionExpand[{Cos[Pi/8],Sin[Pi/8]}]
```

```
Out[73]={Sqrt[2+Sqrt[2]]/2,Sqrt[2-Sqrt[2]]/2}
```

Для всех более сложных случаев поверх `FunctionExpand` рекомендуется применять `Simplify`:

```
In[74]:=Simplify[FunctionExpand[Cos[2*Pi/7]]]
```

```
Out[74]=-1/6+1/6*(7/2*(1-3*I*Sqrt[3]))^(1/3)+
      (I*(7/2*(1-3*I*Sqrt[3]))^(2/3))/(3*I+9*Sqrt[3])
```

Если ответ все еще представляется Вам сложным, то только потому, что Вы не видели, что `FunctionExpand` возвратило до применения `Simplify`!

Команда `FunctionExpand` может применяться и к значениям числовых функций в случае символьных аргументов, удовлетворяющих определенным предположениям. В этом случае она вызывается в том же формате, что и команда `Refine`:

```
FunctionExpand[expression,assumptions]
```

Однако она действует совершенно иначе, чем `Refine`. Это очень хорошо видно на следующем примере:

```
In[75]:=Refine[Log[x*y],x>0&&y>0]
```

```
Out[75]=Log[x*y]
```

```
In[76]:=FunctionExpand[Log[x*y],x>0&&y>0]
```

Out[76]=Log[x]+Log[y]

• **Структурные манипуляции.** В ядре системы описаны основные структурные манипуляции в кольцах  $\text{Trig}_{\mathbb{R}}$  и  $\text{Trig}_{\mathbb{C}}$  вещественных и комплексных тригонометрических многочленов. Вот некоторые простейшие из них:

◦ **TrigExpand** представляет тригонометрический многочлен как линейную комбинацию одночленов  $\cos(x)^m \sin(x)^n$ ;

◦ **TrigFactor** раскладывает тригонометрический многочлен на множители;

◦ **TrigReduce** приводит тригонометрическое выражение к наиболее простому с точки зрения системы виду.

Возьмем функцию  $\text{Cos}[2*x]*\text{Cos}[3*x]$  и посмотрим, что с ней делают эти преобразования:

In[77]:=TrigExpand[Cos[2\*x]\*Cos[3\*x]]

Out[77]=Cos[x]/2+Cos[x]^5/2-5\*Cos[x]^3\*Sin[x]^2+5/2\*Cos[x]\*Sin[x]^4

In[78]:=TrigFactor[Cos[2\*x]\*Cos[3\*x]]

Out[78]=Cos[x]\*(-1+2\*Cos[2\*x])\*(Cos[x]-Sin[x])\*(Cos[x]+Sin[x])

In[79]:=TrigReduce[Cos[2\*x]\*Cos[3\*x]]

Out[79]=1/2\*(Cos[x]+Cos[5\*x])

При решении уравнений в элементарных функциях предварительная обработка уравнений этими преобразованиями обычно гораздо эффективнее, чем непосредственное применение команды **Solve**. Скажем, непосредственное вычисление

In[80]:=Solve[(Sin[x]+Cos[x])/Sqrt[2]+Cos[2\*x]/Sqrt[3]==1,x]

(пример, фактически предлагавшийся на вступительном экзамене по математике на экономический факультет) дает ответ, но это такой ответ, который сам не помотришь и другим не покажешь. Конечно, правильная стратегия состоит в проведении следующего вычисления:

In[80]:=TrigFactor[(Sin[x]+Cos[x])/Sqrt[2]+Cos[2\*x]/Sqrt[3]]

Out[80]=1/6\*(Cos[x]+Sin[x])\*  
(3\*Sqrt[2]+2\*Sqrt[3]\*Cos[x]-2\*Sqrt[3]\*Sin[x])

после чего решения уравнения непосредственно очевидны.

• **Тригонометрические преобразования.** К сожалению, довольно часто приходится вручную контролировать, какие именно преобразования проводятся с тригонометрическими выражениями. К этому приходится прибегать для функций больших степеней, сложного вида аргументов, а также в тех случаях, когда Вы хотите увидеть ответ в каком-то определенном виде. Дело в том, что система всегда может породить *какой-то* ответ, но может случиться, что он будет выражен не в той форме, которая Вам нужна. Допустим, мы хотим выразить  $\text{tg}(x+y+z)$  в терминах  $\text{tg}(x)$ ,  $\text{tg}(y)$  и  $\text{tg}(z)$ . Непосредственное применение  $\text{tg}(x+y+z)$  команды

`FunctionExpand` приведет к нескольким строчкам косинусов и синусов  $x, y$  и  $z$ , от которых начинает рябить в глазах. В этом месте полезно вспомнить команду `Together`, приводящую дроби к общему знаменателю:

```
In[81]:=Together[TrigExpand[Tan[x+y+z]]]
Out[81]=(Cos[y]*Cos[z]*Sin[x]+Cos[x]*Cos[z]*Sin[y]+
        Cos[x]*Cos[y]*Sin[z]-Sin[x]*Sin[y]*Sin[z])/
        (Cos[x]*Cos[y]*Cos[z]-Cos[z]*Sin[x]*Sin[y]-
        Cos[y]*Sin[x]*Sin[z]-Cos[x]*Sin[y]*Sin[z])
```

Но это все еще не то, на что мы надеялись. Не приводит к успеху ни применение `FunctionExpand`, `Refine` или `Simplify`. В этом случае нам не остается ничего другого, кроме как явно задавать **правила преобразования**, которые мы хотим применить. Мы подробно описывает йогу замен, правил и подстановок в Выпуске 2. Пока ограничимся замечанием, что применение правила `lhs->rhs` к выражению `expression` оформляется в виде:

```
expression /. lhs->rhs
```

если мы хотим однократно применить это правило ко всем частям выражения и в виде

```
expression //. lhs->rhs
```

если мы хотим, чтобы система применяла правило не только к исходному выражению, но и к получающимся выражениям `quantum satis`, пока выражение не перестает меняться. Первая из этих форм называется `ReplaceAll`, а вторая — `ReplaceRepeated`.

При этом само правило преобразования `lhs->rhs` задается в формате

```
f[x_,y_->g[x,y]
```

где  $f(x, y)$  — функция, которую мы хотим переписать в виде  $g(x, y)$ . Обратите особое внимание на **бланки** \_ в левой части!!! Эти бланки делают  $x$  и  $y$  немymi переменными и сообщают системе, что это правило должно применяться, если подставить сюда вместо  $x$  и  $y$  любые символы и/или численные значения. ПРОПУСК БЛАНКОВ ЯВЛЯЕТСЯ ГРУБЕЙШЕЙ СИНТАКСИЧЕСКОЙ ОШИБКОЙ. В этом случае система будет знать, что  $f(x, y)$  следует заменить на  $g(x, y)$ , но это ее знание не будет распространяться на  $f(a, b)$ ,  $f(z, w)$  и т.д.

Проиллюстрируем задание правил преобразования на примере. Допустим, мы не знаем, что команда `TrigFactor` автоматически переписывает  $\operatorname{ctg}(x) - \operatorname{ctg}(y)$  в виде  $-\sin(x - y)/(\sin(x)\sin(y))$  и хотим задать соответствующее правило преобразования. В этом случае мы можем задать его в виде

```
Cot[x_]-Cot[y_->-Sin[x-y]/(Sin[x]*Sin[y])
```

Вернемся теперь к разложению  $\operatorname{tg}(x + y + z)$  и заставим систему преобразовывать  $\operatorname{tg}(x + y)$  в  $(\operatorname{tg}(x) + \operatorname{tg}(y))/(1 - \operatorname{tg}(x)\operatorname{tg}(y))$  столько раз, сколько она видит выражение такого вида:

```
In[82]:=Together[Tan[x+y+z] //.
```



```

Tan[x_+y_]->(Tan[x]+Tan[y])/(1-Tan[x]*Tan[y])
Out[82]=(-Tan[x]-Tan[y]-Tan[z]+Tan[x]*Tan[y]*Tan[z])/
(-1+Tan[x]*Tan[y]+Tan[x]*Tan[z]+Tan[y]*Tan[z])

```

После приобретения некоторого опыта правила преобразования становятся одним из основных инструментов программирования на языке *Mathematica*, позволяющим полностью контролировать вид ответа.

• **Экспоненциальная и тригонометрическая форма.** Функции  $x \mapsto \cos(ax)$  и  $x \mapsto \sin(ax)$  порождают то же пространство, что  $x \mapsto e^{iax}$  и  $x \mapsto e^{-iax}$ . Следующие команды осуществляют пересчет из тригонометрического базиса в экспоненциальный и наоборот:

- **TrigToExp** — преобразование выражения из тригонометрической формы в экспоненциальную;
- **ExpToTrig** — преобразование выражения из экспоненциальной формы в тригонометрическую.

В следующих вычислениях мы преобразуем тригонометрическое выражение в экспоненциальную форму:

```

In[83]:=FactorTerms[TrigToExp[Cos[2*x]*Cos[3*x]]]
Out[83]=1/4*(E^(-I*x)+E^(I*x)+E^(-5*I*x)+E^(5*I*x))
In[84]:=TrigToExp[Tan[x],Cot[x]]
Out[84]={(I*(E^(-I*x)-E^(I*x)))/(E^(-I*x)+E^(I*x)),
-(I*(E^(-I*x)+E^(I*x)))/(E^(-I*x)-E^(I*x))}

```

Функция **FactorTerms** применена, чтобы вынести общий численный множитель.

Вот еще один очень интересный пример. По умолчанию команды типа **Solve** записывают первообразные корни из 1 степени 5 в экспоненциальном виде:

```

In[85]:=Solve[x^4+x^3+x^2+x+1==0,x]
Out[85]={ {x->-(-1)^(1/5)}, {x->-(-1)^(2/5)},
{ x->-(-1)^(3/5)}, {x->-(-1)^(4/5)} }

```

Однако большинство начинающих предпочтут увидеть их в тригонометрическом виде:

```

In[86]:=Map[ExpToTrig,Solve[x^4+x^3+x^2+x+1==0,x],3]
Out[86]={ {x->-1/4-Sqrt[5]/4-1/2*I*Sqrt[1/2*(5-Sqrt[5])]}
{ {x->-1/4+Sqrt[5]/4+1/2*I*Sqrt[1/2*(5+Sqrt[5])]}
{ {x->-1/4+Sqrt[5]/4-1/2*I*Sqrt[1/2*(5+Sqrt[5])]}
{ {x->-1/4-Sqrt[5]/4+1/2*I*Sqrt[1/2*(5-Sqrt[5])]} }

```

Команда **Map**, вызванная в формате

```
Map[f,list,d]
```

применяет функцию  $f$  к списку  $list$  на уровне  $d$ . В приведенном выше примере **ExpToTrig** применяется на уровне 3. На уровне 3 в выражении,

получающемся после применения `Solve`, лежат  $x$  и собственно корни из 1. Однако ясно, что с  $x$  команда `ExpToTrig` ничего не делает.

• **Экстремумы функции.** При помощи системы `Mathematica` можно провести “исследование функций”. Упомянем лишь часто встречающуюся задачу отыскания максимумов и минимумов. Основными командами для этого являются `Maximize` и `Minimize`. Так как их использование абсолютно аналогично, в дальнейшем мы будем говорить только о `Maximize`. Для поиска глобального максимума можно вызывать эту команду в формате

```
Maximize[f[x], x]
```

При этом ответ возвращается в формате  $\{a, \{x \rightarrow c\}\}$  с указанием максимума  $a$  функции  $x \mapsto f(x)$  и *какого-то* (обычно наименьшего) значения аргумента  $c$ , в котором этот максимум достигается.

Разумеется, глобальный максимум не обязан существовать или может быть бесконечным. Например, при попытке вычислить `Maximize[x^2, x]` мы получим сообщение

```
Maximize::natt: The maximum is not attained at any point
satisfying the given constraints.
```

и ответ  $\{\text{Infinity}, \{x \rightarrow -\text{Infinity}\}\}$ .

В действительности гораздо чаще приходится производить не глобальную оптимизацию, а оптимизацию с **ограничениями** `constraints`. Для этого функции `Maximize` и `Minimize` вызываются в формате

```
Maximize[{f[x], constraints}, x]
```

где `constraints` обозначает ограничения или условия, при которых ищется экстремум. Вот, скажем, как ищется максимум функции  $f$  на отрезке  $[a, b]$ :

```
Maximize[{f[x], a <= x <= b}, x]
```

Однако уже для очень простых трансцендентных функций явная оптимизация может приводить к довольно сложным ответам:

```
In[87]:=FullSimplify[Maximize[{Sqrt[x]-Exp[x], 0<=x<=2}, x]]
```

```
Out[87]={(-1+ProductLog[1/2])/Sqrt[2*ProductLog[1/2]],
{x->1/2*ProductLog[1/2]}}
```

Встречающаяся здесь функция `ProductLog[x]` представляет собой главное решение уравнения  $ye^y = x$ , удовлетворяющее дифференциальному уравнению  $\frac{dy}{dx} = \frac{y}{x(1+y)}$ . Эта функция очень часто возникает при решении уравнений, в которые входят экспонента и/или логарифм.

В данном случае нам крупно повезло, что нашлась функция, решающая возникающее при оптимизации уравнение. Однако в большинстве случаев, сводящихся к решению трансцендентных уравнений, система будет не в состоянии найти точные максимумы и минимумы. Например, попытка вычислить

```
In[88]:=Maximize[{Log[x]+Sin[x], 0<=x<=5}, x]
```

не даст никакого вразумительного ответа. В подобных случаях можно применять команды численной оптимизации `NMaximize` и `NMinimize`. Эти команды вызываются точно в таком же формате, как команды `Maximize` и `Minimize` и дают приближенные значение экстремума и той точки, в которой оно достигается:

```
In[89]:=NMaximize[{Log[x]+Sin[x],0<=x<=5},x]
Out[89]={1.60552,{x->2.07393}}
```

## § 6. ГРАФИКИ ФУНКЦИЙ

Die Mathematik ist vielmehr eine Wissenschaft für das Auge als eine für das Ohr. — Математика в гораздо большей степени обращается к глазу, чем к уху.

Carl Friedrich Gauß

Und was man sieht und abzeichnet ist immer besser als was man nur so selbst erfindet. — И то, что срисовываешь с натуры, всегда получается лучше того, что пытаешься высосать из пальца.

Wilhelm Hauff

Из-за искажения масштаба дисплеем компьютера окружность выглядит как эллипс.

Владимир Дьяконов<sup>53</sup>

A conjecture both deep and profound  
Is whether the circle is round;  
In a paper by Erdős,  
written in Kurdish,  
A counterexample is found.<sup>54</sup>

В действительности с точки зрения профессионала одной из самых сильных сторон системы `Mathematica`, являются огромные возможности **визуализации** вычислений. С практической точки зрения эти возможности ограничены *только* способностью пользователя призывать их. Дело в том, что хотя картинки произвольно высокой сложности могут быть порождены чрезвычайно простыми формулами, способность фактически установить полное соответствие между вычислениями и графикой требует, как минимум,

- о хорошего понимания математики, стоящей за этими формулами,
- о свободного владения функциональным программированием,

<sup>53</sup>В настоящей книге встречаются изредка НЕПРАВИЛЬНО ИСТОЛКОВАННЫЕ НЕ ЦИТАТЫ (*misconstrued misquotations*), но именно этот эпиграф самый что ни на есть *подлинный*!!! Тот, кто думает, что мы его снова разыгрываем, может сам взглянуть на рисунок и легенду к нему на странице 323 следующей книги: В.П.Дьяконов, *Mathematica 4*. — Питер, 2001, с.1–654.

<sup>54</sup>Окружность я не рисовал, я с детства рисовал овал. — Поль Эрдеши (перевод с курдского).

о собственно навыка применения графических команд, настройки их опций и т.д.,

которые приобретаются только в результате длительных упражнений. Поэтому в настоящем параграфе мы опишем только применение небольшого числа встроенных команд для построения графиков функций одной и двух переменных — и то только в *простейших* вариантах. Выпуск 4 настоящей книги почти полностью посвящен именно детальному описанию этих и других графических команд и настройке графических опций и директив для получения желаемого эффекта.

• **Построение графиков.** График функции строится при помощи команды `Plot`, которая порождает объект формата `Graphics`. В простейшем варианте, когда Вы доверяете системе принять вместо Вас **все** эстетические решения, эта команда вызывается в формате:

```
Plot[f[x], {x, a, b}]
```

Эта команда порождает график функции  $f(x)$ , когда  $x$  меняется от  $a$  до  $b$ . Однако в действительности, у команды `Plot` еще 30 факультативных аргументов, называемых **опциями**. Вот опции команды `Plot`, вместе с их фабричными установками:

```
In[90]:=Options[Plot]
```

```
Out[90]={AspectRatio->1/GoldenRatio,Axes->Automatic,
        AxesLabel->None,AxesOrigin->Automatic,
        AxesStyle->Automatic,Background->Automatic,
        ColorOutput->Automatic,Compiled->True,
        DefaultColor->Automatic,DefaultFont->$DefaultFont,
        DisplayFunction->$DisplayFunction,Epilog->{},
        FormatType->$FormatType,Frame->False,
        FrameLabel->None,FrameStyle->Automatic,
        FrameTicks->Automatic,GridLines->None,
        ImageSize->Automatic,MaxBend->10,
        PlotDivision->30,PlotLabel->None,
        PlotPoints->25,PlotRange->Automatic,
        PlotRegion->Automatic,PlotStyle->Automatic,
        Prolog->{},RotateLabel->True,
        TextStyle->$TextStyle,Ticks->Automatic}
```

Мы отложим обсуждение большинства этих опций до выпуска 4. К сожалению, для того, чтобы увидеть то, что Вам хочется, так, как Вам этого хочется, необходимо с самого начала учиться настраивать самые важные опции, *хотя бы* `AspectRatio` и `PlotRange`, может быть, еще `PlotStyle`, `Axes`, `AxesStyle`, `Ticks`, `GridLines`, `PlotLabel` и `TextStyle`. Однако без понимания того, как работают `AspectRatio` и `PlotRange` — как наглядно продемонстрировал В.П.Дьяконов — не удастся узнаваемо нарисовать ровным счетом ничего, даже окружность.

Вот как выглядит типичный вызов функции `Plot`. На рисунке 1 мы

видим график функции  $x \mapsto \ln(x) + \sin(x)$ , построенный при помощи следующей команды:

```
In[91]:=Plot[Log[x]+Sin[x],{x,0,5},
            AspectRatio->Automatic,
            PlotRange->{-3,2},
            PlotStyle->AbsoluteThickness[1.5],
            AxesStyle->AbsoluteThickness[1],
            GridLines->Automatic,
            PlotLabel->"Figure 1: Log[x]+Sin[x]"]
```

Разумеется, мы получили бы график той же функции и напечатав просто

```
In[92]:=Plot[Log[x]+Sin[x],{x,0,5}]
```

без всяких там опций. Но это был бы совсем не такой красивый и информативный график, поэтому поясним использование опций.

◦ **AspectRatio** задает **форматное отношение**, т.е. отношение высоты рисунка к его ширине. Например, **AspectRatio->2** означает, что высота в два раза больше ширины, а **AspectRatio->1/2** — что ширина в два раза больше высоты. Дефолтное форматное отношение равно **1/GoldenRatio**. Это означает, что рисунок имеет те же пропорции, что стандартный лист бумаги формата A4 в **альбомной ориентации** (landscape orientation). В то же время, отношение равное **GoldenRatio** означало, бы, что рисунок имеет пропорции стандартного листа бумаги формата A4 в **книжной ориентации** (portrait orientation). Мы обычно полагаем

★ либо **AspectRatio->1** — в этом случае рисунок вписывается в квадрат;

★ либо **AspectRatio->ysize/xsize**, где **ysize** равно разности концов отрезка **PlotRange**, а **xsize=b-a** — в этом случае масштаб по осям одинаков;

★ либо **AspectRatio->Automatic** — в этом случае система сама решает, что больше соответствует обстановке, и во всех обычных случаях пытается задать одинаковый масштаб по осям. Конечно, это не всегда получается. Например, если функция  $f$  очень быстро растет или убывает, то ее значения могут превосходить значения  $x$  в тысячи раз.

◦ **PlotRange** задает **диапазон графика**, т.е. те значения  $y$ , которые на нем отображаются. Когда функция слишком быстро растет или убывает, система не всегда удачно выбирает значение **PlotRange** и отсекает наиболее интересные части графика. Поэтому часто приходится задавать диапазон вручную:

★ **PlotRange->All** пытается поместить на график **все** значения функции  $f$  при  $x$  меняющемся в заданных пределах;

★ **PlotRange->{c,d}** предлагает команде строить только ту часть графика, которая помещается в горизонтальную полосу  $c \leq y \leq d$ .

Обратите внимание, что в предшествующем примере мы сознательно выбрали **PlotRange** так, чтобы **ysize** равнялось **xsize**. При этом установка **AspectRatio->Automatic** приводит к тому, что масштаб по осям одинаков.

◦ `PlotStyle` задает **стиль графика**, т.е. значения применяемых к нему **графических директив**. Типичными графическими директивами являются задаваемые в типографских точках директивы **ширина** = `Thickness` и **абсолютная ширина** = `AbsoluteThickness`, **штриховка** = `Dashing` и **абсолютная штриховка** = `AbsoluteDashing`, описывающие плотность и прерывистость кривой, а также такие директивы, как **уровень серого** = `GrayLevel`, меняющийся от 0 (черный) до 1 (белый) и многочисленные директивы управления цветом: **тон** = `Hue`, задаваемый списком трех координат HSB (`Hue`, `Saturation`, `Brightness`) и **цвет** = `RGBColor`, задаваемый списком трех координат RGB (`Red`, `Green`, `Blue`), etc., etc.

В данном случае посредством `PlotStyle->AbsoluteThickness[1.5]` мы задали ширину кривой, приблизительно отвечающую насыщенности жирного шрифта. По умолчанию ширина соответствует насыщенности обычного шрифта.

◦ `AxesStyle` задает **стиль осей**, т.е. значения графических директив, применяемых к осям. В этот момент читатель должен догадаться, что напечатав `AxesStyle->AbsoluteThickness[1]`, мы добились того, чтобы оси имели ширину в 1 пункт, чуть меньшую, чем насыщенность полужирного шрифта.

◦ `GridLines` задает **координатную сетку**, т.е. прямые, параллельные координатным осям. По умолчанию `GridLines->None`, так что никаких прямых, кроме самих осей не проводится.

★ `GridLines->Automatic` — прямые проводятся через **метки** (`Ticks`) на осях;

★ `GridLines->{{x1,...,xm},{y1,...,yn}}` — прямые, параллельные оси  $y$ , проводятся через точки  $x_1, \dots, x_m$  на оси  $x$ , а прямые, параллельные оси  $x$  — через точки  $y_1, \dots, y_n$  на оси  $y$ .

В данном случае мы задали опцию `GridLines->Automatic`, что побудило систему проводить прямые через целые точки на осях.

◦ `PlotLabel` задает **ярлык графика**, т.е. заголовок или название, которое обычно пишется *над* графиком. Если Вам хочется перевести `Label` как *метка*, не торопитесь, так как **метка** или **засечка** является стандартным переводом термина `Tick`, который встретится нам далее. Другими близкими опциями являются

★ **ярлыки осей** = `AxesLabel` — текст, который пишется на осях,

★ **ярлыки рамки** = `FrameLabel` — текст, который пишется на рамке,

★ **легенда** = `PlotLegend` — помещенные в рамочку пояснения под графиком.

Обратите внимание, что включенный в задание опции `PlotLabel` текст "Figure 1: Log[x]+Sin[x]" заключается в кавычки. Это превращает этот текст из последовательности символов в **строинг** = `String`, текстовый объект, воспринимаемый и воспроизводимый *verbatim* (буква в букву). Над строингом не производятся *никакие* обычные вычисления. Иными словами,

все специальные символы, которые в природных условиях интерпретировались бы как операторы, в составе строки представляют собой просто типографские знаки. В Выпуске 4 мы детально описываем специальные **текстовые команды**, при помощи которых проводятся преобразования строк.

**Комментарий.** В действительности те картинки, которые Вы фактически видите в этой книге, порождены при помощи несколько более сложного выбора опций. Дело в том, что нам нужно еще привести в соответствие встречающийся в ярлыках, метках и легендах шрифт в соответствие со шрифтом, использованным в основном тексте. Текст этой книги набран в Times New Roman, в то время как стандартные настройки Mathematica порождают вывод в Courier. Это значит, что в действительности для задания ярлыков нам приходилось печатать что-то в таком духе:

```
PlotLabel->StyleForm["Figure 1: Log[x]+Sin[x]",
    FontFamily->"Times", FontWeight->"Bold", FontSize->12]
```

С целью изменения шрифта меток нам нужно было включать в тело команды опцию

```
TextStyle->{FontFamily->"Times",FontSize->12}
```

Обратите внимание на задание опции внутри опции!!! Понятно, что все эти вещи делались исключительно из типографских соображений и при проведении реальных вычислений никому не следует подобным образом извращаться.

• **Основная формула тригонометрии.** Во многих школьных учебниках упоминается “основная формула тригонометрии”, которая при этом записывается в форме  $\cos^2(x) + \sin^2(x) = 1$ . Однако каждый может моментально проверить, что в таком виде эта формула БЕЗНАДЕЖНО НЕВЕРНА. В этом легко убедиться, например, заметив, что

$$\cos^2(\pi/2) + \sin^2(\pi/2) = 1 + \sin(1) \neq 1.$$

Для того, чтобы развеять все вредные иллюзии, изобразим на рисунке 2 график функции  $x \mapsto \cos^2(x) + \sin^2(x)$ , построенный при помощи следующей команды:

```
In[93]:=Plot[Sin[Sin[x]]+Cos[Cos[x]],{x,-3*Pi,3*Pi},
    PlotRange->{0,2},
    AspectRatio->1/5,
    Ticks->{Table[n*Pi,{n,-3,3]},{1,2}},
    AxesStyle->AbsoluteThickness[1],
    PlotStyle->AbsoluteThickness[1.5],
    PlotLabel->"Figure 2: Sin[Sin[x]]+Cos[Cos[x]]"]
```

Повторимся, что мы могли получить *почти* такой же результат и напечатать просто

```
In[94]:=Plot[Sin[Sin[x]]+Cos[Cos[x]],{x,-3*Pi,3*Pi}]
```

без всяких опций. Все остальное — это декорации и отделка деталей.

Большая часть этого текста должна быть Вам уже знакома после разбора предыдущего примера. Обратите внимание на то, что порядок задания опций не имеет значения. В предыдущем примере мы вначале определили AspectRatio, а потом PlotRange, но никто не мешает нам задавать их в

другой последовательности. Кроме того, в этом примере встречается новая опция

◦ **Ticks** — это **метки** или **засечки**, т.е. характерные точки, отмечаемые на осях. При дефолтной установке **Ticks->Automatic** система сама отмечает такие точки. Однако возможны другие установки:

★ **Ticks->None** — метки на осях не ставятся;

★ **Ticks->{{x1,...,xm},{y1,...,yn}}** — метки на оси  $x$  ставятся в точках  $x_1, \dots, x_m$ , а на оси  $y$  — в точках  $y_1, \dots, y_n$ .

В данном случае нам показалось, что для этого графика на оси  $x$  гораздо естественнее отмечать целые кратные  $\pi$ , а вовсе не целые числа. При этом, чтобы чуть сократить ввод с клавиатуры и облегчить возможность дальнейших изменений, мы породили множество целых кратных  $\pi$  как список посредством команды **Table**, см. § 11.

То соотношение между основными тригонометрическими функциями, которое действительно имеет место, это не какая-то мифическая “основная формула тригонометрии”, а **теорема Пифагора**  $\cos(x)^2 + \sin(x)^2 = 1$ . Мы надеемся, что после подобной наглядной демонстрации того, к каким грубым ошибкам неизбежно приводят неряшливые обозначения, читатель будет тщательно различать возведение в квадрат *функции* и возведение в квадрат *ее значения*!

Если к этому моменту Вам надоело печатать при построении каждого графика **AspectRatio->1** или **AspectRatio->Automatic**, то Вы абсолютно правы. В действительности, если Вы на протяжении сессии собираетесь строить графики нескольких функций, имеет смысл с самого начала изменить опции команды **Plot** и/или других используемых Вами графических команд. Например, для того, чтобы во всех графиках, которые Вы строите на протяжении сессии, форматное отношение стало *по умолчанию* равным 1, а метки на осях не ставились, нужно лишь один в начале сессии произвести следующее вычисление:

```
In[95]:=SetOptions[Plot,AspectRatio->1,Ticks->None]
```

Смысл этого вычисления состоит в том, что при этом опциям **AspectRatio** и **Ticks** присваиваются новые значения, отличные от фабричных, и эти новые значения действуют на протяжении всей сессии, пока не будут снова изменены или переопределены.

• **Несколько функций на одном графике.** Особенно интересна возможность строить графики нескольких функций на одной картинке. Вызванная в формате

```
Plot[{f1[x],f2[x],...},{x,a,b}]
```

команда **Plot** *одновременно* строит графики функций  $f_1(x), f_2(x), \dots$  при  $x$  меняющемся от  $a$  до  $b$ .

Например, очень интересно сравнить графики функций  $\sin$ ,  $\sin^2$  и  $x \mapsto \sin(x)^2$ . Это можно сделать при помощи следующей команды:

```
In[96]:=Plot[{Sin[x],Sin[Sin[x]],Sin[x]^2},{x,-2*Pi,2*Pi},
```



```

PlotStyle->
  {{AbsoluteThickness[1.5], Dashing[{0.002,0.02}]},
   {AbsoluteThickness[1], GrayLevel[0]},
   {AbsoluteThickness[1], Dashing[{0.03}]}}},
AxesStyle->AbsoluteThickness[1],
PlotLabel->"Figure 3: Sin[x], Sin[Sin[x]], Sin[x]^2"
Ticks->{{-2*Pi, -3*Pi/2, -Pi/2, 0, Pi/2, 3*Pi/2},
        {-1, 0.5, 1}}]

```

Результат этого вычисления воспроизведен на Рисунке 3. Отметим лишь те моменты, которые нам раньше не встречались.

о Значения графических директив для списка функций тоже могут задаваться списком, при этом первый элемент этого списка будет применяться к первой функции, второй — ко второй и т.д. Например, в данном случае мы рисуем график первой функции с шириной 1.5, а график второй и третьей — с шириной 1. Это делается для того, чтобы они производили впечатление одинаковой жирности.

о Директива **Dashing** описывает **штриховку**. При этом ее аргумент задается в виде списка, элементы которого определяют длины последовательных сегментов кривой, из которых нечетные закрашиваются, а четные — не закрашиваются. После исчерпания элементов списка длины начинают циклически повторяться. При этом, в отличие от **абсолютной штриховки** = **AbsoluteDashing**, эти длины указываются не в каких-то абсолютных величинах, а в долях от общего размера графика. Таким образом, **Dashing[{0.002,0.02}]** описывает кривую, составленную из точек, расстояния между которыми в десять раз больше размера самих точек, а **Dashing[{0.03}]** — кривую, состоящую из закрашенных и незакрашенных сегментов одинаковой длины.

о Наконец, директива **GrayLevel[0]** предписывает рисовать вторую кривую, а именно, график функции  $\sin^2$ , сплошной черной линией.

**Упражнение.** Нарисуйте картинку, воспроизведенную на Рисунке 4.

**Указание.** Обратите внимание на ярлык и метки!

Стоит подчеркнуть, что штриховка здесь использована исключительно из типографских соображений. В обычных условиях для вывода картинки на экран или цветной принтер мы бы варьировали бы не штриховку, а **цвет** кривых. Иными словами, мы задавали бы опцию **PlotStyle** следующим образом:

```
PlotStyle->{RGBColor[1,0,0],RGBColor[1,0,0],RGBColor[0,0,1]}
```

При этом первая кривая изображается чистым красным цветом (в самом деле, **RGBColor[1,0,0]** означает *максимальную* насыщенность красного и *нулевую* насыщенность зеленого и синего), вторая — зеленым и третья — синим. Еще лучше подгрузить определения цветов посредством

```
<<Graphics`Colors`
```

после чего можно обращаться ко всем обычным цветам непосредственно по их обычным английским названиям. Чтобы увидеть имена всех цветов, определения которых известны системе, можно выполнить команду `AllColors`. В Выпуске 4 мы приведем список цветов и их RGB-координаты, с тем, чтобы подлинный эстет мог по аналогии определить свои собственные цвета. Пока ограничимся констатацией того, что `Length[AllColors]` равно 193. Таким образом, *после* подгрузки пакета `Graphics`Colors`` мы можем печатать просто

```
PlotStyle->{Red,Green,Blue}
```

Еще одна интересная представляющаяся здесь возможность состоит в том, чтобы строить на одном графике вещественную и мнимую часть комплексной функции вещественной переменной. Например, на Рисунке 5 изображены графики вещественной и мнимой части дзета-функции Римана на критической прямой, причем вещественная часть изображена пунктиром, а график мнимой части — сплошной. Этот рисунок порожден при помощи следующего текста:

```
In[97]:=Plot[{Re[Zeta[1/2+x*I]],Im[Zeta[1/2+x*I]]},{x,-60,60},
             PlotStyle ->
               {{AbsoluteThickness[1.5],AbsoluteDashing[{1,3]}},
               {AbsoluteThickness[1],GrayLevel[0]}},
             AxesStyle->AbsoluteThickness[1],
             PlotLabel->"Figure 5: Riemann Zeta"
             Ticks -> Automatic]
```

Все в этом тексте уже нам встречалось, за исключением ровно одного момента:

- о Для задания штриховки мы пользуемся не директивой `Dashing`, а директивой **абсолютная штриховка** = `AbsoluteDashing`, аргумент которой представляет собой список длин последовательных сегментов кривой, задаваемых в типографских пунктах. Таким образом, по замыслу график вещественной части состоит из точек размером 1 типографский пункт, разделенных промежутками в три типографских пункта. Однако из-за того, что мы задали абсолютную ширину кривой в 1.5 пункта, точки выглядят больше, чем расстояния между ними.

Естественно, в природных условиях мы различаем вещественную и мнимую часть не штриховкой, а цветом, например, так:

```
PlotStyle->{RGBColor[1,0,0],RGBColor[0,0,1]}
```

В этом случае на экране компьютера график вещественной части будет красным, а график мнимой части — синим.

Стоит предупредить читателя об одной существенной тонкости. Дело в том, что команда `Plot` необычным образом вычисляет свой первый аргумент. Если быть совсем точным, она его вообще никак не вычисляет: она вычисляет лишь *значения* этого аргумента в каких-то точках, а это совсем не то же самое, что вычислять функцию!!!. Поэтому, хотя ее первый

аргумент может быть списком функций, этот список должен быть задан *явно*, а не посредством команды генерации списков. Например, попытка следующим образом построить на одном графике 10 первых многочленов Чебышева

```
Plot[Table[ChebyshevT[n, x], {n, 1, 10}], {x, -1.01, 1.01}]]
```

обречена на провал, так как ее результатом будет серия сообщений об ошибке, наподобие следующего,

```
Plot::plnr: Table[ChebyshevT[n,x],{n,1,10}] is not
a machine-size real number at x = -1.01 ,
```

за которыми последует сообщение, что системе *надоело* предупреждать Вас об ошибке:

```
General::stop: Further output of Plot::plnr will be suppressed
during this calculation.
```

Никакого графика Вам при этом увидеть не удастся. Правильный способ состоит в том, чтобы аннулировать действие атрибута `HoldAll` при помощи команды `Evaluate`, которая заставляет систему немедленно вычислить аргумент. Таким образом, построение графика таблицы функций производится в следующем формате:

```
Plot[Evaluate[Table[f[i][x], {i, 1, n}]], {x, xmin, xmax}]
```

Вот, например, как выполнено построение графика 10 первых многочленов Чебышева, представленное на рисунке 6:

```
In[97]:=Plot[Evaluate[Table[ChebyshevT[n,x],{n,1,10}]],
{ x, -1.01, 1.01},
AspectRatio->Automatic,
PlotStyle->AbsoluteThickness[0.8],
PlotRange->{-1.7, 1.7},
Ticks->{{-1, -0.5, 0.5, 1, 1.5}, {-1, -0.5, 0.5, 1}},
PlotLabel->StyleForm["Figure 6: Chebyshev polynomials"]
```

Все остальные моменты, кроме только что объясненной необходимости использования команды `Evaluate`, уже встречались нам в предыдущих примерах.

• **Параметрический график.** Еще один важнейший способ построить объект формата `Graphics` состоит в том, чтобы использовать команду `ParametricPlot`. В простейшем варианте эта команда вызывается в следующем формате:

```
ParametricPlot[{f[t], g[t]}, {t, a, b}]
```

Эта команда порождает графический объект, изображающий кривую, описываемую точкой с координатами  $(f(t), g(t))$ , когда  $t$  меняется от  $a$  до  $b$ .

Например, во многих областях естествознания возникают **фигуры Лиссажу**, т.е. траектории точки, совершающей гармонические колебания в двух ортогональных направлениях. Конкретный вид этой фигуры определяется периодами, разностью фаз и амплитудами колебаний. Например, на

Рисунке 7 изображена фигура Лиссажу, отвечающая отношению периодов  $3/7$  и разности фаз  $\pi/2$ :

```
In[98]:=ParametricPlot[{Cos[3*t],Sin[7*t]},{t,0,2*Pi},
      PlotStyle->AbsoluteThickness[1.5],
      AxesStyle->AbsoluteThickness[1],
      AspectRatio->Automatic,
      PlotLabel->"Figure 7: Lissajout[3/7,Pi/2]"]
```

**Упражнение.** Постройте кривые Лиссажу, изображенные на рисунках 8–10.

Как и в случае команды `Plot`, мы можем изобразить несколько параметрических кривых на одной картинке. Для этого команду `ParametricPlot` нужно вызывать в следующем формате:

```
ParametricPlot[{f1[t],g1[t]},{f2[t],g2[t]},...,{t,tmin,tmax}]
```

Команда `ParametricPlot` имеет те же особенности, что и `Plot`. В частности, если Вы задаете ее первый аргумент неявным образом, который требует предварительной обработки до того как можно вычислять его значения, то к этому аргументу необходимо применить команду `Evaluate`.

• **График неявной функции.** Пожалуй еще более замечательной, чем возможность строить параметрические графики, является возможность построения графиков неявных функций. Это делается при помощи команды `ImplicitPlot`. Работа команды `ImplicitPlot` основана на способности системы решать системы уравнений (ее имплементация самым существенным образом взывает к команде `Solve` и др.) и, как и многие другие графические функции требует довольно значительного компьютерного ресурса. Поэтому разработчики системы решили не включать ее в ядро — как и большинство аналогичных функций трехмерной графики, в частности, изумительную по возможностям функцию `ParametricPlot3D`. Однако она входит в стандартный пакет `Graphics`ImplicitPlot`` и, поэтому фактически столь же доступна, как функции ядра. Напомним, что для того, чтобы вызывать эту функцию необходимо вначале подгрузить пакет, в котором она определена, выполнив следующую команду:

```
<<Graphics`ImplicitPlot`
```

При желании Вы можете даже подгрузить **все** графические пакеты одновременно при помощи

```
<<Graphics`
```

После этого команда `ImplicitPlot` вызывается в обычном формате

```
ImplicitPlot[f[x,y]==g[x,y],{x,a,b}]
```

где  $f(x,y) = g(x,y)$  — уравнение, решения которого мы хотим изобразить на графике, а  $x$  меняется от  $a$  до  $b$ . В таком формате  $x$  принимается за независимую переменную, и уравнение будет приводиться к объединению графиков функций  $y = h_1(x), y = h_2(x), \dots$ . Ясно, что напечатав

```
ImplicitPlot[f[x,y]==g[x,y],{y,c,d}]
```

мы (при подходящем выборе  $c$  и  $d$ ) получим ту же картинку, но развернутую на 90 градусов, так как теперь независимой переменной считается  $y$ , а не  $x$ . Мы не будем обсуждать, что происходит, если напечатать

```
ImplicitPlot[f[x,y]==g[x,y],{x,a,b},{y,c,d}]
```

так как ответ на этот вопрос существенно зависит от вида уравнения!

На рисунках 11–14 представлены **эллиптические кривые**. Для непосвященного читателя заметим, что они называются так вовсе не потому, что похожи на эллипс, а потому, что их изучение было первоначально мотивировано теорией **эллиптических интегралов** и **эллиптических функций**, которые, в свою очередь, действительно впервые появились в работе братьев Бернулли при вычислении длины дуги эллипса. В прошлое десятилетие эллиптические кривые стали чрезвычайно знамениты в широких кругах образованной публики благодаря той роли, которую они сыграли в доказательстве последней теоремы Ферма Эндрю Уайлсом<sup>55</sup>.

Вот, например, как строилась первая из этих кривых:

```
In[99]:=ImplicitPlot[y^2==x^3-x,{x,-1.2,1.6},
    AspectRatio->Automatic,
    PlotStyle->AbsoluteThickness[1.5],
    AxesStyle->AbsoluteThickness[1],
    PlotRange->{-1.7,1.7},
    Ticks->{{-1,-0.5,0.5,1,1.5},{-1,-0.5,0.5,1}},
    PlotLabel->"Figure 11: y^2=x^3-x"]
```

**Упражнение.** Постройте эллиптические кривые, изображенные на рисунках 12–14, обращая при этом внимание на детали!

• **Графическое представление неравенств.** Еще одной совершенно изумительной функцией системы, которая за секунды проделывает то, на выполнение чего обычному человеку нужно трудиться полдня, является команда `InequalityPlot`, описанная в стандартном графическом пакете `Graphics`InequalityGraphics``. Как всегда, прежде чем первый раз вызывать эту функцию, необходимо вначале подгрузить пакет:

```
<<Graphics`InequalityGraphics`
```

После этого функция `InequalityPlot` вызывается в обычном формате

```
InequalityPlot[f1[x,y]<g1[x,y]&&f2[x,y]<g2[x,y]&&...,
    {x,a,b},{y,c,d}]
```

и изображает те точки прямоугольника

$$\{(x, y) \in \mathbb{R}^2 \mid a \leq x \leq b, c \leq y \leq d\},$$

которые удовлетворяют неравенствам

$$f_1(x, y) < g_1(x, y), \quad f_2(x, y) < g_2(x, y), \quad \dots$$

<sup>55</sup>A.Wiles, Modular elliptic curves and Fermat's last theorem. — Ann. Math., 1995, vol.141, p.443–551.

Вот, например, как строится рисунок 15, изображающий решения системы неравенств, которая обсуждалась в конце § 4:

```
In[100]:=InequalityPlot[Abs[x]+Abs[y]>1&& x^2+x*y+y^2<1,
                        {x,-3,3},{y,-3,3},
                        AspectRatio->1,
                        PlotStyle->AbsoluteThickness[1.5],
                        AxesStyle->AbsoluteThickness[1],
                        PlotLabel->"Figure 15: Abs[x]+Abs[y]>1&& x^2+x*y+y^2<1"]
```

Вот еще один совершенно замечательный пример, полученный незначительной обработкой примера, разобранный в пакете `InequalityGraphics`. А именно, на рисунке 16 построена таблица, состоящая из графиков решения неравенств  $|x|^p + |y|^q \leq 1$ . Напомним, что при  $p = q$  решения этого неравенства представляют собой шары по отношению к метрике Гельдера или, коротко, **шары Гельдера** = Hölder balls. Частными случаями метрики Гельдера являются

- городская метрика (при  $p = 1$ ),
- обычная евклидова метрика (при  $p = 2$ ),
- метрика Чебышева (при  $p = \infty$ ).

Хорошо видно, что в городской метрике шар является квадратом (как и следовало ожидать!), в евклидовой метрике — кругом (как и следовало ожидать!), а потом по мере того как  $p$  растет, его форма снова приближается к квадрату:

```
In[101]:=GraphicsArray[Table[InequalityPlot[Abs[x]^p+Abs[y]^q<=1,
                        {x,-1,1},{y,-1,1},
                        PlotStyle->AbsoluteThickness[1.5],
                        AxesStyle->AbsoluteThickness[1],
                        Ticks->None,
                        DisplayFunction->Identity],
                        {p,1,5},{q,1,5}],
                        PlotLabel->"Figure 16: Hoelder balls"]
```

Здесь используются две не встречавшиеся нам до сих пор команды.

- Вызванная в формате

```
GraphicsArray[{u,v,w,...}]
```

команда `GraphicsArray` собирает графические объекты  $u, v, w, \dots$  в один объект, состоящий из *одинаковых* прямоугольников, в которые вписаны объекты  $u, v, w, \dots$ . Вызванная в формате

