Ю. И. Рыжиков

ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ

- Традиционные разделы вычислительной математики, усиленные новыми подходами
- Методология математического моделирования
- Способы эффективного программирования
- Приемы и примеры реализации типовых вычислительных алгоритмов
- Работа с матрицами специальной структуры
- Задачи оптимизации
- Математические библиотеки и пакеты научных подпрограмм

УЧЕБНОЕ ПОСОБИЕ

bhv

Ю. И. Рыжиков

ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ

Рекомендовано учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов, обучающихся по направлению 230100 — «Информатика и вычислительная техника»

Санкт-Петербург
2007

УДК 519.6(075.8)
ББК 22.193я73
Р94

Рыжиков Ю. И.

Р94 Вычислительные методы. — СПб.: БХВ-Петербург, 2007. — 400 с.: ил.

ISBN 978-5-9775-0137-8

Книга написана инженером специально для инженеров и посвящена основам решения инженерных задач с акцентом на программную реализацию методов вычислительной математики. Включает в себя постановку задачи математического моделирования, описание вычислительных алгоритмов линейной алгебры, приближения функций, численного дифференцирования и интегрирования. Приводятся приемы эффективного программирования, описаны математические пакеты и библиотеки. Текст книги сопровождается программами или их фрагментами, таблицами и графиками.

Для студентов технических вузов, аспирантов и инженеров

УДК 519.6(075.8)
ББК 22.193я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Татьяна Лапина*
Зав. редакцией	*Григорий Добин*
Компьютерная верстка	*Юрия Рыжикова*
Корректор	*Людмила Минина*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

Рецензенты:

П. В. Герасименко, заслуженный деятель науки РФ, доктор технических наук, профессор Санкт-Петербургского университета путей сообщения

А. Я. Перельман, доктор физико-математических наук, профессор Санкт-Петербургской лесотехнической академии

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 16.07.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 32,25.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29

ISBN 978-5-9775-0137-8

Оглавление

Введение

«Мы знаем что-то, если можем это запрограммировать».
А. П. Ершов

0.1. Потребности в математике

Стремительный прогресс научных знаний и техники XX века — освобождение внутриядерной энергии, освоение космического пространства, раскрытие тайн наследственности и т. д. — не только является воплощением в жизнь самых смелых фантазий, но и обгоняет их. Объем научных исследований в последнее время удваивается каждые 10 лет, а количество ныне живущих работников науки примерно равно числу ученых за всю историю человечества. Как старые, так и возникшие уже в XX столетии новые отрасли знания бурно развиваются и постоянно рождаются все новые: кибернетика, квантовая теория поля, биофизика, космическая биология, механика космического полета, формальная генетика, структурная лингвистика и т. д. Развиваются геофизические методы поиска полезных ископаемых, мерзлотоведение.

Интенсивно ведется расшифровка структуры вещества по экспериментальной информации, полученной методами рентгенографии, электронографии и нейтронографии. Решаются стереохимические задачи, ведутся работы по моделированию химического анализа и синтеза. Изучается структура белков, динамика популяций и биоценозов. Этот перечень можно продолжать практически бесконечно.

Основной и наиболее характерной особенностью современной науки является растущая *математизация* ее. Примером могут служить перечисленные выше новые дисциплины, а также проникновение математики в экономику, медицину, педагогику, психологию, лингвистику, теорию

искусства и т. п. Все эти факты подтверждают известное положение: «Наука только тогда достигает совершенства, когда ей удается пользоваться математикой».

Основой столь широкого использования математических методов является соответствие между свойствами физических и абстрактных математических объектов. Математически описав количественные связи между элементами некоторого явления, исследователь получает его абстрактную модель. Формальный анализ модели позволяет установить законы изменения переменных и другие необходимые свойства математического решения, а затем с их помощью получить новое знание об исходном физическом объекте. Математическое моделирование облегчается тем, что одни и те же зависимости могут описывать явления различной физической природы (например, изменение тока в RC-контуре, колебания сжатой пружины и математического маятника).

0.2. Специфика современного этапа

0.2.1. Состояние математики

Математика древнего мира имела дело со сравнительно простыми (с современной точки зрения) задачами — главным образом из области измерения площадей и объемов, а также астрономии. Математический аппарат, возникший сначала с развитием алгебры, а затем дифференциального и интегрального исчисления, дал в руки естествоиспытателей и инженеров мощное средство моделирования действительности, без которого не было бы современной науки и техники. Однако использование этого аппарата далеко не всегда приводило к простым и изящным формулам. Уже давно выяснилось, что:

- многие неопределенные интегралы не могут быть выражены через элементарные функции;
- многие уравнения, в частности — уравнения вида

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n = 0$$

при $n \geq 5$, в общем случае неразрешимы в радикалах, и точные значения их корней не могут быть получены выполнением конечного числа элементарных действий;

- лишь редкие дифференциальные уравнения имеют решения в элементарных функциях.

Однако решение не обязательно должно иметь аналитический вид. В конце концов для практики нужны числовые результаты при конкретных наборах исходных данных, причем эти результаты могут иметь некоторую погрешность. Следовательно, практическим целям не хуже формул могут служить *алгоритмы* более общего вида, т. е. совокупности формальных предписаний, однозначно определяющих результат вычислений при заданном наборе исходных данных. Численные методы часто оказываются универсальнее аналитических. Так, формула

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} A_i f(x_i)$$

при достаточно большом n и надлежащем выборе весовых коэффициентов $\{A_i\}$ и узлов интегрирования $\{x_i\}$ может обеспечить любую наперед заданную точность для произвольной (достаточно гладкой) подынтегральной функции. Методы численного решения математических задач путем сведения их к последовательности арифметических операций и составляют предмет *вычислительной математики* (численного анализа).

Становление вычислительной математики происходило параллельно с развитием «чистой» математики, также связанной с именами Ньютона, Лейбница, Стирлинга, Бернулли, Эйлера, Лагранжа и др. Крупнейшую роль в ее развитии играли выдающиеся русские ученые П. Л. Чебышев и Н. И. Лобачевский. Первый в мире курс приближенных вычислений был прочитан акад. А. Н. Крыловым, которому принадлежат и важные научные результаты в этой области. Большой вклад в вычислительную математику внесли и советские ученые.

Математика наших дней интенсивно развивается под действием как внутренних, так и внешних стимулов. Внутренние стимулы двигают главным образом чистую математику в направлениях логического завершения, обобщения и взаимопроникновения ранее созданных теорий — без непосредственной связи с приложениями. Внутриматематическая ценность этих работ несомненна: они способствуют развитию математики в целом. «Математика — это большой город, чьи предместья не перестают разрастаться несколько хаотическим образом на окружающем его пространстве, в то время как центр периодически перестраивается, следуя каждый раз все более ясному плану и стремясь ко все более и

более величественному расположению, в то время как старые кварталы с их лабиринтом переулков сносятся для того, чтобы проложить к окраине улицы все более широкие, все более удобные» (Н. Бурбаки, [61, с. 18]). Однако «экспорт» новейших идей предельной общности в приложения и особенно в математическое образование прикладников должен осуществляться с крайней осторожностью. Как отмечено в [11], сохраняется тенденция «обурбачивания» все новых разделов математики, в частности вычислительной. Отойдя от традиций Эйлера и других математиков «золотого периода», математики теоретико-множественного направления *перестали вычислять*. Утверждения о решении носят «экзистенциальный» характер — без указания пути к нему. Асимптотические оценки даются без указания константы или с неимоверным завышением ее. Как в России (ранее — в СССР), так и на Западе «учителя, употреблявшие при каждом удобном случае слово "множество", считают себя пионерами педагогических изысканий» [94, с. 10].

С другой стороны, математизация новых разделов науки и потребности практики неизбежно оказывают обратное влияние на математику, прежде всего прикладную. В частности, появились новые методы, приспособленные для решения задач большой размерности, некорректных, плохо обусловленных, связанных с «неудобными» комбинациями параметров («жесткие» системы дифференциальных уравнений), многомерной оптимизации, проблем машинной графики (сплайны). Прикладные исследования имеют непосредственную отдачу; это усиливает доверие общества к математике, расширяет понимание ее проблем и способствует вложению средств в ее развитие.

0.2.2. Развитие ЭВМ

В наши дни сколько-нибудь серьезные вычисления выполняются только на вычислительных машинах — под управлением реализующих вычислительные методы программ. Рост технических характеристик ЭВМ обгоняет самую бурную фантазию. Менее чем за 50 лет быстродействие процессора возросло на 10 порядков, тогда как скорость передвижения человека (от пешехода до космонавта) — лишь в 5000 раз [10]. Не менее важно и увеличение доступных объемов оперативной памяти — автор никогда не забудет свои мучительные попытки уложить программу и данные в 4000 слов М-20 в начале 1960-х гг. Наконец, возросла надежность вычислительных машин, средние интервалы между отказами которых измеряются уже не единицами часов, а годами. Появились

векторные вычислители, эффективно реализующие накопление скалярных произведений и иные матричные операции, мультипроцессорные и многомашинные вычислительные комплексы.

В результате некоторые аспекты вычислительной практики для типовых случаев радикально трансформировались. Практически потеряли смысл вопросы о разрядности промежуточных вычислений и оперативном контроле их правильности; уменьшилась важность минимизации объема вычислений и требуемой оперативной памяти. Тем не менее в связи с вышеперечисленными тенденциями в изменении решаемых задач сохраняют актуальность проблемы работы с упакованными матрицами специальной структуры, ускорения сходимости итерационных процессов решения уравнений и оптимизации. Возрос интерес к *распределенным* вычислениям и численным методам, приспособленным для них.

0.2.3. Программирование вычислений

Хорошо известна стандартная последовательность действий пользователя ЭВМ:

1) содержательная постановка задачи;

2) математическая формулировка ее;

3) сбор исходных данных;

4) дискретизация задачи (выбор численного метода);

5) программирование;

6) отладка программы;

7) рабочий счет;

8) обработка результатов.

Однако их содержание претерпело заметные изменения. Расширился арсенал численных методов; изменились критерии предпочтительности (к примеру, задачи нелинейной оптимизации сейчас решаются преимущественно методами поиска — без использования теоремы Куна—Таккера).

Опыт программирования показывает, что качество программы в решающей степени зависит от выбранного алгоритма и в значительно меньшей — от примененных приемов программирования. Например, решение

системы из 10 линейных алгебраических уравнений методом Гаусса требует примерно в 300 000 раз меньше арифметических операций, чем применение известных из общего курса математики формул Крамера (в предположении, что каждый определитель вычисляется как сумма произведений по 10 множителей в каждом). Это объясняет, почему хороший программист должен знать вычислительную математику.

Появилось множество языков программирования, в той или иной степени пригодных для описания численных алгоритмов, в связи с чем возникла проблема выбора рабочего языка. Высший приоритет получили ясность и логическая простота программы.

Многие типовые алгоритмы вычислительной математики реализованы в составе библиотек соответствующих систем программирования или пакетов прикладных программ. Все такие пакеты обеспечивают матричные вычисления с помощью *базисного набора* алгоритмов линейной алгебры — в частности, представления матриц общего вида в виде произведения специальных. Для интерполяции они используют эрмитовы кубические кривые и кривые Безье, для вычисления определенных интегралов рекомендуют адаптивные алгоритмы или формулы Кронрода.

В численных методах всегда имеется бездна ловушек, о которых пользователь должен быть предупрежден — чтобы он мог правильно диагностировать признаки «численного нездоровья» [37, с. 15]. Подпрограммы пакетов проверяют приемлемость входных параметров и выдают предупреждающие сообщения. Универсальные или применимые к очень широкому кругу задач процедуры, как правило, имеют низкую точность и медленную сходимость к точным значениям. Это требует определенного уровня понимания используемых методов.

Созданы интегрированные среды для решения частных математических задач, в том числе в символьной форме (Mathematica, Maple, MathCAD, MatLab, Derive, Scientific WorkPlace и др.). Предложены новые технологии программирования (в частности, объектно-ориентированное), имеющие целью уменьшить трудоемкость разработки и количество ошибок в программах.

0.2.4. «Вычислительные кадры»

Выше уже отмечался прогресс, достигнутый в последние десятилетия по всему фронту компьютерных наук (Computer Science). К сожалению, этого нельзя сказать о компьютерном (фактически — обо всем) образовании. Общей тенденцией (со стороны как преподавателей, так и

студентов) стало натаскивание на обеспечение сиюминутных потребностей: дай результат здесь и сейчас, не вникая в способ его получения. Это начинается еще в школе, выпускники которой в своей массе не умеют думать и панически боятся минимального умственного усилия[1].

Слабое знание математики основной массой пользователей определяет преобладающее использование самых примитивных методов. Приведем опубликованные на международной конференции по численному программному обеспечению статистические данные о Токийском университете: из 3094 случаев численного интегрирования в 118 применялся метод трапеций, в 2461 — Симпсона, в 411 — Гаусса и только в 104 — правило Ромберга. Таким образом, 80% пользователей не пошли дальше формул Симпсона. Ненамного лучше обстояло дело с линейной алгеброй: здесь 2/3 задач решались методом Гаусса.

Таблица 1. Решение систем линейных уравнений

Метод	Кратность
Гаусс (без выбора)	2200
Гаусс (с выбором)	2380
Гаусса—Жордана	31
LU-декомпозиция	1230
Холецкого	390
Наименьших квадратов	390
Всего	6621

Под выбором понимается выбор ведущего элемента.

Огромный вред принесла концепция «непрограммирующих» пользователей, породившая армию неспособных к самоусовершенствованию компьютеризованных дебилов. В том же русле находятся попытки *заменить* преподавание вычислительной математики решением задач в одной или нескольких из упомянутых математических сред.

0.3. Как учить вычислительной математике

Перечислим первоочередные требования к целевой установке, содержанию и методике обучения вычислительной математике (ВМ).

[1] Автор 40 лет преподает в инженерном вузе и постоянно наблюдает эту удручающую тенденцию.

1. Студент должен знать базовые понятия ВМ и их взаимные связи, идейные основы и расчетные схемы типовых алгоритмов, сравнительные достоинства и области предпочтительного применения последних, способы контроля правильности результатов, требования к программному продукту.

2. Знание алгоритмов должно проверяться *умением запрограммировать* указанный метод (см. эпиграф к Введению). Автор имел честь участвовать в заседании Санкт-Петербургского математического общества, которое постановило «мочить на месте» (В. В. Путин) преподавателей, заставляющих вручную обращать матрицы четвертого порядка.

3. Доводить результаты до числа, сравнивать альтернативные способы решения задачи (например, тип и порядок квадратурных формул) нужно с помощью программ или в математических средах. С работой в последних следует знакомить на *практических* занятиях.

0.4. О данной книге

Вычислительная математика (ВМ) — одна из немногих областей науки, по которой продолжается переиздание классики и выпуск новой литературы. Переиздания и книги, ориентированные на стандартный втузовский учебный план, как правило не учитывают обсуждавшиеся выше особенности современного этапа практики вычислений. Значительная часть литературы ориентирована на физико-математические специальности. Эти книги чрезмерно усложнены, перегружены доказательствами и уделяют явно недостаточное внимание реализации методов. Известно, как упорно боролся со «слишком чистыми математиками» Л. Д. Ландау, доказывая необходимость «считающих» методов.

Опасна и противоположная крайность — уклон в чисто рецептурную сторону «информационных технологий» и натаскивание студентов на работу с математическими пакетами без понимания сути, взаимосвязей, возможностей, ограничений и опять же — *реализации* используемых методов.

В связи с обоими равно непродуктивными «уклонами» хочется напомнить о давней и плодотворной традиции изучения ВМ и программи-

рования в комплексе (*Мак-Кракен Д., Дорн У.* Программирование на Фортране. — М.: Мир, 1969). Такое объединение весьма ценно и для преподавания *программирования*, поскольку обеспечивает постановку реальных и содержательных задач.

Отметим, наконец, что необъятность проблемы, разнообразие специальностей и личного научного и педагогического опыта преподавателей вполне оправдывают — более того, требуют — существования *набора* учебников по ВМ.

Автор каждой книги по классическому направлению науки обязан сопоставить ее с аналогичными публикациями последних лет. Большинство новых или переизданных учебников [10, 15, 46, 88, 102] при всем уважении к научной квалификации их авторов ориентировано на студентов физико-математических специальностей и к знанию на таком уровне не подводит. Они перегружены длинными выкладками и обсуждением частностей. Стиль их написания соответствует давно прошедшим временам, когда в прикладном программировании четко разделялись функции программистов-кодировщиков и постановщиков задачи. Теперь (по крайней мере для студента и молодого специалиста) они совмещены, а серьезному программированию расчетных задач никто не учит. Из обсуждаемого списка только в [41, 44] приводятся примеры Паскаль-программ. Такие примеры обычно тривиальны и рациональным приемам программирования не учат. Книга С. В. Поршнева [76] заявленного права называться курсом лекций никак не заслуживает, поскольку примерно половина ее описывает работу с пакетом MatLab, т. е. посвящена технологии *практических занятий*, а уровень ее теоретической части становится ясным уже из написания фамилии классика (Koshi вместо Cauchy – с. 159). Книга [47] позиционирована как написанная по совету акад. А. А. Самарского для параллельного преподавания программирования и вычислительной математики (заметьте: *параллельного*, т. е. разными преподавателями, а не совмещенного).

В отличие от большинства учебников по численным методам, *эта книга* написана инженером специально для инженеров. Предлагаемый вариант учитывает вышеупомянутые особенности состояния теоретических основ ВМ, развития вычислительной техники, численного программного обеспечения и контингента обучаемых. Нашей целью было пояснить основные идеи математических методов и общие закономерности рассматриваемых явлений. Напротив, формальные доказательства, рассмотрение исключений и усложняющих факторов по возможности опущены. Основной целью было воспитать у студентов прикладную ма-

тематическую культуру, развить необходимые эрудицию и интуицию в вопросах применения математики, а также логическое и алгоритмическое мышление. Хотелось бы убедить их в том, что машина, освобождая нас от многих обязанностей, не освобождает от трех: творчески мыслить, владеть математическим аппаратом и программировать.

О содержании книги достаточно ясное представление дает оглавление. Здесь мы лишь обратим внимание на вошедшие в нее «нетипичные» разделы: обзор современного Фортрана; многочисленные примеры фортрановских реализаций алгоритмов или их частей; фортрановские научные библиотеки; приемы рациональной организации вычислений многочленов, степенных рядов и непрерывных дробей, определенных интегралов; работу с разреженными матрицами регулярной структуры; решение некоторых редко обсуждаемых классов матричных уравнений.

В книгу не вошли вопросы решения краевых задач, дифференциальных уравнений в частных производных, теории оптимального управления, быстрого преобразования Фурье. На наш взгляд, в инженерных вузах эти проблемы слишком сильно окрашены конкретными приложениями, и их предпочтительно излагать в составе специальных дисциплин.

В раздел оптимизации включены только те методы нелинейного программирования, которые по своему математическому аппарату непосредственно примыкают к вычислительной математике. Соответственно, оставлены в стороне специфические проблемы линейного и динамического программирования и задачи на графах.

Дополнительные сведения по всем обсуждаемым (и оставшимся «за бортом») вопросам можно почерпнуть из источников, перечисленных в списке литературы.

В связи с «программистской» ориентацией учебника степень подробности числовых примеров не ориентирована на ручной счет. По той же причине целая часть числа отделяется от дробной *точкой*, а в роли разделителя мантиссы и экспоненциальной части используется привычное для пользователя ЭВМ «e». Все доказательства теорем завершаются знаком ▲.

Автор много сил затратил на попытки унификации обозначений, но успеха не достиг. В качестве примера возникших трудностей сошлемся на необходимость различения в многомерных задачах номера итерации и номера компоненты вектора. Перенос последней в верхний индекс вызывал постоянные затруднения из-за необходимости в показателях степени, указателях транспонирования и т. п. Другой проблемой явилось шрифтовое различение скаляров, векторов и матриц. Хотя набор век-

торов полужирным шрифтом весьма распространен, встречаются книги, где наряду с этим *векторная функция* набрана обычным шрифтом. Матрицы набираются то жирным шрифтом, то обычным, а в классической книге Ф. Р. Гантмахера шрифтовые выделения для векторов и матриц не используются вообще. Существует десяток разных обозначений для градиента. Разного рода надчеркивания, подчеркивания, тильды и «крышки» скорее затрудняют восприятие, чем помогают читателю. Кроме того, случайный пропуск одного из заявленных украшений может изменить смысл математического выражения. В связи с изложенным автор предпочел использовать опыт вузовских педагогов: лектор у доски не может пользоваться шрифтовыми выделениями в принципе, время на упомянутые украшения жалеет и обходится самыми простыми средствами, предварительно оговорив особенности ситуации.

В разных книгах при расчете скалярного произведения комплекснозначных векторов сопряженным берется то первый, то второй вектор. Индексация коэффициентов многочлена n-й степени даже в одной книге начинается то с нуля, то с n — в зависимости от решаемой задачи и в целях большей понятности происходящего. Буквой Δ обозначаются как погрешность, так и приращение переменной. Этот перечень можно продолжать бесконечно.

Автор не решился взять на себя ответственность за «наведение порядка» в ВМ (в частности, потому, что предлагать в этой сфере радикальные новшества значит потерять читателя вообще) и признает себя, но не только себя, повинным во всех перечисленных грехах.

Автор посвящает эту книгу светлой памяти своего учителя и соавтора по [92] доктора физико-математических наук профессора Х. Л. Смолицкого (1912–2003).

Глава 1. Содержание и средства математического моделирования

1.1. Математическое моделирование

Любая целенаправленная деятельность предполагает оценку результатов действий. Метод проб и ошибок в этом смысле всегда является наихудшим. Гораздо выгоднее (дешевле, быстрее, безопаснее) провести моделирование интересующего нас явления или процесса и принять требуемое решение (в науке — гипотезу) на основе анализа поведения *модели*. Необходимость в моделировании возникает также, если объект слишком велик (корабль) или мал (атом); удален от исследователя (галактика); недосягаем во времени (прошлое или будущее); дорог.

Основной целью моделирования является постановка над моделью экспериментов с последующей *интерпретацией* их результатов для моделируемой системы. Модель *концентрирует* в себе записанную на определенном языке (естественном, математическом, алгоритмическом) совокупность наших знаний, представлений и гипотез о соответствующем объекте или явлении. Поскольку эти знания никогда не бывают абсолютными, а гипотезы могут вынужденно или намеренно не учитывать некоторые эффекты, модель лишь приближенно описывает поведение реальной системы. Важнейшая особенность модели состоит в возможности неограниченного накопления специализированных знаний без потери целостного взгляда на объект исследования. Любые научно-технические расчеты (на прочность, устойчивость, надежность, безопасность, точность и

т. п.) — важнейшие элементы профессиональной деятельности инженера — в сущности, являются специализированными видами моделирования. В [39] приводится ряд постепенно усложняемых моделей радиоэлектронных устройств — от диода до лазерного излучателя.