```
GraphicsArray[{ {u,v,...}, {w,z,...}, ...}]
```

она делает то же самое, но при этом располагает прямоугольники, содержащие объекты  $u, v, w, z, \dots$ , в виде двумерной таблицы. В данном случае мы по отдельности строим 25 шаров, а потом соединяем их в один графический объект  $5 \times 5$ .

Мы могли бы, например, строить Рисунки 7–10 и 11–14 на одной странице при помощи команды `GraphicsArray`. Но фактически мы поступали иначе, пользуясь командами двумерной графики `Graphics` и `Rectangle`, вручную задавая размеры прямоугольников, в которые вписаны отдельные части картинки.

◦ Опция `DisplayFunction->Identity` встроена в тело `InequalityPlot` для того, чтобы подавить вывод промежуточных результатов на экран. По умолчанию

`DisplayFunction->DisplayFunction,`

при этом выводилась бы не только окончательная таблица из 25 шаров, но и каждый из них по отдельности, по мере вычисления соответствующего `InequalityPlot`. Обратите внимание, что значение этой опции распространяется на `InequalityPlot`, но не на `GraphicsArray`. Почему?

• **Простейшая двумерная графика.** Мы не будем пытаться сколь-нибудь систематически описывать здесь технику построения двумерных и, тем более, многомерных картинок, так как этому посвящена значительная часть Выпуска 4. Ограничимся поэтому построением двух знаменитых картинок. При этом мы не стремимся здесь дать наиболее короткую или эффектную программу, а приводим такое описание, где смысл каждой команды должен быть понятен начинающему. В действительности, мы обычно не вводим никаких координат руками, а вычисляем их (как таблицы, решения уравнений или что-нибудь в таком духе), после чего применяем к списку координат графические примитивы посредством конструкций типа `Map[Point,...]`, `Map[Line,...]` и т.д. Получающийся текст не намного короче, но имеет более прозрачную с точки зрения опытного пользователя структуру и при этом резко уменьшается вероятность ошибки.

Первый из приведенных текстов порождает изображение **плоскости Фано** — проективной плоскости над полем из двух элементов. Как хорошо известно, в терминах этой картинки описывается умножение семи мнимых единиц в **алгебре октав Кэли**:

```
In[102]:=rt=Sqrt[3]/2;
nodes={{0,0},{1,0},{2,0},{1,2*rt},
        {1,2*rt/3},{1/2,rt},{3/2,rt}};
Show[Graphics[{
  {AbsolutePointSize[5],
    Map[Point,nodes]},
  {AbsoluteThickness[1.5],
    Line[{{0,0},{2,0},{1,2*rt},{0,0}}],
    Line[{{0,0},{3/2,rt}}],
    Line[{{2,0},{1/2,rt}}],
    Line[{{1,2*rt},{1,0}}],
    Circle[{1,2*rt/3},2*rt/3]}]]]
```

Второй текст порождает рисунок из Альмагеста, изображающий построение правильного пятиугольника при помощи циркуля и линейки:

```
In[103]:=Show[Graphics[{
    {AbsolutePointSize[4],
      Point[{-1/2, 0}],
      Point[{-1/2+Sqrt[5]/2,0}],
      Point[{Cos[2*Pi/5],Sin[2*Pi/5]}],
      Point[{Cos[2*Pi/5],-Sin[2*Pi/5]}]},
    {AbsoluteThickness[0.5],
      Circle[{0,0}, 1],
      Circle[{-1/2,0],Sqrt[5]/2],
      Circle[{-1/2+Sqrt[5]/2,0}, 1]}},
  Axes->True]]
```

Мы не будем детально комментировать эти тексты, но большая часть того, что происходит, понятна сама по себе:

- о Команда **Graphics** соединяет определенные в ее теле графические примитивы в составной двумерный объект формата **Graphics**, который после этого обрабатывается как единое целое.

- о Команда **Show** отображает этот графический объект на экране. У начинающего в этом месте должен возникнуть вопрос, а что еще можно сделать с графическим объектом, кроме как отобразить его на экране? Ну, с ним можно *много чего* делать: вместо этого мы могли бы, например, сразу записать его в файл, включить его в определение другого графического объекта или мультипликации, преобразовать в другой формат, изменить настройки опций, применить к нему какое-то геометрическое преобразование и т.д.

Следующие три типа объектов называются **графическими примитивами**, при помощи них строятся различные элементы картинки, в данном случае точки, линии и окружности:

- о Примитив **точка** `Point[{x,y}]` определяет точку  $(x, y)$  с координатами  $x$  и  $y$ .

- о Примитив **линия** `Line[{u,v},{x,y}]` определяет отрезок с концами  $(u, v)$  и  $(x, y)$ . Вызванный с более длинным списком координат этот примитив порождает ломаную, состоящую из отрезков, соединяющих последовательные точки. Например, `Line[{u,v},{x,y},{z,w}]` состоит из *двух* отрезков, а именно, отрезка с концами  $(u, v)$  и  $(x, y)$  и второго отрезка с концами  $(x, y)$  и  $(z, w)$ . В случае, когда последняя точка списка совпадает с первой, мы получим замкнутую ломаную. Например,

```
Line[{u,v},{x,y},{z,w},{u,v}]
```

изображает треугольник с вершинами  $(u, v)$ ,  $(x, y)$  и  $(z, w)$ .

- о Примитив **окружность** `Circle[{x,y},r]` определяют окружности радиуса  $r$  с центром в  $(x, y)$ .

Кроме того, в системе имеется много других графических примитивов, например, **Rectangle**, **Disk** и **Polygon** для закрашенных фигур, **Raster**, **PostScript**, **Text** и т.д.



**Графические директивы**, примененные к соответствующим примитивам, задают значения параметров, которые применяются при их построении:

- Директива **абсолютный размер точки** `AbsolutePointSize[d]` задает (абсолютный) размер точек в типографских пунктах.
- Уже встречавшаяся нам при обсуждении графиков директива **абсолютная толщина** `AbsoluteThickness[d]` задает (абсолютную) толщину линий в типографских пунктах.

Обратите внимание на синтаксис!! Графические директивы (в отличие от опций, применяемых к объекту в целом) образуют список вместе с теми графическими примитивами, к которым они относятся. Это сделано для того, чтобы на одном и том же рисунке можно было рисовать точки разных размеров и линии разной толщины. Имеется большое количество других графических директив: `PointSize`, задающая (относительный) размер точек, `Dashing` и `AbsoluteDashing` для создания пунктирных линий, `GrayLevel`, `RGBColor`, `Hue` и т.д.

- Опция `Axes->True` предлагает включить в графический объект координатные оси.

• **Построение графиков функций двух переменных.** График функции двух переменных строится при помощи команды `Plot3D`, которая порождает объект формата `SurfaceGraphics`. В простейшем варианте эта команда вызывается в формате:

```
Plot3D[f[x,y],{x,a,b},{y,c,d}]
```

Эта команда порождает трехмерный график функции  $f(x,y)$  при  $x$  изменяющемся от  $a$  до  $b$ , и  $y$  изменяющемся от  $c$  до  $d$ .

В одном отношении использование команды `Plot3D` радикально отличается от использования команды `Plot`. А именно, `Plot` трактует список функций как предложение построить графики нескольких функций на одной картинке. При этом `Plot` может построить графики сколь угодно большого количества функций, это лимитируется только памятью Вашего компьютера и разрешением устройств вывода. В то же время первый аргумент команды `Plot3D` может быть либо *одной* функцией, либо списком из *двух* функций. Однако вызванная в формате

```
Plot3D[{f[x,y],g[x,y]},{x,a,b},{y,c,d}]
```

команда `Plot3D` будет воспринята отнюдь не как пожелание построить графики функций  $f$  и  $g$  на одной картинке. Нет, это пожелание построить график функции  $f$ , для которого **затенение** задается функцией  $g$ .

Приведем совсем простой пример использования команды `Plot3D`, в выпуске 4 мы вернемся к подробному описанию форматов трехмерной графики, структуры возникающих при этом объектов, используемых при этом опций, графических примитивов, etc. Воспроизведенный на Рисунке 19 график функции  $\cos(x^2 - y^2)$  построен при помощи следующей команды:

```
In[104]:=Plot3D[Cos[x^2-y^2],{x,-Pi,Pi},{y,-Pi,Pi},
```

```

Shading->False,
MeshStyle->AbsoluteThickness[1],
BoxStyle->AbsoluteThickness[0.8],
PlotPoints->60,
PlotLabel->"Figure 19: Cos[x^2-y^2]"

```

Чтобы понять, что здесь происходит, нужно знать, что графический объект формата `SurfaceGraphics` состоит из большого количества данных, самыми важными из которых являются:

- собственно **поверхность** с координатами  $(x, y, f(x, y))$ ,
- наброшенная на нее **сетка** или **мешок** `Mesh`,
- **затенение** или — в сочетании с подсветкой — **нюанс** `Shading`,
- **подсветка** `Lighting`,
- **ящик** `Box`.

Кроме того, в полное описание объекта этого типа входят еще **точка зрения** `ViewPoint`, **оси** `Axes`, **решетка** `Grid`, **текст** `Text` и т.д.

◦ Опция `Shading->False` убирает нюанс и введена *исключительно* для получения черно-белой картинки.

В природных условиях мы **никогда** не используем эту опцию. В действительности для получения изображения высокого качества на экране компьютера или цветном принтере мы поступаем прямо противоположным образом. А именно, в этом случае мы обычно убираем не нюанс, а сетку `Mesh->False`. При этом получается изображение, приближающееся по своим художественным достоинствам к лучшим холстам и доскам Дени, Дали и Дельво.

Использование следующих двух опций полностью аналогично использованию уже встречавшихся нам опций `PlotStyle` и `AxesStyle`. Они задают графические директивы, которые следует применять к соответствующим частям трехмерного графика:

- Опция `MeshStyle` описывает **стиль мешка**, в данном случае абсолютную ширину линий.
- Опция `BoxStyle` описывает **стиль ящика**, в данном случае снова мы меняем только абсолютную ширину линий.
- Опция `PlotPoints` показывает, в каком минимальном количестве точек следует вычислять значения функции. Вызванная в формате

```
PlotPoints->n
```

эта опция предписывает команде `Plot` вычислять значение функции *как минимум* в  $n$  точках — после чего продолжать вычислять значения в промежуточных точках для быстро осциллирующих функций. По умолчанию `PlotPoints->25`. Для команды `Plot3D` опция `PlotPoints` в формате `PlotPoints->n` предписывает вычислять значения в  $n$  точках по каждой из координат. Чтобы вычислять значения в  $m$  точках по  $x$  и в  $n$  точках по  $y$ , эту опцию следует задавать в формате

`PlotPoints->{m,n}`.

Для получения графики высокого качества мы часто задаем `PlotPoints->500` и даже `PlotPoints->1000`, что изначально требует вычисления *миллиона* значений функции и может занять 2–3 секунды. Понятно, что это имеет смысл только, если Вы собираетесь выводить эту картинку на такое устройство, которое допускает соответствующее разрешение, и только если одновременно убрать сетку, положив `Mesh->False`.

**Упражнение.** Постройте воспроизведенный на рисунке 20 график функции  $\cos(x^2 + y^2)$ .

Стоит подчеркнуть, что в объекте формата `SurfaceGraphics` хранится *гораздо* больше информации, чем в видимом изображении. Например, этот объект сохраняет и те части поверхности, которые при просмотре с данной точки зрения закрыты от зрителя другими частями. Однако эти части можно увидеть, если повернуть поверхность или изменить точку зрения. Более того, опция `HiddenSurface->False` делает поверхность прозрачной и, таким образом, все части поверхности, обычно закрытые другими ее частями, становятся видимыми.

На последней странице мы приводим платоновские тела — *and beyond*. В выпуске 4 мы как раз и ставим цель научить читателя строить подобные картинки — и получать от этого удовольствие.

## § 7. СУММЫ, ПРОИЗВЕДЕНИЯ, ПРЕДЕЛЫ

Рамануджан говорил, что богиня Намаккал внушала ему формулы во сне. Часто встав с кровати он (не) мог записать результаты и быстро проверить их . . . Рамануджан имел устойчивые религиозные взгляды. Он особенно почитал богиню Намаккал.

Сеши Нараяна Айар и Рамачандра Рао, “Жизнь Рамануджана”

Я меньше всего хочу, чтобы Вы вскидывали руки вверх и восклицали: “Здесь есть что-то необъяснимое, таинственное проявление древней восточной мудрости.” Не верю я в эту древнюю восточную мудрость.

Гарольд Годфри Харди<sup>56</sup>

Системы компьютерной алгебры полностью обесценивают **все** традиционные *навыки*, которым обучают в курсах математического анализа, высшей математики, теории дифференциальных уравнений. В этом и следующем параграфах мы увидим, что *Mathematica* умеет вычислять пределы, ряды, бесконечные произведения, дифференцировать, интегрировать, раскладывать в степенные и тригонометрические ряды, решать дифференциальные уравнения и уравнения в частных производных и т.д. Более того, мы готовы утверждать, что она делает все эти вещи лучше, чем **любой** неспециалист, и все еще лучше, чем подавляющее большинство профессиональных математиков.

<sup>56</sup>Г.Г.Харди, Двенадцать лекций о Рамануджане. — М., ИКИ, 2002, с.1–335; стр.13.

• **Суммы.** Одной из простейших и самых полезных команд системы является команда `Sum`. А именно, `Sum[f[i], {i, m, n}]` выражает сумму  $\sum_{i=m}^n f(i)$  значений функции  $f$  по  $i$  от  $m$  до  $n$  (с шагом 1). Обратите внимание на форму итератора `{i, m, n}`, которая используется в большинстве итеративных команд системы. Эта форма взята из языка `C`, остальные универсальные системы компьютерной алгебры, такие как `Axiom`, `Maple`, `MuPAD` используют `i=m..n` взятую из `Pascal`. По умолчанию шаг суммирования равен 1, однако задание итератора в форме `{i, m, n, d}` говорит, что  $i$  должно меняться от  $m$  до  $n$  с шагом  $d$ .

Многие алгебраические выражения, многочлены, дроби и тому подобное часто удобно выражать в виде сумм. Вот, скажем, как проще всего ввести многочлен Тэйлора экспоненты:

```
In[1]:=Sum[x^n/n!, {n, 0, 6}]
```

```
Out[1]=1+x+x^2/2+x^3/6+x^4/24+x^5/120+x^6/720
```

В § 1 мы уже обсуждали применение команды `Sum` для суммирования конкретных чисел. Однако это умеют делать и калькуляторы вроде `Mathcad`. Гораздо интереснее, что в `Mathematica` при помощи команды `Sum` можно производить **все** символьные манипуляции, которые могут встретиться начинающему, причем не только с конечными суммами, но и с бесконечными рядами!

• **Символьное вычисление сумм.** Совершенно замечательно, что суммы могут вычисляться не только в численном, но и в символьном виде, причем один или оба предела суммирования могут быть бесконечными! Что еще интереснее, пределы суммирования сами могут быть символами!!! Именно этим и отличаются интеллигентные системы `Maple` и `Mathematica` от бездушных калькуляторов наподобие `Matlab`.

Следующий совершенно удивительный пример показывает, что `Mathematica` *хорошо усвоила*<sup>57</sup>, что такое многочлены Бернулли:

```
In[2]:=Do[Print[Sum[i^m, {i, 1, n}]], {m, 1, 8}]
```

```
Out[2]=1/2*n*(1+n)
```

```
1/6*n*(1+n)*(1+2*n)
```

```
1/4*n^2*(1+n)^2
```

```
1/30*n*(1+n)*(1+2*n)*(-1+3*n+3*n^2)
```

```
1/12*n^2*(1+n)^2*(-1+2*n+2*n^2)
```

```
1/42*n*(1+n)*(1+2*n)*(1-3*n+6*n^3+3*n^4)
```

```
1/24*n^2*(1+n)^2*(2-4*n-n^2+6*n^3+3*n^4)
```

```
1/90*n*(1+n)*(1+2*n)*(-3+9*n-n^2-15*n^3+5*n^4+
```

<sup>57</sup>По причинам исторического и психологического плана мы предпочитаем избегать напрашивающегося выражения **понимает**, которое было бы использовано в случае успешного применения математической индукции в аналогичном контексте *человеческим* существом.

$$15*n^5+5*n^6)$$

**A parte:** еще бы ей этого не знать, если многочлены Бернулли и Эйлера являются основным инструментом суммирования полиномиальных рядов вообще.

Итеративная конструкция с `Do` и `Print` нам уже встречалась при обсуждении круговых многочленов. Первая из представленных здесь сумм, традиционно называется *суммой арифметической прогрессии* и изучается в школе, вторая и третья часто обсуждаются в математических кружках, но вот дальнейшие суммы, открытые в XVII веке Йоганном Фаульхабером и Яковом Бернулли<sup>58</sup>, известны главным образом только профессиональным математикам. Поскольку невозможно представить, что все подобные суммы содержатся в системе в табличной форме (ниже мы приводим еще несколько примеров), это значит, что система в какой-то форме владеет идеей математической индукции.

В действительности, следующий пример должен убедить каждого минимально знакомого с предметом в том, что *Mathematica* все же не знает на память большинство сумм, а по-настоящему их вычисляет, причем делает это совершенно иначе, чем это свойственно человеку. Вот сумма, которую каждый будущий профессиональный математик открыл к возрасту пяти лет:

```
In[3]:=Sum[i,{i,1,2*n-1,2}]
Out[3]=1+Floor[1/2*(-2+2*n)]+
      Floor[1/2*(-2+2*n)](1+Floor[1/2*(-2+2*n)])
```

Еще раз напомним, что вызов итератора в форме  $\{i, m, n, d\}$  означает суммирование по  $i$  от  $m$  до  $n$  с шагом  $d$ . Таким образом, мы предложили системе просуммировать все *нечетные* числа от 1 до  $2n-1$ . Видно, что здесь представлен не заученный ответ, а плод фактического (притом достаточно сурового) размышления. Разумеется, мы привыкли видеть этот ответ в несколько более прозаической форме, поэтому применим упрощение:

```
In[4]:=Simplify[Sum[i,{i,1,2*n-1,2}]]
Out[4]=Floor[n]^2
```

Поскольку мы изначально подразумевали (хотя пока и не сообщили системе), что число  $n$  целое, мы видим, что ценой небанального напряжения *Mathematica* сумела открыть формулу  $1 + 3 + \dots + (2n - 1) = n^2$ . В действительности в языке системы есть средства, которые позволяют декларировать, что  $n$  является целым числом (или, с точки зрения внутреннего языка *Mathematica*, имеет объектный тип `Integer`):

- обсуждаемая в Выпуске 2 спецификация доменов,
- обсуждаемая в Выпуске 3 спецификация паттернов.

Мы пока не изучали этих средств, относящихся к более высоким сферам программирования. Разумеется, после провозглашения  $n$  целым, система

<sup>58</sup>В.В.Прасолов, Многочлены. — М., МЦНМО, 2000, с.1–335; стр.135–137.

сразу вернула бы ответ в форме  $n^2$ . Однако в § 5 нам уже встречалась команды `Refine` и `Assuming`, которые позволяют упростить ответ при определенных предположениях. Итак, сделаем еще одну попытку:

```
In[5]:=Refine[Simplify[Sum[i,{i,1,2*n-1,2}],Element[n,Integers]]]
Out[5]=n^2
```

Понятно, что здесь произошло? Мы наконец-то внятно объяснили системе, что  $n$  целое. А именно, предположение `Element[n,Integers]` как раз и состоит в том, что  $n$  является элементом домена целых чисел `Integers`. разумеется, применив вначале `Refine` с условием `Element[n,Integers]`, а потом `Simplify`, мы тоже получили бы результат  $n^2$ .

Теперь, когда мы знаем, что `Mathematica` не вытаскивает ответы из каких-то загадочных таблиц, а порождает их силой чистого разума, предложим ей что-нибудь чуть более хитрое:

```
In[6]:=Sum[2^i/i!,{i,1,n}]
Out[6]=-1+E^2*(1+n)*Gamma[1+n,2]/Gamma[2+n]
```

Появляющаяся в числителе функция  $\Gamma(x, y)$  — это **неполная гамма-функция**:

$$\Gamma(x, y) = \int_y^{\infty} t^{x-1} e^{-t} dt,$$

в знаменателе стоит обычная гамма функция Эйлера  $\Gamma(x) = \Gamma(x, 0)$ . Вы действительно думаете, что и этот ответ можно извлечь из таблицы? И где то место, где может храниться *такая* Таблица?

• **Кратные суммы.** Кратная сумма вида  $\sum_{i=m}^n \sum_{j=k}^l f(i, j)$  выражается при помощи `Sum[f[i,j],{i,m,n},{j,k,l}]`. Здесь мы первый раз встречаемся с явлением, понимание которого чрезвычайно существенно для правильного использования итерационных команд вообще и команд формирования списков и работы с списками в особенности! Аналогичное соглашение действует для кратного интегрирования, построения графиков функций нескольких переменных и т.д. А именно, обратите внимание, что внутренние итераторы пишутся последними! Иными словами, `Sum[f[i,j],{j,k,l},{i,m,n}]` значит совершенно не то же самое, что `Sum[f[i,j],{i,m,n},{j,k,l}]`. В самом деле, при этой новой записи вначале происходит суммирование по  $i$ , а уже потом суммирование по  $j$ , в то время как в нашей исходной сумме вначале происходило суммирование по  $j$  и только потом суммирование по  $i$ . Разумеется, для *конечных* сумм, в том случае, когда пределы суммирования внутренних сумм сами не зависят от  $i$ , порядок итераторов можно изменить, значение суммы при этом не меняется — именно этот прием и называется **изменением порядка суммирования**. Однако для бесконечных сумм, а также в том случае, когда  $k$  и  $l$  сами являются функциями от  $i$ , ничего подобного делать нельзя.

Проиллюстрируем это явление на простом но важном примере. В компьютерной математике и анализе алгоритмов очень часто встречаются ча-

стичные суммы гармонического ряда

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \sum_{i=1}^n \frac{1}{i},$$

которые называются **гармоническими числами**. В *Mathematica* есть специальное обозначение для  $H_n$ , а именно `HarmonicNumber[n]`, однако в дальнейшем мы иногда будем делать вид, что этого не знаем. Гармонический ряд расходится — иными словами,  $H_n$  стремятся к бесконечности, хотя и очень медленно, порядка  $\ln(n)$ . Вот несколько первых гармонических чисел:

```
In[7]:=Table[HarmonicNumber[n],{n,1,12}]
```

```
Out[7]={1,3/2,11/6,25/12,137/60,49/20,363/140,761/280,
7129/2520,7381/2520,83711/27720,86021/27720}
```

Так как *Mathematica* знает определение гармонического числа, то вычисление `Sum[1/i,{i,1,n}]` дает `HarmonicNumber[n]`. В прежние годы в качестве упражнения на математическую индукцию мы часто предлагали студентам вычислить сумму первых гармонических чисел, т.е. сумму

$$H_1 + H_2 + \dots + H_m.$$

Разумеется, это один из тех случаев, когда формулировка задачи содержит подсказку, а именно, предложение *вычислить* сумму недвусмысленно указывает на то, что это *возможно*. А теперь сравните следующие два текста:

```
In[8]:=Sum[1/i,{n,1,m},{i,1,n}]
```

```
Out[8]=-m+HarmonicNumber[m]+m*HarmonicNumber[m]
```

```
In[9]:=Sum[1/i,{i,1,n},{n,1,m}]
```

```
Out[9]=m*HarmonicNumber[n]
```

Понятно, что происходит? В первой сумме осуществляется суммирование вначале по  $i$  от 1 до  $n$ , а потом по  $n$  от 1 до  $m$  и это как раз то, что мы хотели. С другой стороны, во второй сумме сначала осуществляется суммирование по  $n$  от 1 до  $m$ , а потом все эти  $m$  *одинаковых* сумм (ведь  $1/i$  не зависит от  $n$ ) суммируются по  $i$  от 1 до  $n$ .

Таким образом, если Вы не уверены, что отчетливо понимаете, что здесь происходит, то вот Вам добрый совет: оформляйте кратные суммы обычным в математике образом, с повторением команды `Sum`. Например, искомую сумму гармонических чисел можно еще выразить посредством

```
In[10]:=Sum[Sum[1/i,{i,1,n}],{n,1,m}]
```

Это чуть длиннее и менее элегантно с программистской точки зрения, зато не оставляет никакого места сомнению. Ясно, что здесь суммирование происходит вначале по  $i$  от 1 до  $n$  — это дает нам  $n$ -е гармоническое число  $H_n$  — а потом полученные результаты суммируются по  $n$  от 1 до  $m$ .

Уже весьма простые на вид кратные суммы при символьном вычислении часто приводят к достаточно замысловатым ответам. Попытка вычислить первую двойную сумму, которая приходит в голову,

```
In[11]:=Sum[1/(i+j),{i,1,m},{j,1,n}]
```

```
Out[11]=-PolyGamma[0,1+n]+PolyGamma[0,1+m+n]
```

сразу приводит нас к функции **дигамма**  $\psi(x) = \Gamma'(x)/\Gamma(x)$  (логарифмическая производная гамма-функции Эйлера). Ясно, что получение подобного ответа вручную в реальном времени лежит за пределами возможностей большинства профессиональных математиков, кроме избранных специалистов по комбинаторике, теории чисел, классическому анализу и математической физике.

• **Бесконечные суммы.** Mathematica умеет выражать многие бесконечные суммы как значения элементарных или специальных функций:

```
In[12]:=Sum[1/(n!x^n),{n,0,Infinity}]
```

```
Out[12]=E^(1/x)
```

```
In[13]:=Sum[1/n^3,{n,1,Infinity}]
```

```
Out[13]=Zeta[3]
```

```
In[14]:=Sum[1/(n!n^3),{n,1,Infinity}]
```

```
Out[14]=HypergeometricPFQ[{1,1,1,1},{2,2,2,2},1]
```

В первом случае мы видим экспоненту, во втором — значение  $\zeta(3)$  дзета-функции Римана, а фигурирующая в третьем ответе `HypergeometricPFQ` представляет собой обобщенную гипергеометрическую функцию, которая возникает в самых разных вопросах математики, от комбинаторики и теории чисел до математической физики.

• **Произведения.** Формат команды `Product` полностью аналогично формату `Sum`. А именно,

```
Product[f[i],{i,m,n}]
```

выражает произведение  $\prod_{i=m}^n f(i)$  значений функции  $f$  по  $i$  от  $m$  до  $n$  (с шагом 1). Как видим, здесь применяется обычная форма итератора  $\{i, m, n\}$ .

Как и в случае сумм, произведения могут использоваться для численных вычислений, такой пример мы уже приводили, и для короткой записи алгебраических выражений. Вот, например, как проще всего ввести **гауссов многочлен**, выражающий количество базисов в пространстве размерности  $n$  над полем из  $q$  элементов:

```
In[15]:=Product[q^n-q^i,{i,0,n-1}] /. n->10
```

```
Out[15]=(-1+q^10)(-q+q^10)(-q^2+q^10)(-q^3+q^10)(-q^4+q^10)
(-q^5+q^10)(-q^6+q^10)(-q^7+q^10)(-q^8+q^10)(-q^9+q^10)
```

Используемая в этом тексте замена `/. n->10` состоит из оператора `/.` или, в полной форме, `ReplaceAll` и правила подстановки `n->10`. Примененные вместе они побуждают систему заменить  $n$  на 10 всюду, на протяжении одного вычисления. Преимущество такой конструкции перед присваиванием



$n=10$  состоит в том, что  $n$  не приобретает постоянного значения, так что предшествующий текст может быть без всяких изменений включен в другое вычисление, в котором  $n$  имеет любое другое значение — или само является своим собственным значением, т.е. рассматривается как независимая переменная. Подробнее все нюансы, связанные с переменными и их значениями, а также использованием присваиваний и подстановок, обсуждаются в выпуске 2.

Разумеется, Mathematica не будет автоматически раскрывать скобки в символьном выражении, получающемся в результате образования произведения, чтобы принудительно раскрыть скобки в *конечном* произведении, нужно применить Expand или Distribute:

```
In[16]:=Product[1-x^i,{i,1,10}]
Out[16]=(1-x)(1-x^2)(1-x^3)(1-x^4)(1-x^5)
          (1-x^6)(1-x^7)(1-x^8)(1-x^9)(1-x^10)

In[17]:=Distribute[Product[1-x^i,{i,1,10}]]
Out[17]=1-x-x^2+x^5+x^7+x^11-x^12-x^13-x^14-2*x^15+x^18+x^19+
          x^20+x^21+3*x^22-x^25-x^26-2*x^27-2*x^28-x^29-x^30+
          3*x^33+x^34+x^35+x^36+x^37-2*x^40-x^41-x^42-x^43+
          x^44+x^48+x^50-x^53-x^54+x^55
```

Как и в случае сумм, один или оба предела в задании итератора могут быть бесконечными. Скажем, вычисление эйлеровского произведения

```
In[18]:=Product[(1-1/Prime[n]^2)^(-1),{n,1,Infinity}]
```

где Prime[n] обозначает  $n$ -е простое число, дает явное значение  $\pi^2/6$ .

• **Проверка на порогах.** Вот одно из наших излюбленных произведений, как из уважения к классикам XVIII века, так и конкретно с точки зрения сравнения интеллектуальных возможностей систем компьютерной алгебры и примазывающих к ним:

$$\prod_{i=1}^{2n} i^{(-1)^{i-1}} = \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n}$$

Предложите Вашей любимой системе *вычислить* это произведение. Достоинство этого теста состоит в простоте формулировки и убийственной убедительности ответа. Тот, кто понимает предмет, но не знаком с его историей, должен в этот момент испытывать чувство культурного шока: ожидаем ли мы, что система может *заметить*, что в знаменателе стоит порядок октаэдральной группы  $2^n n!$  —  $2^n n!$  IS THE ANSWER, WHAT IS THE QUESTION? — а в числителе  $(2n-1)!/2^{n-1}(n-1)!$ ? Иными словами, надемся ли мы увидеть что-то в духе

$$\prod_{i=1}^{2n} i^{(-1)^{i-1}} = \frac{(2n-1)!}{2^{2n-1} n! (n-1)!}?$$

Да нет же, мы уже говорили, что проводимые системами компьютерной алгебры вычисления основаны на совершенно других принципах, а РЕАЛЬНОСТЬ ПРЕВОСХОДИТ ВСЕ НАШИ ПРЕДСТАВЛЕНИЯ О НЕЙ. Если система компьютерной алгебры и сможет найти формулу для этого произведения, то это, конечно, будет совсем другая формула! Либо та же формула, но найденная *как-то иначе* и записанная в совершенно другом виде. Абсолютно принципиальный момент, который сразу очень многое объясняет в истории математики, состоит в том, что Mathematica СКОРЕЕ НАЙДЕТ СОВЕРШЕННО НЕВЕРОЯТНУЮ ФОРМУЛУ В ДУХЕ ЭЙЛЕРА И РАМАНУДЖАНА, ЧЕМ ТУ ПРОСТУЮ ФОРМУЛУ, КОТОРУЮ НАПИСАЛ БЫ СОВРЕМЕННЫЙ МАТЕМАТИК. Ясно, что никакой мистики здесь нет, она действует тем способом, который с ее точки зрения является наиболее экономным, разумным и простым! Это всего лишь значит, что ее представление об экономии мысли отличается от нашего!!!

В любом случае, ценой небанального напряжения мысли (больше секунды раздумий!) Mathematica находит общую формулу. Для полного реализма возьмем произведение до  $n$ , а не до  $2n$ , чтобы не оставалось уже никаких сомнений, что начиная вычисление система не может знать ответа:

```
In[19]:=Product[i^((-1)^(i-1)),{i,1,n}]
Out[19]=E^((-1/2)*(-Log[2]+(-1)^n*Log[2]+Log[Pi])+
          (-1)^n*(Log[Gamma[(1+n)/2]]-Log[Gamma[(2+n)/2]]))
```

Ну как, все еще из Таблицы? Да, но почему тогда у Maple 9.0 нет доступа к этой Таблице? Взяв теперь произведение до  $2n$ , мы придем к формуле, которая хотя и эквивалентна написанной выше, но выражена гораздо более экстравагантно:

```
In[20]:=Product[i^((-1)^(i-1)),{i,1,2*n}]
Out[20]=Gamma[1/2+n]/(Sqrt[Pi]*Gamma[1+n])
```

Именно в такой форме, конечно, ее и использовал Рамануджан. Однако в случае компьютера мы *почему-то* твердо уверены, что эта новая формула не внушена богиней Намаккал, а получена в результате честного (и довольно трудного) вычисления — НЕ ВЕРЮ Я В ЭТУ ДРЕВнюю ВОСТОЧную МУДРОСТЬ! Если нам трудно представить, что подобное вычисление может быть проведено человеком, то это только потому, что наше собственное математическое воспитание проходило под девизом ПРОСТОТЫ И НЕИЗБЕЖНОСТИ.

И теперь у нас, конечно, не может быть сомнений, что Mathematica сумеет перейти к пределу в предыдущей формуле:

```
In[21]:=Product[i^((-1)^i),{i,1,Infinity}]
Out[21]=Sqrt[Pi/2]
```

В действительности, она, конечно, не использует здесь предыдущую формулу, так как вычислить конечное произведение *намного* сложнее.

• **Вычисление пределов.** Основной командой для вычисления пределов в Mathematica является Limit. А именно, для вычисления  $\lim_{x \rightarrow c} f(x)$  эта

команда вызывается в следующем экзотическом формате:

```
Limit[f[x], x->c].
```

Обратите внимание, что в угоду традиционному математическому обозначению значение параметра  $c$  задается так, как будто это опция! В действительности, конечно, было бы правильно поступать ровно наоборот, а именно, аналитикам следовало бы предел функции  $f$  точке  $c$  записывать не как  $\lim_{x \rightarrow c} f(x)$ , а просто как  $\lim_c(f)$ , ведь на самом деле здесь никто никуда не переходит, никто ни к кому не стремится, и никакой буковки  $x$  в выражении **предел функции  $f$  точке  $c$**  нет и отродясь не было. Кроме того, запись  $f \mapsto \lim_c(f)$  подчеркивала бы аналогию с другими гомоморфизмами кольца функций, в которых  $c$  выступает в качестве параметра, скажем, с гомоморфизмом эвалюации  $f \mapsto \text{ev}_c(f) = f(c)$ . Но чего только не сделаешь ради поддержания самой абсурдной традиции!

Однако подобные глупости не мешают системе правильно вычислять пределы, причем в символьном виде:

```
In[22]:=Limit[(Cos[m*x]-Cos[n*x])/(x^2), x->0]
```

```
Out[22]=1/2*(-m^2+n^2)
```

Команда `Limit` ищет предел и в тех случаях, когда он не является единственным:

```
In[23]:=Limit[Sin[1/x], x->0]
```

```
Out[23]=Interval[{-1,1}]
```

Разумеется, к точно такому же результату приведет и вычисление

```
In[24]:=Limit[Sin[x], x->Infinity]
```

Для вычисления левого предела  $\lim_{x \rightarrow c-} f(x)$  и правого предела  $\lim_{x \rightarrow c+} f(x)$  команда `Limit` вызывается в следующих форматах:

```
Limit[f[x], x->c, Direction->1]
```

```
Limit[f[x], x->c, Direction->-1].
```

Однако нам встречались случаи, когда система не в состоянии найти предел в символьном виде. В этом случае можно использовать команду `NLimit`, содержащуюся в пакете `NumericalMath`NLimit``.

## § 8. ДИФФЕРЕНЦИРОВАНИЕ

DON'T DRINK AND DERIVE.

Motto of the society: Mathematicians Against Drunk Deriving

В `Mathematica` имеются две команды символьного дифференцирования `D` и `Derivative`, при помощи которых можно явно вычислить все производные, которые могут встретиться любому нематематику.

• **Дифференцирование.** Дифференцирование представляет собой чисто алгоритмическую операцию и производная любой элементарной функции может быть вычислена при помощи чисто механических манипуляций.

В системе *Mathematica* имеются две основные команды для вычисления производной, а именно,

- функция вычисления производной *D*, которая находит *значение* производной;
- оператор дифференцирования *Derivative[1]*, в операторной форме *'*, который вычисляет производную *как функцию*.

В простейшем виде вычисление производной имеет формат *D[f[x],x]*, где *f[x]* — функция, которую мы хотим продифференцировать, а *x* — переменная, по которой производится дифференцирование. Чтобы убедиться в том, что мы правильно понимаем использование команды *D*, продифференцируем семь функций, производные которых мы хорошо знаем, а именно, функцию  $x \mapsto x^n$ , а также функции *exp*, *ln*, *cos*, *sin*, *arcsin* и *arccos*. Следующее вычисление, как раз, и состоит в нахождении этих семи производных:

```
In[25]:=Map[D[# [x], x]&, {#^n&, Exp, Log, Cos, Sin, ArcSin, ArcCos}]
Out[25]={n*x^(-1+n), E^x, 1/x, -Sin[x], Cos[x],
        1/Sqrt[1-x^2], -1/Sqrt[1-x^2]}
```

Тому, кто раньше не имел дела с системами компьютерной алгебры, формат этого вычисления может показаться несколько странным, поэтому прокомментируем наиболее важные моменты. Во-первых, мы вычисляем *список* производных. Это совершенно необходимо, если мы хотим провести несколько вычислений в одной клетке. Для этого мы приводим список функций, которые хотим продифференцировать, и применяем к *элементам* этого списка дифференцирование *D[# [x], x]&* при помощи функции *Map*. Дифференцирование  $f \mapsto f'$  задается здесь в формате **анонимной функции**, *D[# [x], x]&*. Напомним, что при этом *Slot*, в операторной записи *#*, обозначает *аргумент* чистой функции, в данном случае неизвестную функцию от *x*, которую мы хотим продифференцировать, а *Function*, в операторной записи *&*, обозначает *применение* чистой функции, в данном случае функции дифференцирования по *x*. Обратите внимание, что в нашем примере аргумент анонимной функции имеет вид *# [x]*, а не просто *#*, так что эта неизвестная функция рассматривается именно как функция от того самого *x*, по которому производится дифференцирование.

Может быть чуть проще понять использование этой конструкции во втором примере, когда мы задаем в таком формате функцию возведения в *n*-ю степень. А именно, с точки зрения *Mathematica* выражение *#^n&* является *именем* функции  $t \mapsto t^n$  в том же самом смысле, в котором *sin* является *именем* функции  $x \mapsto \sin(x)$ . Используемый формат имени без явного именованного аргументов называется форматом *анонимной функции*. При некотором навыке использование анонимных функций становится одним из основных средств программирования в *Mathematica*. Однако начинающему, вероятно, гораздо удобнее использовать формат *чистой функции*, *Function[t,t^n]* или, в полной форме *Function[t,Power[t,n]]*, в котором явно указывается, что куда переходит. В этом формате *D[# [x], x]&* выглядит как *Function[f,D[f[x],x]]*, где *явно* указано, что функции *f*

сопоставляется ее производная по  $x$ .

Таким образом, вычисление `Function[f,D[f[x],x]][Cos]` даст нам `-Sin[x]`, как и просто вычисление `D[Cos[x],x]`. Чуть более витиеватый формат понадобился нам для того, чтобы вычислить одновременно производные нескольких функций — и то только потому, что мы не знали настоящего имени дифференцирования. В действительности, оператор дифференцирования имеет имя, он называется `Derivative[1]` или, в операторной записи, `'`. Таким образом, мы могли бы провести то же вычисление с использованием команды `Derivative[1]` вместо команды `D`:

```
In[26]:=Map[#&,{#^n&,Exp,Log,Cos,Sin,ArcSin,ArcCos}]
Out[26]={n#1^(-1+n)&,E^#1&,1/#1&,-Sin[#1]&,
        Cos[#1]&,1/Sqrt[1-#1^2]&,-1/Sqrt[1-#1^2]&}
```

Разумеется, первая строчка является просто сокращением строчки

```
In[27]:=Map[Derivative[1],{#^n&,Exp,Log,Cos,Sin,ArcSin,ArcCos}]
```

вычисление которой даст точно такой же результат. Мы видим, что в этом случае вычисляются не *значения* производных, а сами производные, *как функции*. Обратите внимание на появляющееся здесь вместо `#` обозначение аргумента через `#1`. Дело в том, что если у анонимной функции несколько аргументов, то они обозначаются `#1`, `#2`, `#3` и так далее. В данном случае программа не знает, в составе какого более сложного выражения мы собираемся вызвать эти производные, и нумерует их аргументы с тем, чтобы при появлении следующего аргумента дать ему имя `#2` и т.д.

• **Kettenregel, Produktregel, usw.** Разумеется, *Mathematica* знает все основные правила вычисления производных, причем следующее вычисление показывает, что она может применять их в символьном виде к неизвестным функциям. Вот как с ее точки зрения выглядят формулы дифференцирования композиции, произведения и частного двух функций, известные под народными названиями *Kettenregel*, *Produktregel* и *Quotientenregel*, которые дал им фон Лейбниц:

```
In[28]:= {D[f@g[x],x],D[f[x]*g[x],x],D[f[x]/g[x],x],}
Out[28]={f'[g[x]]*g'[x],g[x]*f'[x]+f[x]*g'[x],
        f'[x]/g[x]-f[x]*g'[x]/f[x]^2}
```

Обратите внимание, что  $f$  и  $g$  обозначают здесь произвольные функции. Разумеется, тем более *Mathematica* сможет применить эти формулы к конкретным функциям.

**Комментарий.** Первая из этих формул — это **формула дифференцирования композиции**, которая по-русски испокон веку называлась **цепным правилом** (*Kettenregel*, *chain rule*). К сожалению в последние десятилетия под влиянием украинского языка в русской учебной литературе по так называемой **высшей математике** укоренился полонизм **дифференцирование сложной функции**. В польском языке это словообразование вполне оправдано, так как композиция функций называется там **сложением** (ну а сложение функций, естественно, **додаванием**). Но в русском языке эта малограмотная калька, впервые появившаяся в украинском переводе учебника Банаха, ничем не подкреплена и ее употребление свидетельствует либо о преступной некомпетентности (если писатель

не знает о происхождении этого выражения из польского *funkcja złożona*), либо, в противном случае, о чудовищном манеризме. Кстати, в старых русских учебниках и сама композиция функций называлась *по-простому функцией от функции* (*fonction de fonction*), что, конечно, тоже гораздо лучше, чем мифические *сложные* функции.

Следующий диалог иллюстрирует разницу между  $f^{-1}$  и  $1/f$ :

```
In[29]:={D[InverseFunction[f][x],x],D[1/f[x],x]}
Out[29]={1/f'[f^(-1)[x]],-f'[x]/f[x]^2}
```

При большом желании можно даже выяснить, когда  $f^{-1} = 1/f$  (*чрезвычайно* редко!)

Вот, кстати, **правило дифференцирования степени** (Potenzregel), которое уже далеко не всегда приводится в элементарных учебниках:

```
In[30]:=D[f[x]^g[x],x]
Out[30]=f[x]^g[x]*(g[x]*f'[x]/f[x]+Log[f[x]]*g'[x])
```

Следующий пример показывает, что система действительно полностью отдает себе отчет в том, как применяется цепное правило:

```
In[31]:={D[ArcCos[Log[x]],x],D[Log[ArcCos[x]],x]}
Out[31]={1/(x*Sqrt[1-Log[x]^2]),-1/(Sqrt[1-x^2]*ArcCos[x])}
```

После этого ясно, что любое другое подобное вычисление является просто упражнением в терпении. Вот пример из тех, которые преподаватели **высшей математики** любят давать на зачетах:

```
In[32]:=D[Log[ArcCos[x]]/ArcCos[Log[x]],x]
Out[32]=-1/(Sqrt[1-x^2]*ArcCos[x]*ArcCos[Log[x]])+
Log[ArcCos[x]]/(x*ArcCos[Log[x]]^2*Sqrt[1-Log[x]^2])
```

• **Высшие производные.** Понятно, что значение второй производной  $f''(x)$  функции  $f$  по  $x$  можно вычислить при помощи  $D[D[f[x],x],x]$ , однако в языке системы предусмотрены и две более короткие записи этого выражения, а именно  $D[f[x],x,x]$  и  $D[f[x],\{x,2\}]$ . В действительности внутренним образом все эти три выражения интерпретируются абсолютно одинаково, а именно как  $\text{Derivative}[2][f][x]$ , т.е. как *значение*  $\text{Derivative}[2][f]$  в точке  $x$ . В свою очередь  $\text{Derivative}[2][f]$  обозначает собственно вторую производную  $f''$ , рассматриваемую как *функцию*. Ну и, наконец,  $\text{Derivative}[2]$  обозначает *дифференциальный оператор*, сопоставляющий функции ее вторую производную. Именно для того, чтобы обращаться по имени к этому оператору Mathematica и интерпретирует  $\text{Derivative}$  как функцию *одного* аргумента, а именно, *порядка!!!* У функции  $\text{Derivative}[n]$  тоже один аргумент, а именно *функция*,  $n$ -ю производную которой мы хотим вычислить. Наконец, у функции  $\text{Derivative}[n][f]$  снова один аргумент, а именно, *точка*, в которой мы вычисляем значение  $f^{(n)}$ . В Части 2 мы подробнейшим образом обсуждаем, чем  $f[x][y]$  отличается от  $f[x,y]$  и от  $f[\{x,y\}]$  и почему в разных ситуациях используются разные форматы. Однако уже сейчас читатель должен запомнить, что вызов  $\text{Derivative}$  в формате  $\text{Derivative}[n,f,x]$  является грубейшей синтаксической ошибкой.

Вот, скажем, чему равна вторая производная  $x^{x^x}$ :