При исследовании сложных систем может потребоваться разработка *набора* моделей, соответствующих различным иерархическим уровням рассмотрения и функциональным аспектам деятельности системы. Такое описание на каждом уровне использует свой набор понятий и терминов.

Моделирование может быть натурным (физическим), математическим и комбинированным.

В сравнении с натурным экспериментом математическое моделирование имеет следующие преимущества:

1. Экономичность и сбережение ресурсов реальной системы.
2. Возможность разделить любое исследование на *предметные* вопросы, относящиеся к изучаемым объектам, и *логические*, разрешаемые средствами языка.
3. Возможность моделирования гипотетических, т. е. не реализованных в натуре объектов (прежде всего на разных этапах проектирования).
4. Возможность реализации режимов, опасных или трудновоспроизводимых в натуре (глобальные последствия 100-мегатонного ядерного конфликта, критический режим ядерного реактора, работа системы ПРО).
5. Возможность изменения масштаба времени.
6. Легкость многоаспектного анализа.
7. Сжатие информации и ее единообразное представление, способствующие усмотрению целого.
8. Наличие строго сформулированных правил, позволяющее:
   - вскрыть ложность некоторых предубеждений;
   - заменить содержательные рассуждения формальным преобразованием выражений — в частности, выполнить оптимизацию;
   - выявить глубинные (сущностные) свойства и отношения и как следствие получить бóльшую прогностическую силу и построить разнообразные аналогии.

9. Универсальность технического и программного обеспечения проводимой работы (ЭЦВМ, системы программирования и пакеты прикладных программ широкого назначения).

Результаты расчетов часто позволяют предсказывать и обнаруживать не наблюдавшиеся ранее явления.

Математическое моделирование на определенных этапах исследования может сочетаться с натурным. Классическим примером такого подхода является исследование динамики летательного аппарата на комплексе из математической модели движения самого аппарата, воспроизводимой на аналоговой или цифровой ЭВМ, и макета реальной аппаратуры управления. Практическое использование модели возможно лишь после тщательного ее исследования и настройки.

Моделирование требует установления *критериев подобия*, т. е. формулировки тех условий, при которых модель может считаться закономерно отражающей (в том или ином смысле) оригинал. Полным считается подобие во времени и пространстве (с отвлечением от несущественных процессов), неполным — только во времени или в пространстве. При этом возможно введение масштабов. В [3, 14] приведены сводки критериев механического, гидродинамического, теплового, электрического подобия.

Весьма поучительный пример моделирования с использованием критериев подобия приводится в воспоминаниях А. Н. Крылова о его службе в опытовом бассейне [50, с. 133]: «Я приказал вычисление сопротивления (воды — движению судна — *Ю. Р.*) делать следующим образом. Из наблюденного при работе с моделью сопротивления вычесть рассчитанное сопротивление от трения: получится остаточное сопротивление, которое умножается на куб масштаба и дает остаточное сопротивление корабля. К этому остаточному сопротивлению прибавляется рассчитанное сопротивление от трения на корабль — получится полное сопротивление корабля при скорости его, соответствующей скорости модели».

Структура моделирования показана на рис. 1.1.

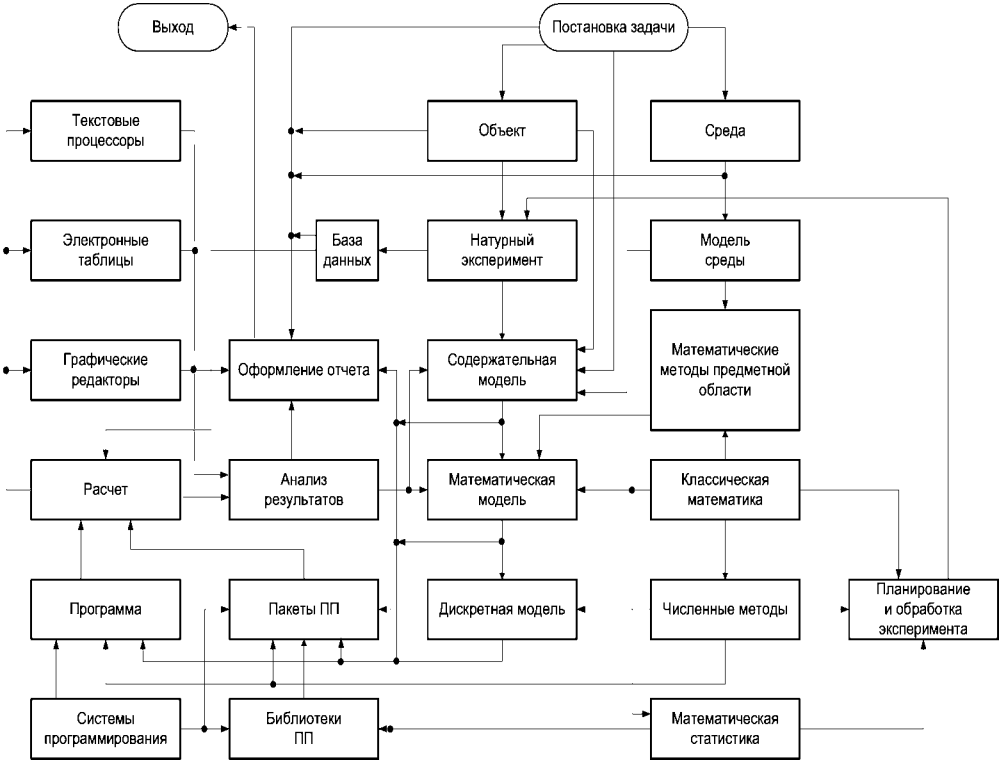

Рис. 1.1. Организация математического эксперимента

В контексте данной книги мы будем обсуждать преимущественно математические аспекты моделирования.

1.2. Виды математических моделей

Математическое моделирование можно разделить на аналитическое, имитационное и комбинированное. При *аналитическом* моделировании процессы функционирования элементов системы записывают в виде алгебраических, интегральных, дифференциальных, конечно-разностных и других соотношений и логических условий.

В круге аналитических методов прежде всего нужно выделить дающие общее представление о явлении. *Качественная* теория дифференциальных уравнений позволяет выявить структуру всех возможных решений — регулярных и особых. В ее основе лежит понятие *фазового*

портрета системы: траектории состояния в осях «переменная — производная от нее». На фазовой плоскости наглядно видны устойчивый и неустойчивый процессы (сходящаяся и расходящаяся спирали соответственно), устойчивые колебания (эллипс), аттракторы (предельные режимы) и области их притяжения и т. п. Качественные методы позволяют установить свойства спектра, монотонность зависимостей, выпуклость и т. д. Это существенно влияет на выбор метода решения.

Под *аналитическим решением* в узком смысле слова понимают явную формулу, связывающую искомую величину с параметрами модели. Ее исследование дает весьма ценную информацию. Возьмем для примера формулу Полячека—Хинчина для среднего времени ожидания заявки в системе массового обслуживания вида $M/G/1$:

$$w = \frac{\lambda b_2}{2(1-\lambda b_1)}, \tag{1.2.1}$$

где λ — интенсивность потока заявок, а $\{b_k\}$ — начальные моменты распределения длительности обслуживания. Ясно, что произведение λb_1 (коэффициент загрузки системы) должно быть строго меньше единицы, и при его устремлении к единице среднее время ожидания стремится к бесконечности. Далее, второй момент распределения обслуживания $b_2 = b_1^2 + D$, где D — дисперсия. При фиксированных λ и b_1 ожидание w минимально при регулярном обслуживании $(D = 0)$, а при переходе к обычно используемому показательному закону увеличивается ровно вдвое.

Такое решение («замкнутое») всегда предпочтительно, но вывести его обычно удается лишь после ряда упрощающих предположений. Возможность его получения весьма критична к изменениям модели; здесь требуются высокая квалификация и значительные творческие усилия разработчиков.

Решение задачи часто удается получить с помощью преобразований (Лапласа, Фурье и др.) исходных функций к виду, где легче выполняются нужные операции (дифференцирование, интегрирование, свертка). Затем выполняется обратный переход.

При исследовании процессов вариационными методами проводится оптимизация *функционала* — функции от функции, при оптимальном выборе которой можно достичь минимальных промаха, энергозатрат или времени достижения желаемого состояния.

Гораздо чаще удается применить *численные* методы — в особенности при наличии хорошей библиотеки подпрограмм. Численные методы

дают лишь частные результаты, по которым трудно делать обобщающие выводы. Для оптимизации модели необходим многовариантный счет. Если решением задачи является функция непрерывной переменной, то эта функция часто заменяется сеточной функцией, определяемой лишь для некоторых дискретных аргументов. Получаемые алгоритмы называют разностными схемами. Анализ разностных схем позволяет выделить их специальные классы — однородные, устойчивые, помехоустойчивые, экономичные и др.

Выбор модели тем эффективнее, чем больше данных имеется о конечном решении задачи. В процессе прикидочных исследований осуществляется сравнение величин отдельных членов уравнений в изучаемом диапазоне изменения переменных и параметров задачи. Относительно малые слагаемые отбрасываются, нелинейные зависимости линеаризуются, случайные величины заменяются их математическими ожиданиями. Иногда оказывается достаточным даже грубое приближение (например, при решении задач оптимизации с пологими экстремумами). Однако в окрестности экстремума аппроксимация должна быть минимум квадратичной.

Как при аналитических выкладках, так и при численном решении должны проверяться:

- совпадение размерностей складываемых (вычитаемых) величин;
- соблюдение специфических для предметной области соотношений типа законов сохранения;
- осмысленность решения при устремлении параметров к предельным значениям (например, нулю, единице или бесконечности);
- факт задания и выполнение начальных и граничных условий;
- физический смысл промежуточных результатов;
- однозначность решения;
- устойчивость решения при малых вариациях параметров.

При *имитационном* моделировании реализующий модель алгоритм воспроизводит процесс функционирования системы во времени и в пространстве, причем имитируются составляющие процесс элементарные явления с сохранением его логической и временной структуры. Имитационное моделирование усугубляет недостатки численного, но зато

практически свободно от ограничений на класс решаемых задач. Обычная сфера его применения — дискретные случайные процессы (стационарные).

Весьма перспективно *комбинированное* моделирование, сочетающее (в умелых руках) достоинства всех рассмотренных подходов.

1.3. Требования к математическим моделям

Возможность исследования того или иного явления реального мира некоторым математическим методом определяется формальными свойствами построенной модели. Познание разделяется на два этапа: формирование математической модели и получение логических следствий из нее. Оба этапа требуют *осознания и четкой фиксации сделанных допущений и принятых ограничений*. Составление модели требует проникновения в природу вещей и достижения *внешнего* правдоподобия — достаточной степени адекватности математической модели реальному объекту по интересующим исследователя свойствам.

Под *внутренним* правдоподобием понимается ожидаемая точность реализации математических соотношений. Между ними должен быть разумный баланс: точность вычислений должна соответствовать точности исходных данных. Возможны и отклонения от этого принципа: к максимальному уровню внутреннего правдоподобия независимо от уровня внешнего стремятся, если:

- осуществляется проверка внешнего правдоподобия;
- речь идет о разработке нового метода исследований, который предполагается применить к широкому, не фиксированному заранее классу моделей.

Разумеется, искусственное понижение точности счета (например, счет в укороченной по отношению к формату машинного слова разрядной сетке, загрубление алгоритмов расчета стандартных математических функций) бессмысленно.

Второй этап требует умения найти и использовать нужный аппарат. На последний часто наталкивает рассмотрение частных случаев и результатов численных расчетов. Это прежде всего отбор влияющих факторов, а также предельное поведение параметров (например, в задачах теории надежности — быстрое восстановление, простейший поток отказов).