```
In[33]:=Simplify[D[x^x^x,x,x]]
Out[33]=x^(-2+x+x^x)(-1+2*x+x^x+x*(3+x+2*x^x)*Log[x]+
x*(2*x+2*x^x+x^(1+x))*Log[x]^2+
x^2*(1+2*x^x)*Log[x]^3+x^(2+x)*Log[x]^4))
```

Мы применили функцию `Simplify`, чтобы система привела получающееся после применения всех обычных правил выражение для производной к чуть более простому виду, вынеся общий множитель и т.д. Напомним, что система не применяет дистрибутивность автоматически и ей нужно явно подсказывать сделать это. Кстати, обратите внимание и на то, что по умолчанию для функции `Power` используется **правая группировка**, так что  $x^y^z$  истолковывается как  $x^{(y^z)}$ , а вовсе не как  $(x^y)^z$ . Чтобы получить  $(x^y)^z$ , необходимо *явным образом* ставить скобки!!! Вычислим для сравнения вторую производную  $(x^x)^x$ :

```
In[34]:=Simplify[D[(x^x)^x,x,x]]
Out[34]=(x^x)^x*(3+2*Log[x]+(x+x*Log[x]+Log[x^x])^2)
```

Теперь уже совершенно ясно, что  $D[f[x], x, x, x]$  и  $D[f[x], \{x, 3\}]$  выражает значение  $f'''(x)$  третьей производной  $f'''$  функции  $f$  по  $x$ , и интерпретируется как `Derivative[3][f][x]`. В свою очередь функция  $f'''$  вычисляется как `Derivative[3][f]`. Смысл `Derivative[n][f]` и  $D[f[x], \{x, n\}]$  теперь уже совершенно очевиден.

Разумеется, `Mathematica` умеет применять дифференцирования высших порядков в символьном виде к неизвестным функциям. Вот несколько первых примеров **формулы Фаа ди Бруно**<sup>59</sup> дифференцирования композиции:

```
In[35]:=ColumnForm[NestList[D[# ,x]&,f@g[x],4]]
Out[35]=f[g[x]]
f'[g[x]]*g'[x]
g'[x]^2*f''[g[x]]+f'[g[x]]*g''[x]
3*g'[x]*f''[g[x]]*g''[x]+g'[x]^3*f'''[g[x]]+f'[g[x]]*g'''[x]
3*f''[g[x]]*g''[x]^2+6*g'[x]^2*g''[x]*f'''[g[x]]+
4*g'[x]*f''[g[x]]*g'''[x]+g'[x]^4*f^(4)[g[x]]+
f'[g[x]]*g^(4)[x]
```

Как обычно,  $f^{(n)}(x)$  обозначает значение  $n$ -й производной  $\frac{d^n f}{dx^n}$ .

• **Частные производные.** Точно такой же формат как для функций одной переменной имеет и вычисление частных производных по одной из переменных. А именно,  $D[f[x, y, z], \{x, n\}]$  обозначает  $\frac{\partial^n f}{\partial x^n}$ . С другой стороны,  $D[f, x_1, x_2, x_3]$  обозначает смешанную производную  $\frac{\partial}{\partial x_1} \frac{\partial}{\partial x_2} \frac{\partial f}{\partial x_3}$  —

<sup>59</sup> Дональд Кнут предложил называть эту формулу **формулой Арбогаста**, чем очень огорчил официальный Ватикан.

обратите внимание на порядок переменных!! Напомним, что в такой форме мы обращаемся к значениям этих производных. Собственно сама частная

производная  $\frac{\partial^{n_1}}{\partial x_1^{n_1}} \frac{\partial^{n_2}}{\partial x_2^{n_2}} \frac{\partial^{n_3}}{\partial x_3^{n_3}} f$ , рассматриваемая как функция, вызывается посредством `Derivative[n1,n2,n3][f]`. Вот простейший пример

```
In[36]:={D[Exp[x^2+y^2],x,x],D[Exp[x^2+y^2],x,y],D[Exp[x^2+y^2],y,y]}
Out[36]={2*E^(x^2+y^2)+4*E^(x^2+y^2)*x^2,
         4*E^(x^2+y^2)*x*y,
         2*E^(x^2+y^2)+4*E^(x^2+y^2)*y^2}
```

Как всегда, `Mathematica` может считать в символьном виде с неизвестными функциями. Вот, скажем, как она понимает цепное правило для функций двух переменных. Следующие формулы можно найти на первых страницах любого учебника по анализу функций нескольких переменных (multivariable calculus):

```
In[37]:=Simplify[D[f[g[x,y],h[x,y]],x]],
Out[37]={f^(0,1)[g[x,y],h[x,y]]*h^(0,1)[x,y]
        g^(0,1)[x,y]*f^(1,0)[g[x,y],h[x,y]],
In[37]:=Simplify[D[f[g[x,y],h[x,y]],y]]
Out[37]=f^(1,0)[g[x,y],h[x,y]]*g^(1,0)[x,y]
        f^(0,1)[g[x,y],h[x,y]]*h^(1,0)[x,y]
```

Как обычно, через  $f^{(i,j)}(x,y)$  здесь обозначена функция  $\frac{\partial^i}{\partial x^i} \frac{\partial^j}{\partial y^j} f(x,y)$ . А вот соответствующее правило для вторых производных большинство читателей уже вряд ли видело в явном виде.

```
In[38]:=Simplify[D[f[g[x,y],h[x,y]],x,x]]
Out[38]=f^(0,2)[g[x,y],h[x,y]]*h^(1,0)[x,y]^2
        2*g^(1,0)[x,y]*h^(1,0)[x,y]*f^(1,1)[g[x,y],h[x,y]]+
        g^(1,0)[x,y]^2*f^(2,0)[g[x,y],h[x,y]]+
        f^(1,0)[g[x,y],h[x,y]]*g^(2,0)[x,y]
        f^(0,1)[g[x,y],h[x,y]]*h^(2,0)[x,y]
In[39]:=Expand[D[f[g[x,y],h[x,y]],x,y]]
Out[39]=h^(0,1)[x,y]*f^(0,2)[g[x,y],h[x,y]]*h^(1,0)[x,y]
        h^(0,1)[x,y]*g^(1,0)[x,y]*f^(1,1)[g[x,y],h[x,y]]
        g^(0,1)[x,y]*h^(1,0)[x,y]*f^(1,1)[g[x,y],h[x,y]]
        f^(1,0)[g[x,y],h[x,y]]*g^(1,1)[x,y]
        f^(0,1)[g[x,y],h[x,y]]*h^(1,1)[x,y]
        g^(0,1)[x,y]*g^(1,0)[x,y]*f^(2,0)[g[x,y],h[x,y]]
In[40]:=Simplify[D[f[g[x,y],h[x,y]],y,y]]
Out[40]=h^(0,1)[x,y]^2*f^(0,2)[g[x,y],h[x,y]]
        f^(0,1)[g[x,y],h[x,y]]*h^(0,2)[x,y]+
        g^(0,2)[x,y]*f^(1,0)[g[x,y],h[x,y]]+
        2*g^(0,1)[x,y]*h^(0,1)[x,y]*f^(1,1)[g[x,y],h[x,y]]+
        g^(0,1)[x,y]^2*f^(2,0)[g[x,y],h[x,y]]
```



Явно написать аналог формулы Фаа ди Бруно для высших производных композиции функций нескольких переменных, ничего при этом не пропустив, определенно находится за пределами комбинаторных способностей обычного человека. Например, уже в  $D[f[g[x,y],h[x,y]]x,x,y,y]$  входит 46 слагаемых, каждое из которых является произведением двух, трех, четырех или пяти частных производных.

• **Ряд Тэйлора.** Основной командой для вычисления степенных разложений функции  $f$  является `Series`. Вызванная в формате

```
Series[f[x],{x,c,m}]
```

эта команда находит первые  $m$  членов ряда Тэйлора функции  $f$  от  $x$  в окрестности точки  $c$ . В тех случаях, когда `Mathematica` может фактически вычислить производные, она вычисляет их, в противном случае — оставляет их в символьной форме:

```
In[41]:=Series[f[x],{x,c,4}]
```

```
Out[41]=f[c]+f'[c]*(x-c)+1/2*f''[c]*(x-c)^2+
1/6*f'''[c]*(x-c)^3+1/24*f^(4)[c]*(x-c)^4+O[x-c]^5
```

В тех случаях, когда команда `Series` не может найти ряд Тэйлора (например, если значения каких-то производных в точке  $c$  бесконечны), она пишет разложение функции в ряд, в который входят также отрицательные и/или дробные степени  $x - c$ :

```
In[42]:=Series[Exp[Sqrt[x]],{x,0,3}]
```

```
Out[42]=1+x^(1/2)+1/2*x+1/6*x^(3/2)+1/24*x^2+
1/120*x^(5/2)+1/720*x^3+O[x]^(7/2)
```

## § 9. ИНТЕГРИРОВАНИЕ

Are you an Analyst? — I am an Oralist.

George Bergmann

$$\int_1^{\sqrt[3]{3}} z^2 dz \cos(3\pi/9) = \ln(\sqrt[3]{3})$$

Anonymous Analyst<sup>60</sup>

Одной из самых мощных команд системы является команда `Integrate`, при помощи которой вычисляются как неопределенные, так и определенные интегралы от функций одной и нескольких переменных. Напомним, что имплементация команды `Integrate` включает 600 страниц кода на `C` и еще около 500 страниц высокоуровневого кода!

• **Неопределенный интеграл.** Первообразная функции  $f$  вычисляется при помощи команды `Integrate`, вызываемой в том же формате, что и команда дифференцирования `D`, а именно, `Integrate[f[x],x]`. Эта команда

<sup>60</sup>Для удобства **оралистов** приведем американское чтение этого лимерика: Integral  $z$ -squared  $dz$ / from 1 to the cube root of 3/ times the cosine/ of three  $\pi$  over 9/ equals log of the cube root of  $e$ .

возвращает *какую-то* первообразную функции  $f$ . Таким образом, взятие неопределенного интеграла считается операцией обратной дифференцированию:

```
In[43]:=D[Integrate[f[x],x],x],Integrate[D[f[x],x],x]
Out[43]={f[x],f[x]}
```

Вот пара простеньких интегралов, которые легко берутся интегрированием по частям:

```
In[44]:={Integrate[ArcTan[x],x],Integrate[ArcSin[x],x]}
Out[44]={x*ArcTan[x]-1/2*Log[1+x^2],Sqrt[1-x^2]+x*ArcSin[x]}
```

Конечно, система может легко взять любой стандартный интеграл, для вычисления которого не требуется ничего, кроме многократного применения обычных формул для элементарных функций и стандартных подстановок:

```
In[45]:=Simplify[Integrate[Cos[x]^10,x]]
Out[45]=1/10240*(2520*x+2100*Sin[2*x]+600*Sin[4*x]+
150*Sin[6*x]+25*Sin[8*x]+2*Sin[10*x])
In[46]:=Integrate[(x+Sqrt[x^2+1])/(x+1-Sqrt[x^2+1]),x]
Out[46]=1/2*(x+x^2+Sqrt[1+x^2]+x*Sqrt[1+x^2]+
ArcSinh[x]+2*Log[x]-Log[1+Sqrt[1+x^2]])
```

Появление гиперболического арксинуса во втором интеграле совершенно естественно, если вспомнить, что  $\int \frac{1}{\sqrt{1+x^2}} dx = \operatorname{arcsinh}(x) + c$ . Тем не менее, выводить вручную даже такие совсем простые формулы не слишком приятно.

• **Неберущиеся интегралы.** В отличие от вычисления производных вычисление первообразных представляет собой творческое занятие. Тонкий момент здесь состоит в том, что интеграл от элементарной функции уже совершенно не обязательно является элементарной функцией, но никаких очевидных критериев, которые позволяют сказать, когда подобная неприятность случается, нет. Уже композиция двух основных элементарных функций может не иметь элементарного интеграла. Вот несколько совсем простеньких примеров, которые хорошо иллюстрируют возникающие здесь проблемы.

◦ Интеграл  $\int \sin(\ln(x)) dx$  легко берется интегрированием по частям. Каждый из нас должен был проделать следующее вычисление на первом курсе:

```
In[47]:=Integrate[Sin[Log[x]],x]
Out[47]=(1/2)*x*Cos[Log[x]]+(1/2)*x*Sin[Log[x]]
```

◦ Но вот переставив здесь Sin и Log, мы получим интеграл  $\int \ln(\sin(x)) dx$ , в который входит полилогарифм:

```
In[48]:=Integrate[Log[Sin[x]],x]
Out[48]=(-x)*Log[1-E^(2*I*x)]+x*Log[Sin[x]]+
```

$$(1/2)*I(x^2+PolyLog[2,E^{(2*I*x)}])$$

где  $PolyLog[n, x]$  обозначает **полилогарифм**  $Li_n(x)$ , который определяется как

$$Li_n(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^n}.$$

о Также и функция  $\ln^2(x) = \ln(\ln(x))$  не интегрируется в элементарных функциях. А именно, интеграл  $\int \ln^2(x) dx$  равен

In[49]:=Integrate[Log[Log[x]],x]

Out[49]=x\*Log[Log[x]]-LogIntegral[x]

Здесь  $LogIntegral[x]$  обозначает **интегральный логарифм**  $li(x)$ , который можно определить как

$$li(x) = \int_0^x \frac{dt}{\ln(t)}.$$

о А вот  $\int \sin^2(x) dx$  вообще никак не выражается в терминах более известных функций, так что если Вы предложите Mathematica вычислить

In[50]:=Integrate[Sin[Sin[x]],x]

она после некоторого раздумья просто перепишет это выражение. Это значит, что системе не известно никакой более простой или более явной формы для  $Integrate[Sin[Sin[x]],x]$ . Мы не сомневаемся, что к этому моменту читатель уже в состоянии отличить неберущийся интеграл  $\int \sin^2(x) dx$  от интеграла  $\int \sin(x)^2 dx = x/2 - \sin(2x)/4$ .

о Чтобы завершить тему с возведением в квадрат всего, что в  $\sin(x)$  возводится в квадрат, вычислим еще интеграл  $\int \sin(x^2) dx$ . С точностью до нормировки ответ называется **синус интегралом Френеля**:

In[51]:=Integrate[Sin[x^2],x]

Out[51]=Sqrt[Pi/2]\*FresnelS[Sqrt[2/Pi]\*x]

Этот интеграл, как и **косинус интеграл Френеля**  $\int \cos(x^2) dx$  или, после нормировки,  $\int \cos(\pi x^2/2) dx$ , часто встречается в оптике.

Да чего там композиции — уже произведения двух основных элементарных функций совершенно не обязательно интегрируются в элементарных функциях (произведение двух производных вообще не обязано быть производной чего бы то ни было). Вот два совсем простеньких примера. При попытке вычислить  $\int \frac{x}{\sin(x)} dx$  встречается уже знакомый нам полилогарифм

In[52]:=Integrate[x/Sin[x],x]

Out[52]=x\*(Log[1-E^{(I\*x)}]-Log[1+E^{(I\*x)}])+  
I\*(PolyLog[2,-E^{(I\*x)}]-PolyLog[2,E^{(I\*x)}])

А вот  $\int \frac{\sin(x)}{x} dx$  имеет специальное название — это **интегральный синус**:

```
In[53]:=Integrate[Sin[x]/x,x]
Out[53]=SinIntegral[x]
```

Через `SinIntegral[x]` и `CosIntegral[x]` в *Mathematica* как раз и обозначаются интегральный синус и **интегральный косинус**, равные, соответственно

$$\text{Si}(x) = \int_0^x \frac{\sin(t)}{t} dt, \quad \text{Ci}(x) = - \int_x^\infty \frac{\cos(t)}{t} dt,$$

Каждый может узнать все эти факты секунд за 30, со скоростью нужной, чтобы набрать эти интегралы на клавиатуре. Заметим, что большая часть этих интегралов вообще не упоминается ни в общем курсе математического анализа, ни в большинстве стандартных учебников. Впрочем, и в этом отношении замечательное руководство Владимира Антоновича Зорича<sup>61</sup> оказывается с *огромным* отрывом лучшим из всех учебных текстов на русском языке. А именно, в упражнениях 6 и 7 на стр.380–381 первого тома не только перечисляются все эти интегралы, но и объясняется их происхождение из **интегральной экспоненты**:

$$\text{Ei}(x) = \int_{-\infty}^x \frac{e^t}{t} dt,$$

на языке *Mathematica*, естественно, `ExpIntegralEi[x]`.

Поэтому, если Вы не являетесь специалистом по некоторым разделам анализа или математической физики, и не читали Зорича, то Вы, скорее всего, вообще не видели большинства этих интегралов. А теперь скажите, сколько времени Вам понадобилось бы, чтобы извлечь эту информацию из традиционного справочника, ну скажем, из<sup>62</sup>?

• **Определенный интеграл.** Определенный интеграл  $\int_a^b f(x) dx$  вычисляется при помощи той же самой команды `Integrate`, которая в этом случае вызывается в формате `Integrate[f[x],{x,a,b}]`, где явно указаны пределы интегрирования. *Mathematica* знает, как дифференцировать определенный интеграл по пределам интегрирования:

```
In[54]:=D[Integrate[f[t],{t,g[x],h[x]}],x]
Out[54]=-f[g[x]]*g'[x]+f[h[x]]*h'[x]
```

А вот формула Ньютона—Лейбница:

```
In[55]:=f[x.]:=D[g[x],x]; Integrate[f[x],{x,a,b}]
Out[55]=-g[a]+g[b]
```

Еще раз объясним, что здесь происходит. В строке ввода две команды, разделенные точкой с запятой. Первая из них — это операция отложенного присваивания `:=` при помощи которой мы задаем равенство  $f = g'$ . Как всегда при этом бланк \_ используется для того, чтобы сообщить системе, что

<sup>61</sup>В.А.Зорич, Математический анализ. т. I,II. — М., МЦНМО, 2002, с.1–657, с.1–787.

<sup>62</sup>А.П.Прудников, Ю.А.Брычков, О.И.Маричев, Интегралы и ряды. Элементарные функции. — Наука, 1981. с.1—798.

$f(x) = g'(x)$  для всех  $x$ . Вместо этого мы могли при желании произвести *немедленное* присваивание `f=Derivative[1][g]`, но в этом случае нужно задавать саму функцию  $f$ , а не ее значения! После этого вторая команда предлагает системе проинтегрировать  $f$ .

Только что сказанное убеждает нас в том, что система сможет взять любой определенный интеграл, если она может взять соответствующий неопределенный интеграл и значения первообразной на концах конечны. Гораздо интереснее, как она будет выкручиваться при возникновении неопределенностей или бесконечных значений! Оказывается, это тоже не представляет проблемы:

```
In[56]:=Integrate[Log[x],{x,0,1}]
```

```
Out[56]=-1
```

Мы уже знаем, что интеграл от функции  $\sin^2(x) = \sin(\sin(x))$  не берется. Тем не менее, некоторые определенные интегралы от этой функции берутся явно:

```
In[57]:=Integrate[Sin[Sin[x]],{x,0,Pi}]
```

```
Out[57]=Pi*StruveH[0,1]
```

Появляющаяся в этой формуле **функция Струве** `StruveH` является специальным решением неоднородного уравнения Бесселя (см. § 10).

Вот еще один не самый банальный интеграл, вычисление которого обычно сводится к многомерным интегралам, либо интегралам, зависящим от параметра:

```
In[58]:=Integrate[ArcTan[x]/(x*Sqrt[1-x^2]),{x,0,1}]
```

```
Out[58]=1/2*Pi*ArcSinh[1]
```

По форме ответа *кажется*, что *Mathematica* в лоб вычислила соответствующий неопределенный интеграл и использовала формулу Ньютона—Лейбница. Однако это абсолютно не так, поскольку вычислять неопределенный интеграл  $\int \frac{\arctg(x)}{x\sqrt{1-x^2}} dx$  она не умеет<sup>63!!!</sup> Таким образом, для нас остается загадкой, каким образом *Mathematica* получила правильный ответ в данном случае! Впрочем, даже зная этот ответ, неспециалист, скорее всего, далеко не сразу заметит, что это в точности  $\frac{\pi}{2} \ln(1 + \sqrt{2})$ .

• **Несобственные интегралы.** Одними из самых знаменитых и часто встречающихся в приложениях определенных интегралов являются **интеграл Эйлера—Пуассона** известный также как **Гауссов интеграл**<sup>64</sup> и

<sup>63</sup>Это один из *немногих* известных нам интегралов, которые *Mathematica* не умеет вычислять. Впрочем, в подобных случаях большой все равно должен обращаться к доктору, поскольку на стр.268 книги “Интегралы и ряды” этого интеграла тоже нет, а есть только более простой интеграл  $\int \frac{x \arctg(x)}{\sqrt{1-x^2}} dx$ . **Упражнение для профессионалов:** зная ответ придумайте подстановку, вычисляющую этот интеграл.

<sup>64</sup>В теории вероятностей интеграл  $\operatorname{erf}(x) = \int_0^x e^{-t^2} dt$ , рассматриваемый как функция от верхнего предела, называется еще **функцией ошибок** или **интегралом ошибок** = **error function**.

**интеграл Дирихле:**

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}, \quad \int_{-\infty}^{\infty} \frac{\sin(x)}{x} dx = \pi.$$

Посмотрим, может ли **Mathematica** взять эти интегралы, ведь пределы интегрирования в них бесконечны. Оказывается, и это не представляет никакой проблемы:

```
In[59]:=Integrate[E^(-x^2),{x,-Infinity,Infinity}]
Out[59]=Sqrt[Pi]
In[60]:=Integrate[Sin[x]/x,{x,-Infinity,Infinity}]
Out[60]=Pi
```

Вот еще один очень красивый определенный интеграл, соответствующий неопределенный интеграл выражается в терминах полилогарифма, но при интегрировании от 1 до  $\infty$  все исчезает, остается (ПРОСТРАНСТВО, ЗВЕЗДЫ И ПЕВЕЦ) **константа Каталана C**:

```
In[61]:=Integrate[Log[x]/(1+x^2),{x,1,Infinity}]
Out[61]=Catalan
```

Заметим, что  $\int_0^1 \frac{\ln(x)}{1+x^2} dx = -C$ . Если хотите испытать культурный шок, то попробуйте теперь вычислить  $\int_0^{\infty} \frac{\ln(x)}{1+x^2} dx$ .

Попытка вычислить интеграл `Integrate[1/x,{x,-1,1}]` не приведет к большому успеху, так как система выдаст следующее сообщение об ошибке

`Integrate::idiv: Integral of 1/x does not converge on {-1,1}.`

В подобном случае можно попытаться вычислить главное значение интеграла. Вычисление

```
In[62]:=Integrate[1/x,{x,-1,1},PrincipalValue->True]
```

даст значение 0. Правило `PrincipalValue->True` меняет установку одной из **опций** функции `Integrate`. Это типичный пример того, как настройка опций меняет способ работы функций и влияет на получающийся ответ.

• **Кратные интегралы.** Команда `Integrate` служит и для вычисления кратных интегралов. Скажем, при вычислении двойного интеграла она вызывается в формате

`Integrate[f[x,y],{x,a,b},{y,c,d}]`.

Обратите внимание, что, как и в командах наподобие `Table` или `Array`, порядок итераторов здесь таков: внутренние итераторы изображаются последними. Таким образом, `Integrate[f[x,y],{x,a,b},{y,c,d}]` это

$$\int_a^b \int_c^d f(x,y) dy dx, \quad \text{а вовсе не} \quad \int_c^d \int_a^b f(x,y) dx dy.$$

Заметим, что при вычислении кратных интегралов результат в большинстве случаев следует упрощать, так как без этого система возвращает его `as is`, как сумму большого количества слагаемых, в том виде, как они возникают. Вот один из классических двойных интегралов,

```
In[63]:=Simplify[Integrate[1/Sqrt[x^2+y^2],{x,-1,1},{y,-1,1}]]
Out[63]=2*(2*ArcSinh[1]-Log[-1+Sqrt[2]]+Log[1+Sqrt[2]])
```

В этот момент у каждого, кто когда-либо преподавал `Multivariable Calculus`, да и просто у того, кто с самого начала внимательно читал этот параграф, закрадываются подозрения, что это все еще не окончательный ответ. Еще одна попытка заканчивается полной победой:

```
In[64]:=FullSimplify[Integrate[1/Sqrt[x^2+y^2],{x,-1,1},{y,-1,1}]]
Out[64]=8*ArcSinh[1]
```

Математик — это тот, кто никогда не останавливается на достигнутом, однако такого тройного интеграла, пожалуй, не считали даже наши студенты:

```
In[65]:=FullSimplify[Integrate[1/Sqrt[x^2+y^2+z^2],
                               {x,-1,1},{y,-1,1},{z,-1,1}]]
Out[65]=-2*(Pi-4*ArcCsch[Sqrt[2]]+Log[-8(-2+Sqrt[3])]-
          6*Log[1+Sqrt[3]])
```

Естественно, пределы интегрирования сами могут быть функциями от других аргументов:

```
In[66]:=Integrate[1,{x,-1,1},{y,-Sqrt[1-x^2],Sqrt[1-x^2]},
                  {z,-Sqrt[1-x^2-y^2],Sqrt[1-x^2-y^2]}]
Out[66]=4*Pi/3
```

Кстати, что это такое мы тут посчитали?

• **Численное интегрирование.** Мы уже знаем, что в элементарных функциях интеграл  $\int \sin(\sin(x)) dx$  не берется. Но, конечно, для любых конкретных  $a < b$  значение определенного интеграла  $\int_a^b \sin(\sin(x)) dx$  можно найти с любой точностью. Для этой цели проще всего пользоваться командой численного интегрирования `NIntegrate`, синтаксис которой не отличается от синтаксиса команды `Integrate` вызываемой для вычисления определенного интеграла. По умолчанию `NIntegrate` проводит вычисления с машинной точностью (чуть меньше 16 знаков после запятой). Вот пример ее использования:

```
In[67]:=NIntegrate[Sin[Sin[x]],{x,1,2}]
Out[67]=0.81645
```

Команда `NIntegrate` не может, конечно, дать такие значения, как  $\sqrt{\pi}$ . Обычно эта команда работает *чрезвычайно* быстро, так как по умолчанию она вначале компилирует программу вычисления подынтегральной функции и использует небольшую глубину рекурсии. Разумеется, это значит, что в случае часто осциллирующих функций (*highly oscillatory*

functions) применение команды `NIntegrate` — как и вообще *любое* применение приближенных вычислений!!! — может приводить к *очень* серьезным погрешностям. В частности, поэтому ПРИБЛИЖЕННЫЕ ВЫЧИСЛЕНИЯ *никогда* НЕ СЛЕДУЕТ ИСПОЛЬЗОВАТЬ ВНУТРИ РЕКУРРЕНТНЫХ ИЛИ ИТЕРАТИВНЫХ ПРОЦЕДУР, где уже за несколько десятков итераций ошибка может достичь десятков процентов. Разумеется, в подобных случаях система обычно предупреждает о возможных проблемах и для получения правильного ответа следует использовать символьные, а не численные вычисления, даже если соответствующие функции работают чуть дольше. Вот простейший пример, когда уже однократное применение функции `NIntegrate` без настройки опций приводит к чудовищной ошибке:

```
In[67]:=NIntegrate[Sin[1/x],{x,0,1}]
```

Это вычисление производится всего 0.01 секунды, но дает безнадежно неверный ответ 0.504894. Разумеется, система не только сообщает о возможных ошибках, но и дает советы по борьбе с ними:

```
NIntegrate::slwcon:
```

```
Numerical integration converging too slowly; suspect one of the
following: singularity, value of the integration being 0,
oscillatory integrand, or insufficient WorkingPrecision.
If your integrand is oscillatory try using the option
Method->Oscillatory in NIntegrate.
```

```
NIntegrate::ncvb:
```

```
NIntegrate failed to converge to prescribed accuracy after 7
recursive bisections in x near x=0.0035126778890767337'.
```

С нашей точки зрения ПРАВИЛЬНАЯ ПОЛИТИКА во всех подобных случаях СОСТОИТ В ПРОВЕДЕНИИ БЕЗОШИБОЧНЫХ ВЫЧИСЛЕНИЙ, с вычислением приближенного значения на самом последнем шаге:

```
In[68]:=N[Integrate[Sin[1/x],{x,0,1}]]
```

Это вычисление проводится уже 0.06 секунды, но зато дает правильный ответ 0.504067. Разумеется фанатики численных методов могут добиться такой же точности и другими способами, например, просто увеличив глубину рекурсии:

```
In[69]:=NIntegrate[Sin[1/x],{x,0,1},MaxRecursion->20]]
```

В этом случае система выдаст те же сообщения о возможных ошибках, что и выше, но зато вернет правильный ответ 0.504067. Правда теперь на нахождение этого ответа ей понадобится 6 секунд, в 100 раз больше, чем при точных вычислениях! В действительности, этот ответ получается уже при глубине рекурсии 17, но не проведя точного вычисления или оценки погрешности, мы не можем быть уверены в его правильности. К тому же даже проведение 17 подразделений занимает раз в 20 больше времени, чем получение точного ответа!



## § 10. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Man muß drei Dinge tun, um ein echter Mann zu werden: ein Haus bauen, einen Baum pflanzen, den Laplace-Operator in Kugelkoordinaten ausrechnen. — Чтобы стать настоящим мужчиной, нужно сделать три вещи: построить дом, посадить дерево и вычислить оператор Лапласа в сферических координатах.

Richard Courant

Основная команда для решения дифференциальных уравнений и систем дифференциальных уравнений `DSolve` также является одной из самых мощных, полезных и версатильных команд всей системы `Mathematica`. Реализация одной этой команды занимает 200 страниц кода на языке `C` и еще 300 страниц высокоуровневого кода на языке `Mathematica`, которые в полный рост обращаются к другим высокоуровневым командам (скажем, `Integrate`). В зависимости от деталей синтаксиса эта команда решает обыкновенные дифференциальные уравнения, системы обыкновенных дифференциальных уравнений, дифференциальные уравнения в частных производных и системы дифференциальных уравнений в частных производных!

• **Решение ODE.** Для решения одного дифференциального уравнения

$$f(x, y, y', y'', \dots, y^{(n)}) = 0,$$

где  $y$  — неизвестная функция от  $x$ , команда `DSolve` может использоваться в одном из двух следующих форматов

`DSolve[equation, y[x], x]` и `DSolve[equation, y, x]`.

Эти форматы отличаются лишь тем, что в первом случае ответ записывается в форме правила подстановки для значений  $y[x]$ , в то время как во втором случае ответом является *функция*  $y$ , в формате *чистой функции*.

Решение дифференциальных уравнений с постоянными коэффициентами является чисто алгебраической задачей (кольцо дифференциальных операторов с постоянными коэффициентами изоморфно кольцу многочленов!), с которой `Mathematica` справляется без малейшего труда:

```
In[70]:=DSolve[y''[x]+2*y'[x]-y[x]==1,y[x],x]
```

```
Out[70]={{y[x]->-1+E^((-1-Sqrt[2])*x)*C[1]+E^((-1+Sqrt[2])*x)*C[2]}}
```

Вот тот же пример с чуть более сложной правой частью. Заметим, что в подобных случаях следует применять команды `Simplify` или `FullSimplify`, для того, чтобы привести решение к более компактному виду:

```
In[71]:=FullSimplify[DSolve[y''[x]+2*y'[x]-y[x]==Exp[a*x],y[x],x]]
```

```
Out[71]={{y[x]->E^(-(1+Sqrt[2])*x)*(E^((1+Sqrt[2]+a)*x)+
(-1+a(2+a))*(C[1]+E^(2*Sqrt[2]*x)*C[2]))/(-1+a*(2+a))}}
```

В некоторых случаях довольно легко решаются и уравнения вида

$$f(y, y', y'', \dots, y^{(n)}) = 0,$$

в которых функция  $f$  явно не зависит от  $x$ . Вот один из типичных примеров

```
In[72]:=DSolve[y'[x]+y[x]^3==0,y[x],x]
Out[72]={{y[x]->-1/Sqrt[2*x-2*C[1]]},{y[x]->1/Sqrt[2*x-2*C[1]]}}
```

Но, конечно, в случае непостоянных коэффициентов явное решение уже совсем простых дифференциальных уравнений часто становится чрезвычайно хитрым делом. Вот простенький пример, встретившийся в нашей собственной практике, в связи с задачей из области дифференциальной алгебры. Получение следующего решения уравнения

$$y''(x) = xy(x) + 1$$

заняло 0.03 секунды:

```
In[73]:=DSolve[y''[x]==x*y[x]+1,y[x],x]
Out[73]={{y[x]->AiryAi[x]*C[1]+AiryBi[x]*C[2]+
(-3*3^(5/6)*Pi*x*AiryAi[x]*Gamma[1/3]*Gamma[5/3]*
HypergeometricPFQ[{1/3},{2/3,4/3},x^3/9]+
3*3^(1/3)*Pi*x*AiryBi[x]*Gamma[1/3]*Gamma[5/3]*
HypergeometricPFQ[{1/3},{2/3,4/3},x^3/9]-
3*3^(1/6)*Pi*x^2*AiryAi[x]*Gamma[2/3]^2*
HypergeometricPFQ[{2/3},{4/3,5/3},x^3/9]-
3^(2/3)*Pi*x^2*AiryBi[x]*Gamma[2/3]^2*
HypergeometricPFQ[{2/3},{4/3,5/3},x^3/9])/
(27*Gamma[2/3]*Gamma[4/3]*Gamma[5/3])}}
```

Мы, конечно, знали, что **функции Эйри**  $Ai(x)$  и  $Bi(x)$  — на языке Mathematica `AiriAi` и `AiriBi` — образуют фундаментальную систему решений дифференциального уравнения  $y''(x) = xy(x)$ . Однако явная формула, выражающая частное решение неоднородного уравнения как линейную комбинацию произведений функций Эйри на обобщенные гипергеометрические функции, да еще с такими хитрыми коэффициентами, показалась нам достаточно удивительной. Поэтому мы решили *определить*  $y[x]$  этой формулой и вычислить

```
In[74]:=FullSimplify[D[y[x],x,x]-x*y[x]-1]
```

После примерно *четырёх минут* напряженных раздумий система подтвердила, что это действительно 0. Этот пример совершенно поразителен: обычно найти решение в тысячи раз сложнее, чем проверить, что найденное решение действительно является решением, но в данном случае дела обстоят прямо противоположным образом!!!

• **Решение систем ODE.** Команда `DSolve` служит и для решения систем дифференциальных уравнений. Например, для того, чтобы решить систему двух дифференциальных уравнений относительно двух неизвестных функций  $x = x(t)$  и  $y = y(t)$  нужно вызывать эту команду либо в формате

```
DSolve[{equation1,equation2},{x[t],y[t]},t]
```

либо в формате

```
DSolve[{equation1,equation2},{x,y},t]
```

В обоих случаях как уравнения, так и неизвестные функции должны оформляться в виде списков. Как и в случае одного уравнения, единственное различие этих записей состоит в том, что в первом случае ответ будет представляться в виде списка правил преобразования для функций  $x[t]$  и  $y[t]$ , в то время как во втором случае неизвестные функции будут выражены в формате чистых функций. Это различие может иметь значение только если Вы непосредственно вызываете результат вычисления `DSolve` в другой команде.

Следующий незатейливый на первый взгляд пример довольно широко обсуждался в литературе по компьютерной алгебре в связи с тем, что вплоть до середины 1990-х годов *Mathematica 2.2* ко всеобщему удивлению оставалась *единственной* системой общего назначения, способной решить следующую систему дифференциальных уравнений<sup>65</sup>. Разумеется, было бы довольно странно, если бы *Mathematica 5.1* вдруг разучилась решать то, что умела делать уже *Mathematica 2.2*:

$$\frac{dx}{dt} = x \left( 1 + \frac{\cos(t)}{2 + \sin(t)} \right), \quad \frac{dy}{dt} = x - y.$$

Как мы только что сказали, решение этой системы должно оформляться следующим образом:

```
In[75]:=Solve[{D[x[t],t]==x[t]*(1+Cos[t]/(2+Sin[t])),
               D[y[t],t]==x[t]-y[t]},{x[t],y[t]},t]
Out[75]={ {x[t]->E^t*C[1]*(2+Sin[t]),
           y[t]->E^(-t)*C[2]-1/5*E^t*C[1]*(-5+Cos[t]-2*Sin[t])} }
```

• **Начальные условия.** С точки зрения *Mathematica* решение задачи с начальными условиями ничем не отличается от решения системы уравнений. Таким образом, для нахождения решения  $y$  дифференциального уравнения `equation` с начальным условием  $y(a) = c$ , функция `DSolve` вызывается в одном из следующих форматов:

```
o DSolve[{equation,y[a]==c},y[x],x]
o DSolve[equation&&y[a]==c,y[x],x]
```

либо, наконец, в формате, когда левые и правые части уравнений записаны списками.

Проиллюстрируем это на простом примере. **Функции Бесселя**  $J_n$  и  $Y_n$  являются линейно независимыми решениями дифференциального **уравнения Бесселя**

$$x^2 y''(x) + x y'(x) + (x^2 - n^2) y(x) = 0.$$

<sup>65</sup>M.Wester, A review of CAS mathematical capabilities. — различные версии этого те(к)ста многократно публиковались в нескольких журналах по *Computer Science* и прикладной математике в 1990-е годы, последнюю версию можно найти на домашней странице автора на <http://www.math.unm.edu>

Они отличаются тем, что  $J_n$  принимает конечное значение в точке  $x = 0$ , в то время как  $Y_n$  логарифмически уходит в  $-\infty$  при  $x \rightarrow 0+$ . Традиционно функция  $J_n$  называется функцией Бесселя первого рода, а  $Y_n$  — функцией Бесселя второго рода. Функции Бесселя естественно возникают во всех задачах с цилиндрической симметрией.

Вот пример дифференциального уравнения, известного как **уравнение Бернулли**, сводящегося к уравнению Бесселя заменой переменной. Естественно, его решения выражаются через функции Бесселя<sup>66</sup>:

```
In[76]:=DSolve[x*D[y[x],x]+D[y[x],x]+y[x]==0,y[x],x]
Out[76]={{y[x]->BesselJ[0,2*Sqrt[x]]*C[1]+
          2*BesselY[0,2*Sqrt[x]]*C[2]}}
```

Допустим, нас интересует решение, проходящее через точку  $(0, 1)$ , его можно искать, например, при помощи следующей команды. Естественно,  $Y_n$  исчезнет, останется только подходящее кратное  $J_n$ :

```
In[77]:=DSolve[x*y'[x]+y[x]==0&&y[0]==1,y[x],x]
Out[77]={{y[x]->BesselJ[0,2*Sqrt[x]]}}
```

• **Решение PDE.** Формально команда `DSolve` может решать дифференциальные уравнения в частных производных. Например, для отыскания функции  $u(x, y)$  двух переменных  $x, y$ , удовлетворяющей дифференциальному уравнению `equation`, функция `DSolve` вызывается в одном из следующих обычных форматов:

- `DSolve[equation,u[x,y],{x,y}]`
- `DSolve[equation,u,{x,y}]`

в зависимости от того, в каком виде мы хотим увидеть ответ.

Однако фактически за исключением некоторых совсем тривиальных случаев — таких, как уравнения первого порядка и несколько сводящихся к ним уравнений второго порядка — команда `DSolve` решает уравнения в частных производных примерно так же, как команда `Solve` решает трансцендентные уравнения, т.е. *очень* плохо.

Каждый младенец, который прочел первую главу Куранта и Гильберта, (даже если он не умеет еще вычислять оператор Лапласа в цилиндрических координатах) знает, что интегрирование уравнения первого порядка сводится к решению системы обыкновенных дифференциальных уравнений. Рассмотрим тривиальный пример

$$y \frac{\partial u(x, y)}{\partial x} + x \frac{\partial u(x, y)}{\partial y} = 1,$$

который за секунду решается от руки: частное решение  $\ln(x + y)$ , общее решение однородного уравнения  $f(x^2 - y^2)$  для любой дифференцируемой

<sup>66</sup>Исторически, конечно, последовательность событий была обратной. Даниил Бернулли написал свое уравнение и ввел функции Бесселя лет за сто до самого Бесселя. В такой форме, как здесь, уравнение Бернулли записано Эйлером в 1781 году.

функции  $f$ . Однако даже для этого простейшего случая все совсем не так безоблачно, как бы того хотелось:

```
In[78]:=Refine[DSolve[y*D[u[x,y],x]+x*D[u[x,y],y]==1,
                    u[x,y],{x,y}],Element[{x,y},Reals]]
Out[78]={{u[x,y]->-Log[x+Abs[y]]+C[1][1/2*(-x^2+y^2)]}
          {u[x,y]->Log[x+Abs[y]]+C[1][1/2*(-x^2+y^2)]}}
```

Принципиальным новым обстоятельством здесь является то, что  $C[1]$  представляет собой не константу, как раньше, а произвольную дифференцируемую функцию! Это видно, например, из использования квадратных скобок:  $(x^2 - y^2)/2$  рассматривается как аргумент этой функции.

Еще один забавный момент в работе команды `DSolve` состоит в том, что она автоматически не устраняет отсутствие симметрии между  $x$  и  $y$ , проистекающее из хода решения:

```
In[79]:=Refine[DSolve[y*D[u[x,y],x]+x*D[u[x,y],y]==1,
                    u[x,y],{y,x}],Element[{x,y},Reals]]
Out[79]={{u[x,y]->-Log[y+Abs[x]]+C[1][1/2*(x^2-y^2)]}
          {u[x,y]->Log[y+Abs[x]]+C[1][1/2*(x^2-y^2)]}}
```

Все в недоумении: что же за логарифм все-таки возникает в качестве специального решения,  $\ln(x + |y|)$  или  $\ln(y + |x|)$ ??

• **Волновое уравнение.** А вот простейшее гиперболическое уравнение второго порядка, **одномерное волновое уравнение** или **уравнение струны**:

$$\frac{\partial^2 u(x, t)}{\partial x^2} = c^2 \frac{\partial^2 u(x, t)}{\partial t^2}.$$

В данном случае классически известное общее решение — **решение Даламбера** — представляет собой суперпозицию двух плоских волн, распространяющихся не меняя формы вдоль оси  $x$  со скоростью  $c$  в противоположных направлениях:

$$u(x, t) = f(x + ct) + g(x - ct),$$

и действительно `Mathematica` сразу находит это общее решение:

```
In[80]:=Refine[DSolve[D[u[x,t],x,x]==c^2*D[u[x,t],t,t],
                    u[x,t],{x,t}],c>0]
Out[80]={{u[x,t]->C[1][t-c*x]+C[2][t+c*x]}}
```

Из формы ответа видно, что  $C[1]$  и  $C[2]$  являются двумя произвольными функциями. Будем для краткости считать, что  $c = 1$ .

До сих пор все замечательно, однако не следует расслабляться: оказывается, фактически решить в символьном виде задачу с начальными и/или краевыми условиями совсем непросто! Допустим, нас интересуют периодические по  $x$  решения с заданными координатами и скоростями в момент  $t = 0$ .

Для начала, встроенная помощь становится удивительно немногословной именно в то мгновение, когда дело доходит до ключевого аспекта: в каком формате следует задавать граничные условия? При попытке задать нетривиальные граничные условия как дополнительные уравнения *Mathematica* начинает жаловаться либо на переопределенность системы:

```
DSolve::overdet: The system has fewer dependent variables than
equations, so is overdetermined,
```

либо на то, что с ее точки зрения эти дополнительные условия вообще не являются дифференциальными уравнениями:

```
DSolve::deqx: Supplied equations are not differential equations
of the given functions.
```

Можно, конечно, ввести специальные обозначения для произвольных функций, входящих в общее решение, и попытаться решать систему относительно них, но тут нас поджидает новая засада:

```
DSolve::litarg: To avoid possible ambiguity, the arguments of
the dependent variable should literally
match the independent variables.
```

Вероятно, с четвертой или пятой попытки, руководствуясь аналогиями и общим пониманием ситуации (потому что больше руководствоваться совершенно нечем — ВЕДЬ КРОМЕ ТОГО, ЧТО МЫ ЗНАЕМ, МЫ НЕ ЗНАЕМ АБСОЛЮТНО НИЧЕГО) нам удастся написать что-нибудь в духе