1.3.1. Адекватность

Математический эксперимент всегда проводится над абстрактной *моделью* явления, которая, однако, может оказаться недостаточно адекватной оригиналу.

Всякая модель не учитывает некоторые свойства оригинала и потому является его *абстракцией*. Приведем наиболее популярные примеры абстракций:

- Модель абсолютно твердого тела — уравнения движения тел в астрономии, баллистике, динамике полета (для ракеты с жидким топливом такая модель не подходит).
- Сплошная упругая среда в теории упругости.
- Ферма с идеальными шарнирами в строительной механике.
- Вязкая жидкость в гидромеханике.
- Идеальный газ — уравнения Эйлера для невязкого газа.
- Кинетическая теория газов — уравнения Больцмана.
- Экосистемы «хищник — жертва» — уравнения Вольтерра.
- Динамика боя — уравнения Ланчестера, Динера и др.

Чем более высок уровень обобщения, тем грубее модель и шире область интерпретации результатов моделирования.

Первым шагом в решении любой задачи обычно является построение линейной модели, которая позволяет пользоваться принципом *суперпозиции*: реакции на компоненты суммарного сигнала суммируются. Члены низкого порядка разложения функции в ряд Фурье несут обобщенную информацию о ней, высокого — более частную. При распознавании плоских фигур сопоставляют их моменты относительно координатных осей, начиная с моментов нулевого порядка (площади). Первые моменты определяют положение центра масс, вторые — моменты инерции и т. д. Сходство двух образов оценивается по совпадению нескольких начальных моментов. Аналогичный подход применяется и при сравнении распределений случайных величин.

Порой удается построить достаточно простую модель, которая передает основные тенденции явления в некотором диапазоне условий. Например, аэродинамика первоначально строилась в предположении, что воздух является идеальной несжимаемой жидкостью. Такая модель

среды действовала безотказно, пока скорости самолетов не превышали 200 км/час. Далее потребовалось учитывать сжимаемость воздуха. Принципиальные изменения пришлось внести при приближении к сверхзвуковым скоростям. Наконец, модель гиперзвуковой аэродинамики учитывает сложные физико-химические процессы в ближайшей окрестности летательного аппарата и на его поверхности.

Другой пример простой и эффективной модели — ньютоновская теория тяготения, созданная для описания солнечной системы (небесные тела рассматривались как материальные точки). Она позволила определить массы планет и их расстояния от Солнца и, более того, *предсказать* существование двух планет (Нептуна и Плутона) и открыть их. Однако ее пришлось уточнять для движения Меркурия (релятивистские эффекты) и низкоорбитальных спутников и ракет (нецентральность поля тяготения Земли и его локальные искажения, моделируемые системой дополнительных точечных масс).

При разработке новой системы для начала строится и оптимизируется стандартными методами простейшая модель, учитывающая лишь важнейшие факторы. При этом вычисление каждого значения функции оказывается менее трудоемким, упрощается структура линий уровней (постоянства значений) целевой функции. Затем модель постепенно усложняется за счет учета все новых и новых факторов и ставятся задачи оптимизации по все большему числу параметров. Решение каждой из них обычно оказывается хорошим начальным приближением для следующей. Усложнение заканчивается, когда ввод новых дополнительных факторов перестает заметно влиять на целевую функцию.

При последовательном решении подзадач можно уже после получения первых результатов обнаружить рассогласование математической модели с исходным процессом и внести необходимые коррективы. Есть точка зрения [10, с. 347], что крайне трудные для обсчета целевые функции с узкими и длинными ямами («оврагами») возникают при неудачном выборе математической модели явления.

Точная модель, как правило, имеет более узкую область применения, зато представляет в ней бóльшую практическую ценность. В частности, она может использоваться непосредственно в контуре управления производственным процессом. Сравниваемые модели должны быть сопоставимы — работать при одних и тех же внешних условиях.

Многомерные математические задачи обычно возникают из описания сложных процессов. Эти описания являются довольно грубыми, и при решении многомерных задач следует требовать меньшей точности.

Практическое использование модели возможно лишь после тщательного ее исследования и настройки, в процессе которых необходимо решить задачи проверки адекватности, идентификации параметров, оценки значимости параметров и структурного преобразования.

Адекватность модели устанавливается проверкой для нее основных законов предметной области типа законов сохранения и сопоставлением результатов моделирования частных вариантов с известными для этих вариантов решениями. Адекватность модели объекту всегда ограничена и зависит от цели моделирования. Рекомендуется выбирать модель минимальной сложности при заданной точности либо максимальной точности — при заданной сложности.

Адекватность модели следует рассматривать только по признакам, принятым в данном исследовании за основные. Выясняется, приводят ли принятые допущения к приемлемой потере точности или же из них вытекает качественное отличие модели от объекта. Чем выше *побочная* адекватность, тем шире область применения модели.

Задачей *идентификации* является определение значений рабочих параметров модели по наблюдениям над реальной системой. При этом тип модели предполагается известным. Эта задача обычно ставится в форме минимизации функционала отклонения траектории модели от траектории исследуемой системы. Для ее решения традиционно применяются методы наименьших квадратов и наибольшего правдоподобия.

Выявление *значимых параметров* (и пренебрежение остальными) позволяет уменьшить размерность пространства параметров. Оно базируется на определении коэффициентов чувствительности (регрессии) выходных показателей по отношению к входным и представляет особую ценность при решении задач оптимизации. Важным способом уменьшения размерности модели является рекомендуемый в теории подобия переход к обобщенным параметрам.

Принцип *баланса точности* требует соизмеримости погрешностей, вызываемых различными причинами: неполнотой модели, неточным заданием ее параметров, погрешностью выбранного численного метода, счетом в ограниченной разрядной сетке (это не означает необходимости искусственного загрубления уже достигнутых показателей).

1.3.2. Корректность

Задача называется поставленной корректно, если для любых значений исходных данных из некоторого класса ее решение существует, един-

ственно и устойчиво по исходным данным. Последнее означает малость изменения результатов счета при незначительных изменениях входных данных (разумеется, в применении к устойчивым реальным процессам).

Неустойчивые алгоритмы и плохо обусловленные задачи встречаются почти во всех областях вычислительной математики. Во многих учебниках цитируется классический пример Уилкинсона — расчет корней многочлена

$$P_{20}(x) = (x-1)(x-2)\dots(x-20) = x^{20} - 210x^{19} + \dots = 0.$$

При увеличении второго коэффициента на $2^{-23} \approx 10^{-7}$ половина корней становится комплексными [99]. Неустойчивыми моделями являются задачи линейной алгебры с плохо обусловленными матрицами, некоторые виды интегральных уравнений и др.

При решении плохо обусловленных задач выбор метода не поможет. Такие модели должны быть подвергнуты *регуляризации*: замене некорректной задачи на содержащую параметр корректную, которая при устремлении параметра к нулю переходит в исходную [97]; замене точного решения системы уравнений на приближенное в смысле минимума среднеквадратической невязки левой и правой частей; параметризации искомой функции [99] и т. п. Ясно, что практическую ценность могут иметь только устойчивые модели.

1.4. Дискретная модель

Решение задачи редко удается выразить конечной формулой. Основным инструментом являются *численные методы*, позволяющие свести решение задачи к выполнению конечного числа арифметических действий. Незнание методов может привести к постановке математической задачи, не имеющей отношения к реальности.

Необходимость получения результата за конечное число шагов требует использования быстро сходящихся методов. Сходимость означает приближение промежуточного решения к искомому результату (при увеличении числа итераций, уменьшении шага). Область сходимости стандартных методов обычно мала. Поэтому приходится применять естественные аналогии, правдоподобные рассуждения, поиск начальных приближений на грубой сетке и т. п. Все это требует определенного уровня понимания используемых методов независимо от того, какую часть ма-

тематической модели вы программируете сами, а какую будете обсчитывать с помощью стандартных подпрограмм или математических пакетов (см. схему рис. 1.1). *Наилучшим критерием понимания метода является способность его запрограммировать.*

При решении прикладных задач приходится заниматься теоретической и экспериментальной доводкой стандартных методов. Например, для систем нелинейных уравнений и оптимизации уже при умеренной размерности задачи нет универсальных алгоритмов. В то же время известны эффективные алгоритмы для задач, имеющих специфическую внутреннюю структуру.

Развитие вычислительной техники и теории численных методов приводит к непрерывному пересмотру и некоторому сужению совокупности реально применяемых методов. Считается, что эффект, достигаемый за счет совершенствования численных методов, по порядку сравним с эффектом повышения производительности ЭВМ. По многим причинам весьма актуальна разработка численных методов и программных средств для многопроцессорных ЭВМ.

Глава 2.
Основы Фортрана

Реализация дискретной математической модели требует программирования модели в целом или ее частей на ЭВМ. Фортран был первым языком программирования для научно-технических расчетов и остается лучшим инструментом для них (см. развернутую аргументацию по этому поводу в [85]).

Приводимое ниже описание средств современного Фортрана (Фортран 90 и его последующие модификации, далее в совокупности именуемые Ф90) дает лишь минимальный объем сведений, необходимый для понимания приводимых в последующих главах программных реализаций методов вычислительной математики. Оно не может рассматриваться как *вводный* курс программирования. Для самостоятельной практической работы мы настоятельно рекомендуем [8, 85] и советуем отличать описания языка от учебников *программирования* на нем.

2.1. Алфавит и простейшие конструкции

Ключевые слова Фортрана распознаются по контексту. Объекты программы различаются *именами*, которые начинаются с *буквы* и могут включать буквы, знак $, цифры и символ подчеркивания. Предпочтительны *значимые* имена, облегчающие сопоставление программы с математической постановкой задачи.

В строках и комментариях допустимы кириллические символы. Заглавные и малые буквы различаются только в строках. Для большей наглядности включенные в поясняющий текст конструкции Фортрана набраны заглавными буквами.

Пробелы в программе значимы только внутри символьных строк.

Все метки в Ф90 — целые числа без знака (до 5 символов). Метки вида 0010 и 10 считаются одинаковыми. В пределах программной единицы метка повторяться не должна.

Признаком действительного числа является наличие точки в мантиссе и/или разделитель <<e>> (для двойной точности <<d>>) между мантиссой и показательной частью.

2.2. Оформление программы

Программа допускает запись в свободном формате. Мы рекомендуем читателю размещать числовые метки с 1-й позиции, а операторный текст — с 7-й, показывая отступами вложенность конструкций языка и парность ключевых слов. *Свободный формат* разрешается заданием специального режима при вызове компилятора. Строка может быть начата с любой колонки. В одной строке можно разместить несколько операторов, разделяемых точкой с запятой. Признаком наличия у оператора продолжения является завершающий строку амперсанд (&).

Признаком полнострочного комментария служит двойная кавычка в первой позиции. Чистая строка рассматривается как комментарий и переходит в листинг. Комментарий к оператору может быть размещен справа от него в той же строке и отделяется восклицательным знаком.

2.3. Типы данных

Стандартным в Ф90 считается четырехбайтовое представление переменных типа `INTEGER`, `LOGICAL`, `REAL`, `COMPLEX` (в последнем случае отводятся по 4 байта под вещественную и мнимую части). При объявлении таких переменных их длину в байтах можно не указывать. Стандартное представление обеспечивает наиболее быстродействующую реализацию. Кроме того, допустимы типы

```
REAL*8     (=DOUBLE PRECISION)
COMPLEX*16 (=DOUBLE COMPLEX)
```

Для символьных строк всегда указывается требуемое число байтов: `CHARACTER*N`.

Необъявленным переменным по умолчанию назначается стандартный тип в соответствии с *первой буквой имени* (`INTEGER` для `I-N`, `REAL` в остальных случаях).

2.4. Выражения

В отличие от Паскаля, в Фортране имеется операция ** возведения в степень. При выполнении арифметических операций аварийно завершаются: деление на нуль, переполнение порядка, логарифмирование неположительного числа, попытка возведения отрицательного числа в вещественную степень.

При делении целого числа на целое дробная часть отбрасывается. Таким образом, вычисление $1/2 + 1/2$ даст нуль. Для предупреждения такого эффекта хотя бы один из операндов деления должен быть *формально* вещественным.

При выполнении арифметических действий с операндами различных типов производится автоматическое преобразование операнда менее общего типа (в частности, меньшей длины) к операнду более общего типа. Тип значения выражения устанавливается по типу последней выполненной операции.

Для записи отношений используется стандартная математическая символика (например, « $<$ » вместо традиционного «.LT.»).

При выполнении оператора присваивания значение выражения преобразуется к типу левой части. Присваивание вещественного значения целой переменной приводит к *отбрасыванию* дробной части.

2.5. Массивы и действия над ними

2.5.1. Основные определения

Массивом называется упорядоченное множество *однотипных* значений (проекции вектора, таблица результатов эксперимента, матрица преобразования координат). Элементы массивов обычно обрабатываются некоторым стандартным образом. Ссылка на элемент массива состоит из общего имени массива и (в круглых скобках) набора индексов, характеризующих положение в нем избранного элемента: `A(I,J)` означает элемент матрицы `A` на пересечении `I`-й строки и `J`-го столбца. Тем самым отпадает надобность в сотнях различных имен для родственных объектов программы. Индексы сами могут быть элементами массивов. Два массива считаются равными, если равны все их элементы, *стоящие в одинаковых позициях*.

Массивы характеризуются типом значений их элементов, *рангом* — числом измерений (не более семи)[1] и граничными парами — диапазоном индексов по каждому измерению. Границы каждого диапазона разделяются двоеточием; нижняя граница +1 при описании может быть опущена (вместе с двоеточием). Приведем несколько примеров описания массивов:

```
real        a,b,c
integer     k
dimension   a(0:4), b(3,7), c(3,7), k(6)
```

Второй вариант того же описания указывает для каждого объекта полный набор атрибутов:

```
integer, parameter       :: n=3
real,    dimension       :: a(0:n+1)
real,    dimension (3,7) :: b,c
integer, dimension       :: k(6)
```

Здесь для задания границы массива `A` применено *константное* выражение, которое вычисляется на этапе компиляции. Напомним, что граничные пары могут быть указаны и непосредственно в описании типа.

[1] Не путать с рангом матрицы!

Из рассмотренных примеров следует, что атрибуты могут задаваться как порознь, так и вместе, причем объекты с общими атрибутами можно группировать в единый список. Второй вариант как более устойчивый к ошибкам считается предпочтительным.

Протяженность массива по некоторому измерению называется его *экстентом*, а упорядоченный полный набор экстентов — *формой* массива. Массивы одинаковой формы считаются *конформными*. Подчеркнем, что *для конформности массивов совпадения граничных пар не требуется*. Скаляр считается конформным с любым массивом. Произведение экстентов задает *размер* массива. Скаляр *формально* не является массивом нулевого ранга с единичным экстентом и потому не может подставляться вместо такового.

Элементы массивов располагаются в памяти таким образом, что первым меняется старший (самый левый) индекс. В частности, матрицы размещаются *по столбцам*. Это обстоятельство следует учитывать при организации ввода и вывода многомерных массивов.

2.5.2. Вырезки и сечения массивов

Для ссылки на элемент массива задается индексированная переменная; на массив в целом ссылаются по его имени. В Ф90 есть возможность непосредственно работать с частями массива — вырезками и сечениями. *Вырезка* из массива представляет собой подмассив вида

`<имя_массива>(<диапазон>,...,<диапазон>)`

Элементы вырезки из массива могут быть взяты с шагом, отличным от единицы. В этом случае вырезка по соответствующему измерению задается уже не граничной парой, а *триплетом*. Так, `A(I,2:8:2)` — это четные элементы `I`-й строки матрицы `A`. Шаг (третья позиция триплета) может быть и отрицательным; тогда второй элемент триплета должен быть меньше первого, и элементы исходного массива будут выбираться в обратной последовательности. Опущенные нижние или верхние границы вырезки заимствуются у массива в целом. Если по какому-то измерению опущены обе границы вырезки (двоеточие должно быть сохранено), то говорят о *сечении* массива: `A(I,:)` означает `I`-ю строку упомянутой матрицы `A`. Понятие конформности массивов, естественно, переносится и на вырезки.

2.5.3. Задание массивов

Для означивания массивов существуют следующие средства:

- описание начальных значений;
- оператор ввода;
- оператор присваивания.

Приведем примеры использования этих средств (кроме ввода, обсуждаемого отдельно). Переменные и массивы можно инициализировать прямо в описании типа:

```
real                   :: z=0.0
integer, dimension(3)  :: m /2,5,11/
integer, dimension(4)  :: k=(/1,3,5,7/)
```

(запись в правой части последнего описания называется *конструктором массива*). В тех же целях можно использовать оператор

data <список_объектов>/<список_значений>/

где <список_объектов> включает переменные и неявные циклы. При означивании происходит поэлементное сопоставление обоих списков с учетом структуры объектов (массивов) и повторителей вида 4* в списке значений:

```
integer i,j
integer, parameter :: l=4, m=l*(l+1)/2
real q(l,l)
data ((q(i,j), j=1,i), i=1,l) /m*1.0/
```

Здесь заполняется вещественными единицами нижняя треугольная часть матрицы Q размером L*L. Обратите внимание на использование в этом примере именованных констант и константных выражений!

Отметим дополнительно, что:

- циклический список заключается в скобки;
- циклический список может быть вложенным;
- во всех циклах можно дополнительно указать после второй границы шаг приращения параметра (по умолчанию +1);

- каждая переменная или часть ее в программе могут инициализироваться только один раз;
- при различии в типах элементов массива и задаваемых значений происходит автоматическое преобразование последних (как в операторах присваивания).

Пар списков (как и самих операторов DATA) может быть несколько.

2.5.4. Действия над массивами в целом

Над конформными массивами в Ф90 допускаются агрегатные операции сложения, вычитания и присваивания, выполняемые покомпонентно. Скалярная переменная считается конформной любому массиву, что позволяет одним оператором:

- присвоить всем компонентам массива скалярное значение (например, нулевое);
- умножать элементы массива на скаляр или добавлять его к ним.

К массиву в целом может покомпонентно применяться любая функция скалярного аргумента. Во фрагменте

```
real p(5), q(5), r(7)
p=3.0
r=4.67
q=p+0.8*r(3:7)
p=exp(p)
```

первые два присваивания разрешены из-за конформности скаляра и произвольного массива; третье допустимо благодаря выбранной протяженности вырезки и поэлементному характеру сложения векторов; четвертое определяет поэлементное вычисление показательной функции.

2.5.5. Выборочные действия

Конструкция WHERE позволяет запрограммировать действия с некоторыми элементами массива, отбираемыми в соответствии с логическим массивом той же формы (*маской*). Так, во фрагменте

```
where (a<0) a=0
```

все отрицательные элементы матрицы A будут заменены нулями. Заметим, что все компоненты этой и родственных ей конструкций являются матричными. Наиболее общим является оператор

```
where (<ЛВ-массив>)
      <присваивания_массивов>
elsewhere
      <присваивания_массивов>
end where
```

(блок ELSEWHERE не обязателен). Операторов присваивания в каждом блоке может быть несколько, но все они должны относиться к массивам одинаковой формы, совпадающей с формой маски. Логическое выражение-массив может порождаться покомпонентным выполнением операции отношения или произвольно формируемой логической маской:

```
real x(100)
integer sparse (100)
logical, dimension(100) :: mask
.............................
mask(1:100:2)=.true.
where(mask)
   x=x+1.0
   sparse=1
end where
```

В этом примере каждый нечетный элемент из X будет увеличен на единицу, а из SPARSE получит единичное значение.

Сечения управляют выборкой элементов массива по координатам, тогда как оператор WHERE — по условию. Оператор FORALL (Ф95) позволяет объединять оба этих способа. Его синтаксис:

```
forall (<индексное_выражение> [,<скалярная_логическая_маска>])
       <оператор_присваивания>
```

Присваивание выполняется для всех указанных в левой части комбинаций индексов, которым отвечает истинное значение маски. Если, например, все ненулевые элементы массива в диапазоне индексов (2:6,3:N+1) нужно заменить на им обратные, следует записать:

```
forall (i=2:6, j=3:n+1, R(i,j)\=0) R(i,j)=1.0/R(i,j)
```

При необходимости в нескольких операторах присваивания их совокупность завершается END FORALL. Все такие операторы выполняются строго последовательно.

Еще один пример программы на FORALL заполняет единицами нижнюю левую часть матрицы (включая главную диагональ):

```
   program tforall
   integer, parameter: n=4
   integer i,j
   real x(n,n)
   x=0
   forall (i=1,n, j=1,n, i>=j) x(i,j)=1.0
   write (*,3) ((x(i,j), j=1,n), i=1,n)
 3 format (1x,<n>e14.5)
   end
```

Этот пример наряду с применением FORALL иллюстрирует *построчный* вывод результирующей матрицы и применение символьного указателя кратности в спецификации формата.

2.5.6. Массивы с переменными границами

Ф90 разрешает (но не в главной программе) *динамические* массивы, позволяющие выделить память в нужный момент выполнения программы, по фактической потребности и под данные *заявленной структуры*. Каждый динамический массив должен быть объявлен в описании типа и/или структуры с атрибутом ALLOCATABLE. В этом объявлении указываются только тип значений и количество измерений (последнее — посредством двоеточий, каждое из которых представляет одну граничную пару). Для краткости будем именовать такие массивы А-массивами. А-массивы не конформны никаким иным (в том числе другим массивам с этим свойством) и потому не допускают в паре с ними агрегатных операций. Пары массив-скаляр и скалярная функция от массива исключением не являются.

Реальное выделение памяти для каждого А-массива происходит на шаге выполнения по оператору ALLOCATE, указывающему фактические граничные пары. Последние могут быть заданы арифметическими выражениями, все переменные в составе которых к моменту выделения памяти должны быть означены.

Освобождение памяти, затребованной под А-массивы, производит-

ся оператором DEALLOCATE с заключенным в скобки списком освобождаемых объектов. Применение DEALLOCATE к объектам, память под которые в текущий момент не выделена, приводит к непредсказуемым последствиям.

Приведем схему программы с динамическими массивами:

```
real        eff
dimension   eff [allocatable](:,:)
.............................
n=k
m=k+j
.............................
allocate    (eff(n,m))
.............................
deallocate  (eff)
.............................
```

Квадратные скобки, обрамляющие ALLOCATABLE, синтаксически обязательны.

2.6. Встроенные функции

В классическом Фортране встроенные процедуры, реализующие одну и ту же стандартную функцию, в зависимости от типа аргумента имели различные частные имена. Например, натуральный логарифм обозначался как ALOG — для REAL*4, DLOG — для REAL*8, CLOG — для COMPLEX*8. В Ф90 можно ссылаться на общее (родовое) имя функции. Для логарифма это LOG, а для остальных математических функций — частные имена, соответствующие формату REAL*4. Конкретная функция семейства выбирается компилятором в зависимости от типа аргументов.

Встроенную функцию в роли фактического параметра процедуры следует подставлять ее *частным* именем. Это частное имя должно быть специфицировано в вызывающей процедуре как INTRINSIC.