```
In[81]:=DSolve[D[u[t,x],x,x]==D[u[t,x],t,t]&&u[0,x]==f[x]&&
Derivative[1,0][u][0,x]==g[x]&&u[t,0]==u[t,Pi],
u[t,x],{t,x}]
```

что не вызовет *немедленного* протеста системы. Однако в этот момент вскрыется самое неприятное обстоятельство: для большинства конкретных функций  $f$  и  $g$  система просто не в состоянии найти решение задачи с граничными условиями в символьном виде!!!

• **Приближенное решение дифференциальных уравнений.** Все же в оправдание *Mathematica* следует заметить, что, по-видимому, точное решение дифференциальных уравнений в частных производных представляет собой совсем непростую задачу. Большинство математических пакетов, такие как *MatLab*, даже не пытаются делать ничего подобного, и всегда решают дифференциальные уравнения в частных производных только в численном виде. В *Mathematica* это делается при помощи команды **численного решения дифференциальных уравнений** *NDSolve*. Для интересующего нас случая функции от двух переменных  $x$  и  $t$  эта команда вызывается в следующем формате

```
NDSolve[equations,u,{x,a,b},{t,c,d}]
```

где *equations* представляют собой дифференциальные уравнения и граничные условия на функцию  $u$ , которая интерполируется при  $a \leq x \leq b$  и  $c \leq t \leq d$ . Стоит отметить, что команда *NDSolve* представляет собой одну из самых мощных и богатых опциями команд всей системы, в которой

использованы по крайней мере полтора десятка различных приближенных методов, с возможностью переключения между ними в процессе вычисления. Достаточно сказать, что реализация этой команды занимает около 1400 страниц кода, даже больше, чем реализация команды `Integrate!!!`

Команда `NDSolve` возвращает **интерполирующую функцию** — численное приближение к решению. Однако единственный осмысленный способ вывести эту функцию, конечно, состоит в том, чтобы *посмотреть* на нее при помощи команд трехмерной графики. Поэтому мы отложим детальное обсуждение того, как используется команда `NDSolve`, до выпуска 4.

Кроме того, пакет `Calculus`DSolveIntegrals`` содержит чрезвычайно полезную функцию `CompleteIntegral` в терминах которой в некоторых случаях удастся выразить решения задач с граничными условиями.

## § 11. ЗАПИСЬ ВЕКТОРОВ И МАТРИЦ

Как представитель комбинаторики, я идентифицирую реальность с матрицами из 0 и 1.

Дональд Кнут. Математическая типография

Сейчас мы совсем коротко обсудим основы представления, генерации и изображения векторов и матриц в системе *Mathematica*. Мы ограничиваемся лишь *абсолютным минимумом*, необходимым для решения простейших задач линейной алгебры. Более детально все эти вопросы обсуждаются в Выпуске 3.

• **Списки.** Одной из объединяющих концепций, вокруг которых организован язык *Mathematica*, является понятие **списка** (`List`). С точки зрения языка системы и внутреннего представления данных все на свете — множества, наборы, последовательности, векторы, матрицы, тензоры, и т.д. — трактуется как список. Более того, грамотное программирование на языке *Mathematica* состоит в том, чтобы избегать явного использования циклов, а вместо этого объединять переменные, команды, уравнения, условия, подстановки, ... в списки.

В *Mathematica* список  $(x_1, \dots, x_n)$  с компонентами  $x_1, \dots, x_n$  обозначается  $\{x_1, \dots, x_n\}$  или, в полной форме, `List[x1, ..., xn]`. Подчеркнем, что с математической точки зрения список длины  $n$  представляет собой **упорядоченную  $n$ -ку** ( *$n$ -tuple*) или, если мы не хотим явно упоминать ее длину<sup>67</sup>, **тупель**. По определению два списка равны, если равны их длины и все их соответствующие компоненты:

$$(x_1, \dots, x_m) = (y_1, \dots, y_n) \iff m = n \text{ и } x_i = y_i, i = 1, \dots, n$$

В программировании существует устойчивая традиция называть компоненты списка **элементами** этого списка. Как мы только что отметили, при определении равенства списков учитываются кратности и порядок их элементов!

<sup>67</sup>Например, при  $n = k$ .

**Комментарий.** В русской учебной литературе нет единства по поводу того, как следует переводить термин `Tuple`, `tuple`. Под влиянием программирования сегодня `tuple` и в математических работах чаще всего переводится как **список**. Некоторые авторы предлагали перевод **кортеж**, но работающие математики никогда им не пользуются. Поэтому мы пропагандируем термин **тупель** и его производные **дупель**, **трипель**, **квадрупель**, **квинтупель**, **секступель**, **септупель** и **октупель** для обозначения упорядоченных пар, троек, четверок, пятерок, шестерок, семерок и восьмерок.

Таким образом, список  $\{a, b, a, c\}$  или, в полной форме, `List[a, b, a, c]` представляет именно тупель  $(a, b, a, c)$ , а вовсе не набор или множество с элементами  $a, b, a, c$ . Это значит, например, что в ответ на вопрос

$$\text{TrueQ}[\{a, b, c\} == \{b, a, c\}]$$

система возвратит **False** — если, конечно, переменным  $a, b$  не были ранее присвоены одинаковые значения!

Применение к списку команды `Sort` расставляет его элементы в некотором *естественном* порядке, связанном с их записью как **выражений** в языке `Mathematica`. Таким образом, эффективно применение `Sort` заставляет систему *игнорировать* порядок элементов списка, делая два списка равными, в том и только том случае, когда совпадают представленные ими **наборы**  $[x_1, \dots, x_n]$ . Напомним, что при определении равенства наборов учитываются *кратности* входящих в них элементов, но не их порядок:

```
In[82]:=TrueQ[Sort[{a,b,a,c}]==Sort[{b,c,a,a}]]
```

```
Out[82]=True
```

```
In[83]:=TrueQ[Sort[{a,b,a,c}]==Sort[{b,c,b,a}]]
```

```
Out[83]=False
```

С другой стороны, команда `Union` превращает список в **множество**, т.е. заставляет систему игнорировать не только порядок элементов списка, но и их кратности:

```
In[84]:=TrueQ[Union[{a,b,a,c}]==Union[{b,c,b,a}]]
```

```
Out[84]=False
```

В системе содержится несколько десятков операций над списками, относящихся к следующим категориям:

- теоретико-множественные операции;
- манипуляции с частями списка: извлечения, вычеркивания, вставки, замены, выборки, и т.д.
- структурные манипуляции: сортировки, перестановки, выравнивания, разбиения, и т.д.
- применение функций к спискам и их частям: `Map`, `Apply`, `Thread`, `Inner`, `Outer` и их многочисленные варианты.

Гибкое применение возможностей этих операций составляет йогу системы `Mathematica`, однако овладение этим искусством требует некоторого ментального тюнинга и многомесячной практики. Выпуск 3 как раз и посвящен основам функционального и списочного программирования, поэтому здесь мы не будем обсуждать более тонкие аспекты работы со списками, а



ограничимся иллюстрацией применения нескольких простейших встроенных функций к решению стандартных задач линейной алгебры.

• **Векторы.** В простейшем варианте вектор  $(x_1, \dots, x_n)$  с координатами  $x_1, \dots, x_n$  представляется списком  $\{x_1, \dots, x_n\}$ . При этом координаты вектора могут быть любыми выражениями: числами, символами, списками, матрицами и т.д.

При действиях над векторами все обычные операции трактуются как **покомпонентные**:

In[85]:  $\{u, v, w\} + \{x, y, z\}$

Out[85]:  $\{u+x, v+y, w+z\}$

In[86]:  $\{u, v, w\} * \{x, y, z\}$

Out[86]:  $\{u*x, v*y, w*z\}$

Более того, арифметические операции имеют атрибут `Listable`. Это значит, что при выполнении этих операций скаляр  $x$  отождествляется с вектором  $(x, \dots, x)$  подходящей длины. Иными словами,

In[87]:  $\{w + \{x, y, z\}, w * \{x, y, z\}\}$

Out[87]:  $\{\{w+x, w+y, w+z\}, \{w*x, w*y, w*z\}\}$

Стоит подчеркнуть, что в такой форме записи не делается различия между строками и столбцами. Таким образом, вычисление  $\{a, b, c\} \cdot \{x, y, z\}$  или, что то же самое, `Dot[{a, b, c}, {x, y, z}]`, даст  $a*x + b*y + c*z$ . Тот же результат получится и при вычислении `Inner[Times, {a, b, c}, {x, y, z}]`. Однако интерпретировать этот результат можно совершенно по-разному, в зависимости от того, как мы представляем себе исходные векторы и чем являются их элементы.

- произведение строки  $(a, b, c)$  на столбец  $(x, y, z)^t$ ;
- скалярное произведение строки  $(a, b, c)$  на строку  $(x, y, z)$ ;
- скалярное произведение столбца  $(a, b, c)^t$  на столбец  $(x, y, z)^t$ ;
- линейную комбинацию строк или столбцов  $x, y, z$  с коэффициентами  $a, b, c$ ;

и многими другими способами.

**Комментарий.** В действительности,  $\{x, y, z\}$  не является ни строкой, ни столбцом. Интересно, что Вольфрам считает это достоинством: Because of the way *Mathematica* uses lists to represent vectors and matrices, you never have to distinguish between “row” and “column” vectors. Как мы сейчас увидим, вектор  $(x, y, z)$ , рассматриваемый как строка, должен записываться в виде  $\{\{x, y, z\}\}$ , а рассматриваемый как столбец — в виде  $\{\{x\}, \{y\}, \{z\}\}$ . С нашей точки зрения это нарушение симметрии между строками и столбцами является одним из самых серьезных концептуальных дефектов системы. Конечно, оно никак не сказывается на практических вычислениях. С другой стороны, в данном случае *Mathematica* всего лишь следует обычной при элементарном подходе практике, когда не делается никакого различия между правыми и левыми векторными пространствами. В действительности, даже над полем необходимо тщательно различать **правые векторные пространства**, координаты векторов в которых записываются *столбцами*, а базисы, соответственно, строчками (состоящими из столбцов) и **левые**

**векторные пространства**, координаты векторов в которых записываются *строчками*, а базисы, соответственно, столбцами (состоящими из строчек). Стоит только начать различать столбцы и строки и все в линейной алгебре сразу становится на свои места. Например, из четырех матриц  $g$ ,  $g^t$ ,  $g^{-1}$  и  $g^{-t}$ , отвечающих за преобразование координат различных геометрических объектов, остаются всего две: ковариантная  $g$  и контравариантная  $g^{-1}$ .

• **Запись матриц.** Матрица в языке Mathematica записывается как *список*, составленный из *строк* этой матрицы. Список, элементы которого сами являются списками, называется **вложенным списком** (*nested list*). Таким образом, матрица

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

изображается как  $\{\{a,b\},\{c,d\}\}$ . Именно такая форма используется при *вводе* матриц, внутри себя система переводит это выражение в полную форму `List[List[a,b],List[c,d]]`.

Симметрия между строками и столбцами восстанавливается при помощи команды `Transpose`, переводящей матрицу в транспонированную матрицу, строки которой совпадают со столбцами исходной:

```
In[88]:=Transpose[{{a,b},{c,d}}]
```

```
Out[88]={{a,c},{b,d}}
```

По умолчанию команда `Transpose` переставляет *два верхних уровня* вложенности списков. Таким образом, например, команда `Transpose`, примененная к *списку* матриц будет переставлять строки этих матриц между собой, а вовсе не транспонировать сами эти матрицы. Для одновременного транспонирования списка матриц нужно применять `Transpose` к *элементам* этого списка при помощи команды `Map`:

```
In[89]:=Transpose[{{a,b},{c,d}},{{e,f},{g,h}}]
```

```
Out[89]={{a,b},{e,f}},{{c,d},{g,h}}
```

```
In[90]:=Map[Transpose,{{a,b},{c,d}},{{e,f},{g,h}}]
```

```
Out[90]={{a,c},{b,d}},{{e,g},{f,h}}
```

Из описанного представления матриц и того, что было сказано выше об операциях над векторами следует, что при действиях над матрицами все обычные операции трактуются как **покомпонентные**:

```
In[91]:={{a,b},{c,d}}+{{e,f},{g,h}}
```

```
Out[91]={{a+e,b+f},{c+g,d+h}}
```

```
In[92]:={{a,b},{c,d}}*{{e,f},{g,h}}
```

```
Out[92]={{a*e,b*f},{c*g,d*h}}
```

Таким образом, `*` (`Times`) это умножение матриц по Адамару, а вовсе не обычное произведение матриц, которое обозначается `.` (`Dot`).

Более спорным представляется применение того же правила к скалярам, которые при этом отождествляются не с кратными единичной, а с кратными **пробной матрицы**, состоящей из одних единиц! Изобразим для

примера пробную матрицу степени 3:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

В свете этого соглашения нас не должен удивлять следующий результат:

```
In[93]:={{ {a,b},{c,d}}+x,{ {a,b},{c,d}}*x}
```

```
Out[93]:={{ {a+x,b+x},{c+x,d+x}},{ {a*x,b*x},{c*x,d*x}}}
```

• **Генерация векторов и матриц.** Основной командой генерации векторов, матриц и любых других списков в языке *Mathematica* является команда *Table*. Вызванная с одним итератором команда

```
Table[f[i],{i,m,n}]
```

порождает список значений функции  $f$  в точках  $i = m, m + 1, \dots, n$ . Вот типичный пример использования этой команды:

```
In[94]:=Table[x^i/i!,{i,0,5}]
```

```
Out[94]={1,x,x^2/2,x^3/6,x^4/24,x^5/120}
```

А вот забавная вариация на тему этого примера, детально обсуждаемая Анри Пуанкаре в “Новых методах небесной механики”:

```
In[95]:=Table[N[10^i/i!],{i,1,30}]
```

```
Out[95]={10., 50., 166.667, 416.667, 833.333, 1388.89, 1984.13,
2480.16, 2755.73, 2755.73, 2505.21, 2087.68, 1605.9,
1147.07, 764.716, 477.948, 281.146, 156.192, 82.2064,
41.1032, 19.5729, 8.89679, 3.86817, 1.61174, 0.644695,
0.24796, 0.0918369, 0.0327989, 0.01131, 0.00376999}
```

Из этой таблицы видно, что отношение  $10^i/i!$  вначале очень быстро растет (“расходится в смысле астрономов”), а потом очень быстро убывает (“сходится в смысле математиков”).

Вызванная с двумя итераторами команда

```
Table[f[i,j],{i,k,l},{j,m,n}]
```

порождает *вложенный* список значений функции двух аргументов  $f$  в парах  $(i,j)$ , где  $i = k, k + 1, \dots, l$ ,  $j = m, m + 1, \dots, n$ . Этот список будет организован как матрица, причем итератор  $i$  считается **внешним**, а  $j$  — **внутренним**, иными словами,  $i$  нумерует строки, а  $j$  — позиции внутри строк. Таким образом, строки этой матрицы имеют вид  $(f_{im}, \dots, f_{in})$ . Еще раз обратите внимание на следующие два ключевых момента в этом определении:

- МАТРИЦА ТРАКТУЕТСЯ КАК СТРОКА, СОСТАВЛЕННАЯ ИЗ СТРОК,
- ВНУТРЕННИЕ ИТЕРАТОРЫ ПИШУТСЯ ПОСЛЕДНИМИ.

Иными словами, если мы дадим два определения матрицы *Forward*

```
In[96]:=forwaa[n_]:=Table[If[j==i+1,1,0],{i,1,n},{j,1,n}]
```

```
In[97]:=forwbb[n_]:=Table[If[j==i+1,1,0],{j,1,n},{i,1,n}]
```

отличающиеся лишь порядком итераторов, то получившиеся матрицы будут транспонированы друг к другу. Вот как, например, выглядят матрицы `forwaa[4]` и `forwbb[4]` в традиционной математической нотации

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Иными словами, только одна из этих матриц в действительности окажется матрицей **Forward**, вторая будет матрицей **Backward**!

Ну а, скажем, упомянутая в предыдущем пункте пробная матрица задается как

```
test[n_]:=Table[1,{i,1,n},{j,1,n}]
```

В системе **Mathematica** существует большое количество других команд генерации списков и матриц специального вида. Одной из таких команд является `DiagonalMatrix[{d1,...,dn}]`, которая порождает диагональную матрицу с диагональными элементами  $d_1, \dots, d_n$ . Однако в тех простых ситуациях, которые мы здесь рассматриваем, вполне достаточно команды `Table`. Кроме того, как мы обсуждаем ниже, матрицы можно создавать из векторов или, наоборот, более глубоких списков при помощи структурных команд таких, как `Partition` или `Flatten`.

• **Части матриц.** Для выделения частей списков в языке **Mathematica** используется специальный вид скобок — двойные квадратные скобки `[[]]`, являющиеся сокращением команды `Part`. При этом в соответствии с общими правилами спецификации уровня, детально обсуждаемыми в выпуске 2, `x[[i]]` — или, что то же самое, `Part[x,i]` — обозначает  $i$ -ю часть списка  $x$ ; `x[[-i]]` —  $i$ -ю часть с конца; `x[[i,j]]` —  $j$ -ю часть его  $i$ -части и т.д. При этом сами номера позиций тоже могут задаваться списком, а если мы хотим выбрать *все* части на каком-то уровне, то спецификацией `All`. Приведем несколько примеров применения этих соглашений

- `x[[i,j]]` — элемент  $x$  в позиции  $(i,j)$ ;
- `x,[[{i,j},{k,l}]]` — подматрица порядка 2 матрицы  $x$ , стоящая на пересечении строк с номерами  $i,j$  и столбцов с номерами  $k,l$ ;
- `x[[i]]` —  $i$ -я строка матрицы  $x$ ;
- `x[[All,j]]` —  $j$ -й столбец матрицы  $x$ ;
- `Tr[x,List]` — главная диагональ матрицы  $x$ ;

Так, например, `x[[1]]` и `x[[-1]]` обозначают, соответственно, первую и последнюю строку матрицы  $x$ , а `x[[All,1]]` и `x[[All,-1]]` — ее первый и последний столбец. Еще раз обратите внимание на отсутствие здесь симметрии между строками и столбцами! Столбец матрицы является строкой транспонированной матрицы, поэтому  $j$ -й столбец матрицы  $x$  можно породить также посредством `Transpose[x][[j]]`.

Разумеется, также и главную диагональ матрицы можно определять тысячей других способов, например, напрашивающимся

```
Table[x[[i,i]],{i,1,Length[x]}].
```

Однако встроенная команда `Tr[x,List]` не только изящнее, но и работает быстрее, хотя разница во времени становится по настоящему заметной только для матриц порядка несколько десятков тысяч.

• **Просмотр матриц.** Чтобы увидеть матрицу не как вложенный список, а в традиционной записи, надо применить к ней одну из команд `TableForm` или `MatrixForm`. Скажем, следующая строка определяет одну из самых важных в самой математике и ее приложениях матриц — **матрицу Вандермонда**  $V(x_1, \dots, x_n)$ . Собственно говоря, вся теория определителей построена исключительно для анализа этого примера:

```
In[98]:=vandermonde[x_,n_]:=Table[x[j]^i,{i,0,n-1},{j,1,n}]
```

Посмотрим теперь, как выглядит матрица `vandermonde[y,5]`. Это делается при помощи команды `MatrixForm`. Вычислив

```
In[99]:=MatrixForm[vandermonde[y,5]]
```

мы увидим *примерно* следующий результат:

$$V(y_1, y_2, y_3, y_4, y_5) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ y_1 & y_2 & y_3 & y_4 & y_5 \\ y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 \\ y_1^3 & y_2^3 & y_3^3 & y_4^3 & y_5^3 \\ y_1^4 & y_2^4 & y_3^4 & y_4^4 & y_5^4 \end{pmatrix}$$

Разумеется, здесь мы воспроизводим не то, что будет фактически отображено в записной книжке *Mathematica*, а уже облагороженный  $\text{\TeX}$ 'овский аутпут типографского качества, получающийся с помощью применения по-верх `MatrixForm` команды экспорта `TeXForm`.

Мы сами обычно не печатаем форматные команды `TableForm`, `MatrixForm` и т.д. заранее, а применяем их к уже имеющемуся выражению **в постфиксной форме** (задним числом). Это делается при помощи оператора `//`, примерно так

```
In[100]:=vandermonde[y,5] // MatrixForm
```

В первом приближении различие между командами `TableForm` и `MatrixForm` такое же, как между `matrix` и `pmatrix` в  $\text{\TeX}$ 'е. Иными словами, `TableForm[x]` выводит элементы матрицы  $x$  в виде таблицы, а `MatrixForm[x]` — в традиционной математической записи с использованием круглых скобок.

**Комментарий.** В действительности, кроме этого очевидного отличия, между командами `TableForm` и `MatrixForm` имеется большое количество тонких различий в том, как они по умолчанию трактуют горизонтальное (слева, сверху, по центру, по десятичной точке, по фиксированному символу и т.д.) и вертикальное (по верху, по низу, по центру, по базовой линии, по оси) выравнивание строк и столбцов и индивидуальных элементов внутри строки или столбца. В Выпуске 3 мы детально обсуждаем использование

форматных опций, а в Выпуске 4 возможность экспорта результатов сессии **Mathematica** для их включения в текстовый файл. Однако для большинства пользователей, которые интересуются *результатом* вычисления, а не получением типографского продукта профессионального или полупрофессионального качества, подобные тонкости не имеют большого значения.

• **Матричные единицы.** Самыми важными с точки зрения профессионального математика матрицами являются **стандартные матричные единицы**  $e_{ij}$ , у которых в позиции  $(i, j)$  стоит 1 и 0 во всех остальных местах. Матрицы  $e_{ij}$ ,  $1 \leq i, j \leq n$ , образуют базис  $M(n, K)$  как векторного пространства над  $K$ . Более того, это **мультипликативный базис**, в том смысле, что произведение двух базисных элементов либо равно 0, либо снова представляет собой базисный элемент:  $e_{ij}e_{hk} = \delta_{jh}e_{ik}$ . Таким образом,  $e_{ij}e_{hk} = 0$ , если  $j \neq h$ , в то время как  $e_{ij}e_{jk} = e_{ik}$ . Вот одно из возможных определений этих матриц в языке **Mathematica**. Конечно, теперь мы должны явно указывать не только  $i, j$ , но и порядок  $n$ :