2.7. Разветвления

К средствам разветвления стандартного Фортрана в Ф90 добавлен «блочный IF»:

```
IF (<ЛВ1>) THEN
    <Блок_1>
ELSE
    <Блок_2>
END IF
```

Здесь ЛВ — логические выражения, а блоки — последовательность операторов. В отличие от языка Си, блок *не нужно* заканчивать передачей управления преемнику полного условного оператора. IF-группы могут быть вложенными.

Возможен и сокращенный условный оператор, например:

```
IF (N.LT.0) THEN
    X=-N
    Z=2*Y
END IF
```

В случае одного подоператора условный оператор записывается в форме

```
IF (N.LT.0) X=-N
```

Переходы извне к подоператорам блочного IF и цикла запрещаются.

2.8. Циклы

Простейшие циклы Фортрана — с шагом:

```
DO  <параметр> = <начало>, <конец>
    <тело_цикла>
END DO
```

Если шаг отличен от 1, он указывается дополнительно третьим аргументом заголовка. В Ф90 добавлены следующие возможности:

- цикл с логическим условием — заголовок вида `DO WHILE (<ЛВ>)`;
- оператор `CYCLE`, обеспечивающий передачу управления заголовку цикла;

- оператор EXIT выхода из цикла в непосредственно объемлющую конструкцию.

Рассмотрим иллюстрирующую их программу вычисления суммы

$$1 + x + x^2/2! + \ldots + x^{n-1}/(n-1)! + x^n/n! \sum_{i=n}^{\infty} (x/n)^{i-n}$$

с учетом слагаемых, превышающих 0.005.

```
program sum
   data i,x,s,b,c,eps /1,  0.5, 1.0, 1.0, 1.0, 0.005/
   do while (i.le.100) !! кратность назначена с запасом
      b=b*x/c
      if (abs(b).le.eps) then
         exit
      else
         s=s+b
      end if
      i=i+1
      if (i.lt.n) then
         c=c+1
      else
         cycle
      end if
   end do
   write (6,'(1p,e16.6)') s
end
```

2.9. Программа и ее компоненты

Программа на Ф90 представляет собой определенным образом оформленную совокупность операторов. Она может быть монолитной или составной.

Минимальный состав законченной программы — только *головная программа*. Однако любая достаточно сложная проблема требует ее рассмотрения на нескольких уровнях детальности и может быть разделена на относительно самостоятельные подзадачи, что определяет иерархическое построение программы из нескольких субпрограмм (процедур).

Обычно некоторые из подзадач относятся к стандартному арсеналу математика или типичны для рассматриваемого круга приложений — следовательно, уже были или должны быть запрограммированы для многократного применения. Использование процедур делает программу в целом компактной и обозримой; уменьшает трудоемкость разработки и в особенности отладки; позволяет распараллелить разработку между несколькими исполнителями; облегчает внесение исправлений в сложный алгоритм с однотипными фрагментами.

Процедуры могут быть внутренними, внешними и модульными. Описание *внутренней* процедуры непосредственно включается в текст охватывающей программной единицы (носителя) после `contains` и не может содержать вложенных процедур. *Внешняя* процедура существует автономно. Имя такой процедуры не обязано совпадать с именем содержащего ее файла.

Программные единицы бывают следующих видов: главная программа, внешняя процедура, модуль и блок данных. Каждая из них:

- физически отделена от других;
- начинается оператором-заголовком, содержащим специфическое для данной программной единицы ключевое слово: `PROGRAM`, `SUBROUTINE`, `FUNCTION`, `MODULE`, `BLOCK DATA`;
- заканчивается предложением `END` с повторением (иногда не обязательным, но рекомендуемым) вышеупомянутого ключевого слова и собственного имени программной единицы;
- обрабатывается компилятором отдельно от остальных.

Выполняемая программа состоит из головной программы и произвольного числа остальных программных единиц. Частью головной программы могут быть внутренние процедуры. Обобщенный вариант структуры программы показан на рис. 2.1.

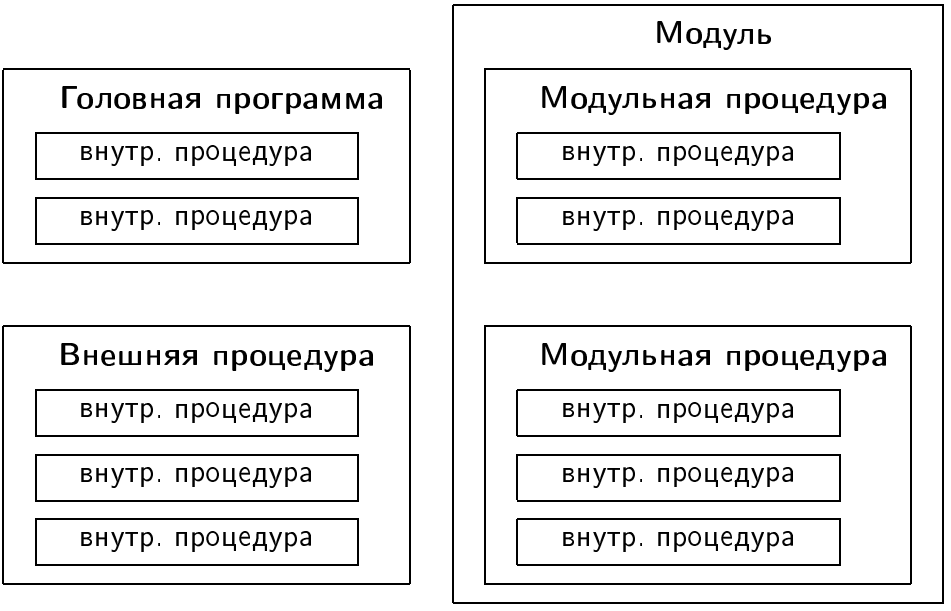

Рис. 2.1. Структура программы

Вложенные процедуры любого вида имеют доступ ко всем объектам своего носителя.

Модуль может содержать несколько модульных процедур, при необходимости включающих в себя внутренние. Кроме того, в нем могут содержаться описания данных, определения производных типов, интерфейсные блоки. Обычно модули создают специализированную среду и инструментарий для решения некоторого класса задач (работа с упакованными матрицами и матрицами специального вида, интервальная арифметика, алгебра нечетких множеств).

Блоки данных содержат только описательные операторы. Они используются для инициализации переменных в именованных общих блоках.

Головная программа имеет следующий вид:

```
[program <имя_программы>]
   [<операторы_описания>]
   [<исполняемые_операторы>]
[contains
      <внутренние_процедуры>]
 end [program [<имя_программы>]]
```

Здесь в квадратные скобки заключены необязательные элементы. Оператор END ограничивает текст программной компоненты и завершает ее выполнение. Прекращение выполнения программы в произвольном месте задается оператором STOP.

Внешняя процедура отличается от головной программы только заголовком и ключевым словом в завершающей строке. Она приводится в действие из вызывающей программной единицы оператором CALL или указателем функции (для функций). Имена внешних процедур и формальные параметры-процедуры должны быть специфицированы в вызывающей процедуре как EXTERNAL либо INTRINSIC.

Внутренние процедуры выглядят так же, как модульные, но не могут иметь вложенных в них процедур.

После загрузки всего программного комплекса в оперативную память управление передается головной программе.

2.10. Подпрограммы

Заголовок процедуры-подпрограммы имеет вид:

```
subroutine <имя_процедуры>(<список_формальных_параметров>)
```

Формальные параметры вводят обозначения для описания стандартных действий, выполняемых процедурой. *Вызов* подпрограммы производится оператором

```
call <имя_подпрограммы>(<список_аргументов>)
```

и отрабатывается в первом приближении как ее тело (описание) с заменой формальных параметров фактическими — аргументами. Хорошее представление о происходящем дает аналогия процедуры с пьесой. Спецификации параметров можно уподобить аннотированному списку действующих лиц, тело процедуры — тексту пьесы, вызов процедуры — исполнению пьесы конкретным составом актеров. Очевидно требование соответствия аргументов параметрам по количеству, типу и смыслу. Совпадение имен фактических и формальных параметров не обязательно, но допустимо.

Поскольку характер формальных параметров определяет их представление и обработку на машинном уровне, они должны быть специфицированы (объявлены) внутри процедуры. Кроме того, для них может (иногда — должен) быть установлен *вид связи* — атрибут INTENT с

указанием в скобках одного из вариантов `IN, OUT, INOUT` (входной, выходной, переопределяемый). Аргумент, замещающий входной параметр, может быть выражением; тогда его значение вычисляется и фиксируется при входе в процедуру. Выходному и переопределяемому параметрам может соответствовать только переменная (в частности, массив или его сечение).

Пример. Пусть необходимо вычислить

$$y = \frac{f(x) + 2f(2x)}{x - f^2(x + 0.5)},$$

где $f(u) = \sin u - e^{-u}$ и x=0.2. Эту задачу решает программа

```
program  pp
   real y,y1,y2,y3, x/0.2/
   call p(x,y1); call p(2.0*x,y2); call p(x+0.5,y3)
   y=(y1+2.0*y2)/(x-y3**2)
   print *, 'y = ',y
end program pp

subroutine p(a,z)
   real a,z
   z=sin(a)-exp(-a)
end subroutine p
```

2.11. Функции

Обращение к *функции* может осуществляться непосредственно из выражения, и возвращаемое значение используется в том же выражении. Это значение является единственным результатом работы функции (возможно, структурированным, т. е. массивом). Оно сопоставляется имени функции; поэтому в заголовке функции отсутствует ответный параметр, а в спецификации для имени функции указываются атрибуты результата:

```
integer function sumints(x)
```

(при отсутствии такого описания в заголовке или внутренних операторах объявления атрибутов тип результата будет назначен по умолчанию в соответствии с первой буквой имени функции). Вычисление функции должно заканчиваться присваиванием ее имени ответного значения, после которого следует завершающий `end function[<имя>]`.

Имя функции должно быть объявлено в *вызывающей* процедуре с атрибутами возвращаемого значения.

При использовании внешней процедуры-функции программа поставленной на с. 48 задачи может иметь вид:

```
program pf
   real f,y,x/.2/
   y=(f(x)+2*f(2*x))/(x-f(x+.5)**2)
   print * ,' y = ',y
end program pf

function f(a)
   real a
   f=sin(a)-exp(-a)
end function f
```

2.12. Расположение операторов

Приведем сводную таблицу относительного расположения операторов программной единицы (табл. 2.1).

Таблица 2.1. Структура программной единицы

<table>
<tr><td colspan="3">Операторы PROGRAM, FUNCTION,
SUBROUTINE или MODULE</td></tr>
<tr><td colspan="3">Операторы USE</td></tr>
<tr><td rowspan="4">Операторы
FORMAT</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>Операторы
PARAMETER</td><td>Операторы IMPLICIT</td></tr>
<tr><td>Операторы
PARAMETER
и DATA</td><td>Определения производных типов,
интерфейсные блоки,
операторы описания типа и
другие операторы описания</td></tr>
<tr><td colspan="2">Исполняемые операторы</td></tr>
<tr><td colspan="3">Оператор CONTAINS</td></tr>
<tr><td colspan="3">Внутренние или модульные процедуры</td></tr>
<tr><td colspan="3">Оператор END</td></tr>
</table>

Относительный порядок конструкций из вертикальных блоков таблицы, имеющих общее горизонтальное сечение, произволен. Все операторы описания данных должны предшествовать выполняемым операторам. Операторы, разделенные горизонтальными линиями, перемежаться не могут.