```
In[101]:=e[n_,i_,j_]:=Table[If[h==i,1,0]*If[k==j,1,0],
                               {h,1,n},{k,1,n}]
```

Еще раз обратите внимание на несколько уже встречавшихся нам принципиальных моментов:

- использование `_ Blank` и `:= SetDelayed` для определения функции;
- использование `== Equal` в уравнениях `i==h` и `j==k`;
- использование условного оператора `If[condition,x,y]`, возвращающего  $x$ , если `condition==True` и  $y$ , если `condition==False`.

**Комментарий.** Как мы обсуждаем в дальнейшем, в общем случае намного разумнее вызывать оператор `If` с четырьмя аргументами, в формате `If[condition,x,y,z]`, явным образом предписывая, что следует делать в случае, когда система не может решить, выполняется условие `condition` или нет. Однако здесь мы используем этот оператор в чисто иллюстративных целях и только в простейшей ситуации сравнения небольших целых чисел, когда у системы не должно быть никаких сомнений, равны они или нет. В действительности вместо `If[i==h,1,0]` мы могли бы воспользоваться встроенной функцией `KroneckerDelta`, но это чуть длиннее.

Посмотрим теперь, как выглядят стандартные матричные единицы степени 3. Применив к ним форматную команду `MatrixForm`

```
In[102]:=Map[MatrixForm,Flatten[Table[e[3,i,j],{i,1,3},{j,1,3}],1]]
```

мы увидим следующее:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Стоит подчеркнуть здесь один момент, не встречавшийся до сих пор. Дело в том, что команда `Table[e[3,i,j],{i,1,3},{j,1,3}]` создает вовсе не список из девяти стандартных матричных единиц, а *вложенный* список `list`, состоящий из *трех* списков в каждом из которых *три* матричные единицы! Таким образом, применение команды `MatrixForm` непосредственно к элементам этого списка даст нам совершенно не то, что хотелось. Команда `Flatten[list,1]` служит как раз для того, чтобы убрать *один* лишний вложенный уровень в `list`, а именно, *верхний*, и создать список из *девяти* матричных единиц (каждая из которых, конечно, сама является матрицей, т.е. вложенным списком!!) Применение команды `Flatten[list]` не достигло бы этой цели, так как убрало бы *все* вложенные уровни, создав список из 81 нуля и единицы.

• **Разреженные матрицы.** Разумеется, уже для матриц степени 100–1000 подобные определения становятся чрезвычайно грубыми и непрактичными. В языке `Mathematica` предусмотрен специальный инструмент для генерации матриц, почти все элементы которых равны между собой (скажем, равны 0 или 1). Это команда `SparseArray`, которая порождает объект специального формата **разреженный массив**, в котором хранятся не все элементы матрицы, а только ее размеры, общий элемент и список позиций и элементов в них, где они отличаются от общего элемента. Разумеется, подобный объект должен обрабатываться специальными алгоритмами, так что большая часть обычных команд для работы с матрицами будут рассматривать его как **сырой объект**. Это значит, что для проведения матричных вычислений при помощи стандартных функций необходимо еще конвертировать его в обычный матричный формат посредством команды `Normal`. Вот как, скажем, выглядит определение стандартной матричной единицы с использованием команды `Sparse Array`:

```
In[103]:=e[n_,h_,k_]:=Normal[SparseArray[{{h,k}->1},{n,n}]]
```

Разумеется, в подобных случаях `SparseArray` используется в чисто иллюстративных целях или для сокращения записи программ. Ясно, что при фактическом проведении матричных вычислений с матрицами порядка нескольких десятков или сотен тысяч, разреженные матрицы *невозможно* перевести в обычный матричный формат, так как просто для хранения нескольких матриц такого размера — не говоря уже про какие-то вычисления с ними! — в плотном формате потребуется вся доступная на бытовом компьютере память.

## § 12. ЛИНЕЙНАЯ АЛГЕБРА

1000 вещей, которые можно сделать с Вашей любимой матрицей.

Из учебника линейной алгебры

В этом параграфе мы опишем как проводить некоторые простейшие матричные вычисления. Как правило, для наглядности мы будем приводить не фактический аутпут системы `Mathematica`, а традиционную форму

матриц, которая получается, если применить к получающимся матрицам `MatrixForm`, а потом экспортировать результат в `TeX`’овском формате посредством `TeXForm`.

• **Решение систем линейных уравнений.** Решение систем линейных уравнений осуществляется при помощи функций `LinearSolve` и `NullSpace`. А именно, функция `LinearSolve` ищет частное решение неоднородной системы линейных уравнений, а функция `NullSpace` — общее решение однородной системы.

Функция `LinearSolve` вызывается в различных форматах, в зависимости от того, нужно ли нам решить одну линейную систему  $ax = b$  с данной матрицей  $a$  или (как весьма часто бывает в приложениях!) несколько систем с одной и той же матрицей и различными правыми частями. В первом случае функция `LinearSolve` вызывается в формате

```
LinearSolve[a,b]
```

в то время как во втором случае — в формате

```
LinearSolve[a][b]
```

Разница состоит в том, что во втором случае система записывает факторизацию матрицы  $a$ , которая используется для решения системы  $ax = b$ . Если матрица  $a$  не обратима, то при вызове команды `LinearSolve` в таком формате выдается сообщение:

```
LinearSolve::sing1: The matrix bla-bla-bla is singular
so a factorization will not be saved.
```

Приведем пример использования функции `LinearSolve`. Следующие команды определяют случайную матрицу порядка  $m \times n$  с целыми коэффициентами между  $-100$  и  $100$  и случайный вектор высоты  $n$  с целыми компонентами в том же интервале:

```
In[104]:=rama[m_,n_] := Table[Random[Integer,{-100,100}],
                               {i,1,m},{j,1,n}]
```

```
In[105]:=cora[n_] := Table[Random[Integer,{-100,100}],{i,1,n}]
```

А именно, вызванная в формате `Random[Integer,{k,l}]`, команда `Random` генерирует случайное целое число  $x$ ,  $k \leq x \leq l$ , естественно, каждый раз новое. Кстати, почему мы задаем `cora[n]` отдельно? Почему мы не можем просто положить `cora[n_] := rama[1,n]` или `cora[n_] := rama[n,1]`? Тот, кто внимательно прочел предыдущий параграф, знает, почему! В самом деле, `rama[m,n]` — в том числе, конечно, `rama[1,n]` и `rama[n,1]` — представляют собой вложенные списки. Для того, чтобы получить `cora[n]` в правильном формате, мы должны были бы убрать один уровень вложенности, например, применив к этим спискам команду `Flatten`.

Следующая команда фактически генерирует случайную  $4 \times 4$ -матрицу и случайный вектор высоты 4:

```
In[106]:=nnn=4;aaa=rama[nnn,nnn];bbb=cora[nnn];
```



Вот так выглядит типичный получающийся при этом результат:

$$a = \begin{pmatrix} 66 & -83 & 1 & 63 \\ -25 & -10 & -49 & 99 \\ 74 & 74 & -31 & -6 \\ -92 & 62 & 44 & 70 \end{pmatrix}, \quad b = \begin{pmatrix} 13 \\ 19 \\ -79 \\ -8 \end{pmatrix}.$$

Определитель такой случайной целочисленной матрицы грандиозен (например, в данном случае он равен 123258840). Поэтому применение команды `LinearSolve` дает

```
In[107]:=LinearSolve[aaa][bbb]
```

```
Out[107]={-1056071/2054314,-3961079/6162942,
          -597458/3081471,-202933/2054314}
```

В то же время команда `NullSpace` возвращает какую-то фундаментальную систему решений уравнения  $ax = 0$ . Вот, к примеру, что получается при применении этой команды к уже встречавшейся нам в предыдущем параграфе матрице `test[4]`:

```
In[108]:=NullSpace[test[4]]
```

```
Out[108]={{-1,0,0,1},{-1,0,1,0},{-1,1,0,0}}
```

Часто хочется увидеть общее решение неоднородной системы, совмещающее ее частное решение и общее решение соответствующей неоднородной системы. Для этого нужно свести вместе результаты работы команд `LinearSolve` и `NullSpace`. Традиционную форму такого общего решения можно породить, например, при помощи следующей конструкции:

```
In[109]:=gensolution[g_,b_]:=StringForm["'+'",
      MatrixForm[LinearSolve[g,b]],
      Table[a[i],{i,1,Length[NullSpace[g]]}]] .
      Map[MatrixForm,NullSpace[g]]]
```

Здесь использована одна из важнейших команд форматирования вывода `StringForm`. Мы часто пользуемся этой командой в тех случаях, когда нам нужно включить результат вычисления в текст. Эта команда служит для включения результатов вычисления в текстовый объект и вызывается в следующем формате:

```
StringForm["ccc'ccc'ccc...",expression1,expression2,...]
```

где, как мы уже знаем, заключенный в двойные кавычки "... " объект рассматривается как стринг, `ccc` обозначает произвольную комбинацию знаков, а каждая пара аксанов ' ' заменяется на *значение* очередного выражения `expression`.

Теперь вычисление

```
In[110]:=gensolution[test[4],{1,1,1,1}]
```

даст нам следующий результат

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + a_1 \begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \end{pmatrix} + a_2 \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + a_3 \begin{pmatrix} -1 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

• **Умножение матриц.** В отличие от покомпонентного умножения `*` или, в полной форме `Times`, обычное матричное умножение представляющее композицию линейных отображений, обозначается `.` или, в полной форме `Dot`. Таким образом, `x.y` или `Dot[x,y]` выражает произведение матриц  $x$  и  $y$ . Например,

```
In[111]:={{a,b},{c,d}}.{{e,f},{g,h}}
Out[111]:={{a*e+b*g,a*f+b*h},{c*e+d*g,c*f+d*h}}
```

Чтобы проиллюстрировать матричные вычисления на содержательном примере, свяжем с каждой функцией  $f$  матрицу Тэйлора:

```
In[112]:=taylor[f_,x_,n_]:=
Table[If[i<=j,D[f,{x,j-i}]/(j-i)!,0},{i,1,n+1},{j,1,n+1}]
```

Взглянем на матрицу Тэйлора степени 5 — с точки зрения анализа, где нумерация производных начинается с нуля, эта матрица является матрицей Тэйлора порядка 4:

$$T(f, 4) = \begin{pmatrix} f(x) & f'(x) & f''(x)/2 & f'''(x)/6 & f''''(x)/24 \\ 0 & f(x) & f'(x) & f''(x)/2 & f'''(x)/6 \\ 0 & 0 & f(x) & f'(x) & f''(x)/2 \\ 0 & 0 & 0 & f(x) & f'(x) \\ 0 & 0 & 0 & 0 & f(x) \end{pmatrix}$$

Вычислим теперь произведение двух матриц Тэйлора:

```
In[113]:=Simplify[taylor[f[x],x,4].taylor[g[x],x,4]] // MatrixForm
```

Мы не воспроизводим результат этого вычисления ввиду его громоздкости, но у каждого, кто когда-нибудь видел формулу Лейбница дифференцирования произведения, в этот момент закрадываются сильные подозрения. После следующего вычисления

```
In[114]:=Timing[Simplify[taylor[f[x],x,50].taylor[g[x],x,50]]==
taylor[f[x]*g[x],x,50]]
```

```
Out[114]:{3.034Second,True}
```

эти подозрения незамедлительно переходят в уверенность. В самом деле, две матрицы степени 51 не могут совпасть по случайной причине! Это значит, что от нас в курсе анализа скрывали нечто весьма существенное, а именно то, что многочлен Тэйлора  $n$ -го порядка произведения двух функций равен произведению их многочленов Тэйлора того же порядка:

$$T(fg, n) = T(g, n)T(f, n).$$

Разумеется, многочлен Тэйлора  $n$ -го порядка следует здесь понимать *с точностью до бесконечно малых более высокого порядка* или, как сказал бы алгебраист, **по модулю**  $x^{n+1}$ .

То, что мы сейчас увидели — это как раз типичный пример грамотного использования `Mathematica`. А именно, опытный пользователь спрашивает

у системы то, что он действительно хочет узнать, в данном случае, равна ли матрица Тэйлора функции  $fg$  *произведению* матриц Тэйлора функций  $f$  и  $g$ . Но для этого совершенно не обязательно фактически смотреть на сами эти матрицы или их произведение!! Неумелое использование, которым грешат не только начинающие, но и многие авторы учебных текстов, состоит в том, чтобы показывать то, что с точки зрения окончательной цели является промежуточным результатом. Например, выводить на экран матрицу `taylor[f[x],x,50].taylor[g[x],x,50]`.

• **Обращение матриц.** Обратная к  $g$  матрица  $g^{-1}$  вычисляется при помощи функции `Inverse`. Прежде всего проверим, что мы правильно понимаем, в каком формате вызывается `Inverse`:

```
In[115]:=Inverse[{{a,b},{c,d}}]
Out[115]={{d/(-b*c+a*d),-b/(-b*c+a*d)},{-c/(-b*c+a*d),a/(-b*c+a*d)}}
```

Проиллюстрируем теперь использование этой функции на содержательном примере. Следующая трехдиагональная матрица является одной из самых знаменитых и важных в математике. Чистым математикам она известна как **матрица Картана**<sup>68</sup>, а прикладным — как **матрица конечных разностей**<sup>69</sup>:

```
In[116]:=cartan[n_]:=Table[Which[i==j,2,Abs[i-j]==1,-1,True,0],
                               {i,1,n},{j,1,n}]
```

Обратите внимание на использование для задания этой матрицы **условного оператора** `Which`. Встречавшийся нам ранее оператор `If[test,x,y,z]`, выбирает одну из трех возможностей  $x, y, z$  в зависимости от трех значений истинности теста `test`, в следующем порядке: `True`, `False`, `Undecided`. В отличие от него оператор `Which` вызывается в формате

```
Which[test1,x1,test2,x2,test3,x3,...]
```

Этот оператор *поочередно* оценивает истинность каждого из тестов `test1`, `test2`, `test3`, ... и возвращает `xi` для *первого* из них, который принимает значение `True`. В нашем примере последний тест *всегда* выдает значение `True`, так что коэффициент матрицы в позиции  $(i, j)$  равен 0, если пара  $(i, j)$  не проходит ни одного из предыдущих тестов. Примерно так же работает и **переключатель** `Switch`, но, в отличие от условного оператора `Which`, он вызывается в другом формате и проверяет не прохождение теста, а совпадение выражения с одной из предписанных форм или его соответствие предписанному паттерну. Так, скажем,

```
Switch[Mod[i-j,4],0,a,1,b,2,c,3,d]
```

принимает значение  $a, b, c$  или  $d$  в соответствии с тем, чему равен вычет  $i - j$  по модулю 4.

<sup>68</sup>Н.Бурбаки,

<sup>69</sup>Р.Грегори, Е.Кришнамурти, Безошибочные вычисления. Методы и приложения. — М., Мир, 1988, с.1–208.

Посмотрим для примера на матрицу `cartan[8]`:

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

Обратная к этой матрице хорошо известна и очень часто используется. Изобразим для примера `9*Inverse[cartan[8]]`:

$$\begin{pmatrix} 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 7 & 14 & 12 & 10 & 8 & 6 & 4 & 2 \\ 6 & 12 & 18 & 15 & 12 & 9 & 6 & 3 \\ 5 & 10 & 15 & 20 & 16 & 12 & 8 & 4 \\ 4 & 8 & 12 & 16 & 20 & 15 & 10 & 5 \\ 3 & 6 & 9 & 12 & 15 & 18 & 12 & 6 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

• **Определитель матрицы.** Определитель матрицы  $g$  вычисляется при помощи функции `Det`. Как всегда, прежде всего стоит проверить на совсем простом примере, правильно ли мы понимаем ее использование:

```
In[117]:=Det[{a,b},{c,d}]
```

```
Out[117]= -b*c+a*d
```

Обратимся теперь к более интересным примерам. Мы уже говорили, что теория определителей была придумана для анализа *единственного* примера — определителя Вандермонда  $\det(V(x_1, \dots, x_n))$ . Так вот с него и начнем. Вычисление `Length[Det[vandermonde[y,5]]]` показывает, что `Mathematica` считает, что этот определитель является суммой 120 слагаемых — КАК В ЭТОМ МИРЕ ВСЕ ТЯЖЕЛО, ВСЕ ГОРЕСТИ ПОЛНО!!!<sup>70</sup> Однако применяя к тому же определителю команду `Simplify`, мы получим уже вполне осмысленный результат:

```
In[118]:=Simplify[Det[vandermonde[y,5]]]
```

```
Out[118]=(y[1]-y[2])*(y[1]-y[3])*(y[2]-y[3])*(y[1]-y[4])*(y[2]-y[4])*
(y[3]-y[4])*(y[1]-y[5])*(y[2]-y[5])*(y[3]-y[5])*(y[4]-y[5])
```

<sup>70</sup>“Deus,” dist li reis, “si penuse est ma vie!!!” — La chanson de Roland.

Команда `Minors[x]` порождает все **дополнительные миноры** матрицы  $x$ . Однако упорядочивает она их лексикографически по порядку *включенных* в них строк и столбцов (а вовсе не исключенных, как это делается в элементарных учебниках линейной алгебры!) Следующее вычисление показывает дополнительные миноры в матрице степени 3:

```
In[119]:=fancy1=Partition[CharacterRange["a","i"],3]
Out[119]={{a,b,c},{d,e,f},{g,h,i}}
In[120]:=Minors[fancy1]
Out[120]={{-b*d+a*e,-c*d+a*f,-c*e+b*f},{-b*g+a*h,-c*g+a*i,-c*h+b*i},
           {-e*g+d*h,-f*g+d*i,-f*h+e*i}}
```

Обратите внимание на то, как мы порождаем матрицу степени 3:

◦ Команда `CharacterRange["a","i"]` генерирует список из 9 букв, лежащих между `a` и `i`, включительно. Отметим необходимость использования здесь кавычек! Эти кавычки означают, что `a` и `i` должны трактоваться не как символы, а как **стринг**т.е. *текстовые объекты*, воспринимаемые *verbatim*. Большинство обычных команд трактует стринг как **сырой объект** и не проводит с ним никаких вычислений. В Выпуске 4 мы среди прочего обсуждаем специальные текстовые команды, которые позволяют проводить со стрингами все обычные манипуляции, аналогичные обычным манипуляциям со списками.

◦ Команда `Minors[x,m]` порождает все миноры порядка  $m$  матрицы  $x$ , в лексикографическом порядке.

```
In[121]:=fancy2=Partition[CharacterRange["a","p"],4]
Out[121]={{a,b,c,d},{e,f,g,h},{i,j,k,l},{m,n,o,p}}
```

Теперь вычисление

```
In[122]:=Minors[fancy2,2]
```

дает следующий результат:

$$\begin{pmatrix} -be + af & -ce + ag & -de + ah & -cf + bg & -df + bh & -dg + ch \\ -bi + aj & -ci + ak & -di + al & -cj + bk & -dj + bl & -dk + cl \\ -bm + an & -cm + ao & -dm + ap & -cn + bo & -dn + bp & -do + cp \\ -fi + ej & -gi + ek & -hi + el & -gj + fk & -hj + fl & -hk + gl \\ -fm + en & -gm + eo & -hm + ep & -gn + fo & -hn + fp & -ho + gp \\ -jm + in & -km + io & -lm + ip & -kn + jo & -ln + jp & -lo + kp \end{pmatrix}$$

• **Собственные числа и векторы.** Собственные числа матрицы ищутся при помощи команды `Eigenvalues`, а соответствующие им собственные векторы — при помощи команды `Eigenvectors`. Их можно найти одновременно при помощи команды `Eigensystem`.

В следующем вычислении мы предлагаем системе вычислить собственные числа и собственные векторы матрицы Фибоначчи  $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ :

```
In[123]:=Eigensystem[{{1,1},{1,0}}]
```

```
Out[123]={{1/2*(1+Sqrt[5]),1/2*(1-Sqrt[5])},
           {{1/2*(1+Sqrt[5]),1},{1/2*(1-Sqrt[5]),1}}}
```

Уже нахождение собственных чисел и векторов целочисленных  $3 \times 3$ -матриц может быть нетривиальной вычислительной проблемой. Одним из интереснейших примеров является матрица кубического золотого сечения<sup>71</sup>

$$\begin{pmatrix} 3 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Функция `CharacteristicPolynomial` вычисляет характеристический многочлен матрицы:

```
In[124]:=CharacteristicPolynomial[{{3,2,1},{2,2,1},{1,1,1}},x]
Out[124]=1-5*x+6*x^2-x^3
```

Однако, как мы уже видели в § 3, корни этого многочлена выглядят достаточно хитро.

• **Функции от матриц.** Начинаящий должен быть особенно осторожен в том, что касается функций, вычисление которых зависит от используемого умножения. Дело в том, что большинство обычных функций по умолчанию используют *покомпонентное* умножение `Times`, называемое в теории матриц умножением по Адамару или умножением по Шуру. В то же время, для правильного вычисления функций от матриц необходимо всюду использовать обычное матричное умножение `Dot`. Классическая теория Кэли—Сильвестра—Фробениуса функций от матриц как раз и основана на том, что для *диагональных* матриц эти два умножения совпадают.

Переводя разговор в практическую плоскость, это значит, например, что для вычисления степени матрицы нужно использовать не функцию `Power`, а функцию `MatrixPower`, для вычисления экспоненты от матрицы — не функцию `Exp`, а функцию `MatrixExp` и т.д.

Вот, например, вычисление экспоненты числа  $1 + \sqrt{2}$ , выраженного целочисленной матрицей  $\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$ :

```
In[125]:=MatrixExp[{{1,2},{1,1}}]
Out[125]={{1/2*E^(1-Sqrt[2])*(1+E^(2*Sqrt[2])),
           E^(1-Sqrt[2])*(-1+E^(2*Sqrt[2]))/Sqrt[2]},
           {E^(1-Sqrt[2])*(-1+E^(2*Sqrt[2]))/(2*Sqrt[2]),
           1/2*E^(1-Sqrt[2])*(1+E^(2*Sqrt[2]))}}}
```

Интересно, что применение `Simplify` не упрощает это выражение, но вот применение `FullSimplify` дает

```
In[126]:=FullSimplify[MatrixExp[{{1,2},{1,1}}]]
Out[126]={{E*Cosh[Sqrt[2]],Sqrt[2]*E*Sinh[Sqrt[2]]}
           {E*Sinh[Sqrt[2]]/Sqrt[2],E*Cosh[Sqrt[2]]}}
```

<sup>71</sup>В.И.Арнольд, Что такое математика? — М., МЦНМО, 2004, с.1–104; с.99.



Ясно, что при рассмотрении матриц степени 1000 подобных эффектов можно достичь изменением одного элемента на  $10^{-1000}$ .

Вычислительная линейная алгебра уже более столетия развивает специальные приемы для борьбы с такого рода явлениями: масштабирование, изучение обусловленности, контроль ошибок. Однако для большинства практически встречающихся случаев самым простым и надежным лекарством является ПОЛНЫЙ ОТКАЗ ОТ ПРИБЛИЖЕННЫХ ВЫЧИСЛЕНИЙ.