2.13. Области видимости меток и имен

Видимыми называются объекты, на которые можно ссылаться из данного места программы. Имена программных компонент и внешних процедур глобальны, т. е. доступны из любого места программы. Каждое из них должно отличаться от всех остальных, а также от имен локальных объектов. Каждое имя видимо из своего непосредственного носителя, исключая те вложенные конструкции (процедуры, интерфейсные блоки и декларации производных типов), где оно переопределяется. Аналогично обстоит дело с использованием числовых меток. Любой оператор перехода GOTO возможен только на видимую из данной точки метку. Поскольку переходы внутрь блоков, минующие их начало, потенциально опасны, компилятор при обнаружении таких ситуаций выдает предупреждения.

2.14. Внутренние процедуры

Описания внутренних процедур (как подпрограмм, так и функций) размещаются в конце главной программы, внешней или модульной процедуры после ключевого слова CONTAINS. Обычно так оформляют частные алгоритмы, особенно тесно связанные с конкретной задачей. Внутренняя процедура работает в среде своего носителя и может быть вызвана только из него или другой внутренней процедуры того же носителя. Она может ссылаться на локальные имена объектов своего носителя, что позволяет передавать ей информацию помимо параметров, уменьшает требуемое число таковых и упрощает ее вызов. В предельном случае возможно использование внутренних процедур без параметров.

При использовании внутренней процедуры-функции обсуждавшаяся выше задача оформляется следующим образом:

```
program pf
   real y,x/0.2/
   y=(f(x)+2.0*f(2.0*x))/(x-f(x+0.5)**2)
   print *, 'y = ',y
contains
   function f(a)
      real a
      f=sin(a)-exp(-a)
```

```
    end function f
  end program pf
```

Разумеется, она может быть решена и с помощью внутренней подпрограммы.

Внутренние процедуры не могут содержать вложенных в них других внутренних процедур и быть фактическими параметрами обращений к другим процедурам. Это не позволяет, например, применить стандартную процедуру численного интегрирования для внутренней подинтегральной функции, часть объектов которой определена в ее носителе.

2.15. Интерфейс процедур

Интерфейсом процедуры называется минимально необходимая для ее вызова совокупность сведений о ней: подпрограмма это или функция, каковы структура списка параметров и атрибуты последних. Будучи сообщенной компилятору, она позволяет сопоставлять типы фактических и формальных параметров на этапе компиляции. Это уменьшает риск ошибок фазы выполнения.

Интерфейс внутренней процедуры непосредственно доступен компилятору и потому считается явным. Для успешной компиляции *внешних* процедур их имена должны быть перечислены в операторе EXTERNAL вызывающей программной единицы перед первым исполняемым оператором. Интерфейс для каждой из них должен быть задан явно в блоке INTERFACE типа

```
interface to subroutine mfact(j,p,n,mg,nn,mf)
integer                 j,n,nn,mg
double precision        p,mf
dimension               p(0:j),mf(nn)
end
```

Все строки блоков INTERFACE должны начинаться с *первой* позиции. Буквенные обозначения параметров в INTERFACE и порядок их перечисления не обязаны совпадать со списком формальных параметров в описании процедуры (допустима иная группировка параметров и атрибутов).

Для автоматической вставки блок INTERFACE должен находиться в том же файле, что и вызывающая процедура. Удобна его вставка препроцессором по режиму INCLUDE из предварительно заготовленного набора интерфейсов к библиотечным процедурам.

Использование блока интерфейса обязательно, если внешняя процедура:

- вызывается с ключевым или необязательным аргументом;
- формирует ответный массив либо строку переменной длины;
- является формальным или фактическим параметром;
- вызывается по родовому имени.

Интерфейс должен быть помещен до первого вызова соответствующей процедуры.

Блок INTERFACE лишь *контролирует* соответствие типов фактических параметров типам формальных, но не проводит их принудительное согласование. Однако нужно иметь в виду, что при отсутствии контроля и рассогласовании типов результат вызова процедуры непредсказуем.

2.16. Специальные случаи параметров процедур

2.16.1. Массивы как параметры

Этот раздел мы начнем с примера заголовка процедуры, специфицирующего разные виды формальных параметров-массивов:

```
subroutine up(n,p,r,s,t)
   real, dimension (1:15,5,10) :: p ! с явными границами
   real, dimension (:,5:)      :: s ! подразумеваемой формы
   real, dimension (n,n+2)     :: t ! автоматический
```

Мы рекомендуем подобную запись заголовка процедуры (с дополнительным указанием в комментариях назначения процедуры, идеи или ссылки на описание алгоритма, *смысла* формальных параметров) для любого серьезного программирования.

Автоматическим называется массив с переменными границами, которые прямо или косвенно выражаются через параметры: например, DIMENSION T(SIZE(A)). Он может быть как формальным параметром процедуры, так и локальным объектом ее.

Массив *подразумеваемой формы* заимствует эту форму (т. е. набор экстентов) у связанного с ним фактического параметра с возможным сдвигом границ индексов. В вышеприведенном примере у массива S нижние границы индексов будут 1 и 5 соответственно независимо от нижних границ аргумента. Если, например, фактический массив W описан как

```
real, dimension(0:10, 0:12) :: w
```

а формальный FW — как

```
real, dimension(:,:) :: fw
```

т. е. с единичными (по умолчанию) нижними границами, то элементу W(I,J) будет сопоставлен FW(I+1,J+1).

Число параметров можно уменьшить, если границы фактических аргументов-массивов определять обращением к встроенной функции

```
size(<имя_параметра_массива>,<dim>)
```

где <DIM> — номер интересующего нас измерения. В результате можно записать процедуру обращения матрицы или решения системы линейных уравнений, указав лишь имя матрицы.

Для процедур с параметрами-массивами обсуждаемого вида обязательно наличие в тексте вызывающей процедуры блока интерфейса с вызываемой процедурой.

Чтобы структура параметра перенималась у аргумента *полностью*, спецификация параметра DIMENSION по каждому измерению должна включать в себя нижнюю границу.

Если результат функции в Ф90 — *массив*, то в ее заголовке должна присутствовать спецификация (RESULT <имя>), а для массива в спецификации параметров следует указать явные границы. Ответное значение получает массив, специфицированный в RESULT.

2.16.2. Процедуры как параметры

Параметром подпрограммы (функции) может быть другая подпрограмма или функция. Подобные параметры необходимы в стандартных *функциональных* процедурах: аппроксимации, интегрирования и дифференцирования произвольных функций, решения уравнений и т. п. Атрибут INTENT для такого параметра не нужен, а блок INTERFACE в вызывающей процедуре — обязателен.

Передаваемая функция (фактический параметр) должна быть внешней или модульной. Для члена *семейства* процедур нельзя указывать родовое имя. Встроенные процедуры, используемые как параметры, должны специфицироваться с атрибутом INTRINSIC. Внутренние процедуры в обсуждаемой роли выступать не могут.

Отмеченные ограничения относятся к случаю, когда процедура передается как *правило* для получения значения. *Готовые* значения (статические формальные параметры) автоматически вычисляются перед входом в процедуру (однократно).

2.17. Рекурсивные процедуры

По умолчанию процедура не может вызвать сама себя ни прямо, ни косвенным образом. Однако рекурсия иногда желательна для проблем, которые могут быть решены как серия задач меньшего размера (сортировки, вычисление кратного интеграла). В последнем случае внешний интеграл использует в качестве подынтегральной функции внутренний, и оба могут быть вычислены повторным обращением к одной и той же процедуре численного интегрирования. Заголовок такой процедуры должен включать в себя префикс RECURSIVE.

Если функция вызывает себя непосредственно, то для ее результата необходимо имя, отличное от имени функции. Тогда ответное значение присваивается выбранному имени, и оно же указывается в заголовке процедуры после списка параметров в форме RESULT(<имя>).

2.18. Ввод-вывод

2.18.1. Структура операторов ввода-вывода

Операторы ввода-вывода в стандартном Фортране имеют вид:

$$\left.\begin{matrix}\texttt{READ}\\ \texttt{WRITE}\end{matrix}\right\} \quad \texttt{u} \quad \texttt{f, <список\_ввода-вывода>}$$

Число U — это номер устройства (файла), участвующего в обмене. На персональных ЭВМ стандартными номерами внешних устройств явля-

ются «5» для клавиатуры и «6» — для экрана. Буква F представляет метку оператора FORMAT. Бесформатный вывод (точнее, вывод в формате, соответствующем описанию переменной) производится оператором PRINT.

2.18.2. Форматы ввода-вывода

Наиболее употребительны форматы:

nX — n пробелов (для печати с новой строки список форматов должен начинаться с хотя бы одного пробела);

A — текстовый (система сама подсчитывает число знаков в строке);

In — целое число в поле из n позиций;

Fn1.n2 — число с фиксированной точкой в поле из $n1$ позиций с $n2$ знаками после точки;

En1.n2 — число с плавающей точкой в поле из $n1$ позиций с $n2$ знаками мантиссы после точки (рекомендуется $n1 = n2 + 9$).

Формат должен быть согласован с типом выводимого объекта. Если поле недостаточно для размещения объекта вывода, вместо этого объекта будут напечатаны звездочки.

2.18.3. Список ввода-вывода

Для вывода скаляра или массива в порядке хранения его элементов в памяти достаточно указать в списке вывода имя нужного объекта. При выводе частичном или в иной последовательности применяются *циклические списки*:

```
      write (*,100) 'Эталон','Свертка'
  100 format(7x,a,8x,a)
      write (*,200) (e(j),f(j), j=1,6)
  200 format(1x,2e14.5)
```

Если записать:

```
      WRITE (*,200)  E,F
```

то сначала будут выведены все элементы E, а затем — все элементы F. Аналогичный прием приходится использовать и для вывода матрицы по строкам (в Фортране матрицы хранятся *по столбцам*). В этом случае список вывода будет вложенным.

При отработке вывода элементы его списка (с учетом возможной агрегатной структуры их) последовательно сопоставляются со значимыми элементами списка форматов — с учетом повторителей. При исчерпании *списка форматов* последний просматривается с начала.

Глава 3. Эффективное программирование

3.1. Факторы эффективности

Вопрос об эффективности программирования следует рассматривать комплексно и в определенном контексте. В данной книге справедливо ограничиться проблемой программирования вычислительных задач. Прежде всего перечислим возможные показатели эффективности программного изделия:

- надежность;
- быстродействие;
- длина программы;
- объем промежуточных данных;
- экономичность;
- дружественность к пользователю.

Относительная значимость их зависит от типа программного изделия:

- частная программа разового применения;
- личная процедура;
- личная библиотека;
- библиотека широкого применения (в масштабах организации);

- коммерческий программный продукт;
- программное обеспечение, встроенное в системы управления.

В любом случае приоритетной является *надежность* — в смысле корректности работы и защищенности от неправильного использования. Пункты вышеприведенного перечня расположены по возрастанию требований к ней. Соответственно будут возрастать и затраты на разработку во временно́м и стоимостном исчислении.

Основы качества программного изделия закладываются на первых этапах его разработки, которыми являются:

1. Концептуальная проработка математической модели.
2. Проверка корректности поставленной задачи и (при необходимости) ее регуляризация.
3. Преобразование задачи к форме, обеспечивающей бо́льшую точность решения.
4. Использование апробированных математических методов.
5. Выяснение областей применения и ограничений выбранных методов и сопоставление их с особенностями поставленной задачи.
6. Применение (по возможности) профессионально составленных подпрограмм.

3.2. Метод и программа

Важно различать метод и подпрограмму. Метод обычно может быть описан несколькими простыми формулами. Чем тоньше и лучше кажется метод, тем хуже он может повести себя в неподходящих условиях.

Программа является гораздо более сложным объектом. Она должна предусматривать такие детали, как выбор величины шага; контроль ошибки; управление временной памятью; обнаружение зацикливания, возможных разрывов и особенностей; взаимодействие с другими программами; выявление ошибок в вызывающей программе.

3.3. Процесс программирования

Очень важно, чтобы программа подразделялась на отдельные блоки с четко локализованными функциями. Программы следует составлять поблочно и по возможности с запасом общности — в расчете на повторное использование.

Работу нужно планировать так, чтобы максимально увеличить шансы заметить что-либо необычное. Если можно включить в процесс счета какие-либо дополнительные проверки или альтернативные методы получения того же результата, это обязательно следует сделать. Выполнение подобных проверок позволит: оценить возможности сравниваемых методов; уточнить области их применения; глубже понять исходную математическую модель; повысить степень уверенности в правильности результатов.

Не ограничивайтесь выдачей только конечного результата. К примеру, по ходу оптимизации важно оценить и крутизну целевой функции в окрестности экстремума. Это позволит установить, можно ли применять приближенные методы, какой ценой достигается улучшение на начальном и на конечном этапе.

Важно понять причины первоначальной неудачи: неполнота модели, ошибка в ее формализации, в выборе метода или его параметров, в программной реализации.

3.4. Измерение времени выполнения программы

3.4.1. Измерение трудоемкости участка

Для расчета времени выполнения участка программы можно применить процедуру `DATE_AND_TIME`. Ее ответный ключевой параметр `values` предполагает массив из 8 целых чисел, четырьмя последними компонентами которого являются часы (h), минуты (m), секунды (s), тысячные доли секунды (t). Поэтому фрагмент

```
.................
integer c(8)
integer h,m,s,t, dh,dm,ds,dt
```

```
real time
call date_and_time(values=c)
h=c(5); m=c(6); s=c(7); t=c(8)
< измеряемый участок>
call date_and_time(values=c)
dh=c(5)-h; dm=c(6)-m; ds=c(7)-s; dt=c(8)-t
time=(dh*60+dm)*60+ds+0.001*dt
```

обеспечит определение времени прогона участка в секундах (с точностью до сотых долей).

3.4.2. Профилировщик

Если сложная программа работает слишком долго, следует установить распределение трудоемкости по ее составным частям. Эту информацию можно получить с помощью специального инструмента — *профилировщика*. Наиболее трудоемкие участки подлежат более тщательному анализу и перепрограммированию — возможно, на языке ассемблера. С помощью профилировщика можно также выявить не выполнявшиеся части проекта, что существенно для управления процессом отладки и для определения безопасности заимствованных программ.

Профилировщик Ф95 выполняет три различных программы: `PREP`, `PROFILE`, `PLIST`, из которых первая готовит измерения, вторая их выполняет, а третья организует вывод.

Профилирование функций может показать:

- время, затраченное на выполнение функции, и кратность ее вызова;
- только количество вызовов;
- списки выполнявшихся и не выполнявшихся функций (покрытие функций);
- вложенность вызовов (стек подпрограмм на момент вызова).

Профилирование строк дает кратность выполнения каждой строки кода (в частности, нулевую). Этот режим требует сбора отладочной информации.

Разовую и эпизодическую потребность в профилировании проще удовлетворить «домашними средствами», т. е. собственноручно написанными вставками с применением `CPU_TIME` и самодельных счетчиков кратностей.

3.5. Повышение быстродействия программ

3.5.1. Стратегический уровень

Рост тактовой частоты и пространства оперативной памяти современных ЭВМ снизили актуальность повышения быстродействия программ и уменьшения требуемых объемов памяти. Однако совсем сбрасывать их со счетов нельзя. Всегда были и будут задачи, решение которых находится на пределе (и даже за пределами) возможностей вычислительной техники текущего дня. Быстродействие программы легче и радикальнее всего можно повысить на *алгоритмическом* уровне:

- упрощением математической модели и ее регуляризацией;
- рациональным выбором структуры данных;
- выбором подходящего к ситуации *специализированного* метода;
- разумным и сбалансированным заданием требований по точности;
- применением предварительно проверенных аппроксимаций;
- улучшением начальных приближений и ускорением сходимости итерационных процессов — по возможности, с оперативной оценкой текущей погрешности;
- применением рекуррентных вычислительных схем (см. пример в разд. 5.2).

3.5.2. Тактический уровень

Компиляторы с современных языков программирования ценой увеличения трудоемкости компиляции могут минимизировать время счета и/или требуемую память. Перечислим некоторые из выполняемых ими оптимизирующих преобразований.

Вставка тел процедур, настроенных на аргументы очередного вызова, взамен операторов вызова (указателей функций).

Реорганизация неявных циклов в списках ввода/вывода.

Вычисление константных выражений, в том числе постоянных компонент адресов для элементов массивов.

Экономия повторяющихся подвыражений.

Чистка циклов: вынесение из них повторяющихся подвыражений, замена индексируемых элементов скалярами (например, накопление суммы парных произведений при перемножении матриц в скаляре вместо ответного элемента с двумя индексами).

Переупорядочение циклов: согласование с размещением элементов массивов в памяти; настройка на работу с кэш-памятью для минимизации обращений к основной; объединение циклов с известным числом повторений и одинаковыми заголовками.

Настройка формируемого кода на конкретный тип процессора.

Эти действия распределены по *уровням оптимизации*, каждый из которых включает возможности предыдущих.

Из дополнительных мер, доступных программисту, укажем использование (по возможности):

- более быстрых в обработке типов данных: в порядке убывания предпочтительности целых (длиной в байт, полуслово и слово), вещественных одинарной и двойной разрядности, комплексных;
- быстрых альтернативных операций: использование целых показателей степени вместо вещественных; замена возведения в степень 0.5 обращением к `SQRT(·)`;
- выражений, не требующих преобразования типов;
- процедур, при работе с «массивными» параметрами не требующих создания *копий* аргументов.

В [77] приведена таблица сравнительной трудоемкости машинных операций: присваивание целых — 1, целочисленное сложение — 1.5, присваивание вещественных — 2, сложение вещественных — 3, умножение вещественных — 5, преобразование целого в вещественную форму — 6, целочисленное умножение — 8, деление — 9, возведение в целую степень — 35, возведение в вещественную степень — 115, вычисление стандартных функций — 150.

Электронная документация к Compaq Visual Fortran рекомендует заменять явно расписанные циклы с шагом на встроенные функции и операторы работы с массивами: «массивные» присваивания, конструкции `WHERE` и `FORALL`, а также применять операции ввода-вывода к массивам в целом.

3.6. Обеспечение надежности

3.6.1. Основные пути

Названная проблема подробно обсуждается в литературе по технологии программирования — см. [81] и тематический выпуск журнала «Программные продукты и системы» № 4 за 1998 г. Назовем основные пути повышения надежности численного программного обеспечения:

1. Простота концепции программы. В частности, может оказаться целесообразным создание рекурсивной подпрограммы, проигрывающей обычному варианту по быстродействию и объему памяти, но предельно ясной по идее. То же относится и к отдельным приемам программирования.
2. Стратегия разработки по схеме «быстрый прототип» с последующим наращиванием возможностей.
3. Современные организация, техническое и программное обеспечение разработки.
4. Создание документации в процессе разработки, в частности — комментирование исходных текстов программ.
5. Обязательная проверка работоспособности программы с отладочными опциями.
6. Тщательное тестирование.

3.6.2. Тестирование

Перечислим специфические методы тестирования численного программного обеспечения.

Задача с известным решением: численное интегрирование, дифференцирование, решение нелинейных уравнений.

Решение взаимообратных задач: подстановка найденного решения в исходное уравнение; перемножение прямой и найденной обратной матриц; обращение найденной обратной матрицы с целью получения исходной; расчет аппроксимирующего распределения по заданному набору моментов и вычисление моментов аппроксимации.

Перекрестное тестирование библиотечных подпрограмм на пересечении областей их применения. На рис. 3.1 приведены результаты

тестирования на различных моделях систем массового обслуживания (СМО) подходящих процедур из разработанного автором пакета программ [80, 84]. Обозначения моделей и подпрограмм даны в соответствии с нотацией Кендалла $A/B/n$, где A — тип распределения интервалов между заявками, B — тип распределения длительности обслуживания, n — число каналов. Общепринятая сигнатура типов распределений:

M — показательное (с марковским свойством),

E_k — эрланговское k-го порядка,

H_k — гиперэкспоненциальное k-го порядка,

D — детерминированное (неслучайная величина),

G — общее (задается непосредственно моментами).

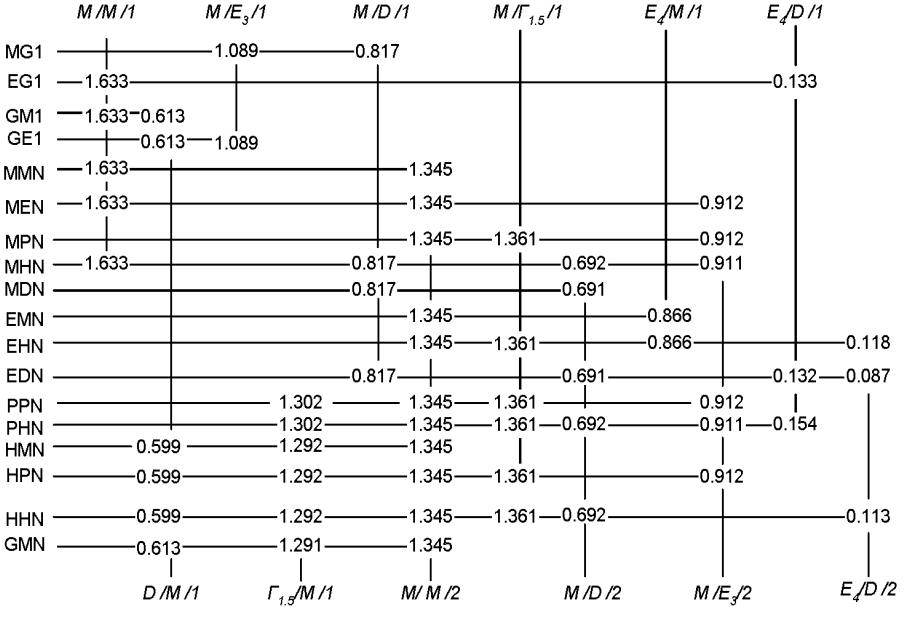

Рис. 3.1. Тестирование процедур расчета разомкнутых систем

Числа на пересечении имен модели и модуля указывают ожидаемую длину очереди: этот показатель компактен, инвариантен к масштабу времени и более чувствителен к способу расчета, чем ожидаемое число заявок в системе.

Глава 4. Приближенные вычисления

4.1. Компоненты погрешности

Точность приближенных чисел определяется их *погрешностью*. Погрешности, встречающиеся в математических задачах, могут быть разбиты на 8 групп:

1. Погрешности физической модели (отвлечение от факторов, рассматриваемых как второстепенные).
2. Погрешности, связанные с постановкой математической задачи. Математические формулировки редко точно отображают реальные явления: они обычно дают лишь более или менее идеализированные модели. Как правило, при изучении тех или иных явлений природы мы вынуждены принять некоторые упрощающие предположения, что порождает погрешность задачи.
3. Иногда бывает и так, что решить задачу в точной постановке трудно или невозможно. Тогда ее заменяют задачей, близкой по ожидаемым результатам, но решаемой, и возникает погрешность *метода*.
4. Погрешность из-за неточного знания числовых параметров модели (начальная погрешность).
5. Погрешности, порождаемые теоретически бесконечными процессами, пределами которых являются искомые решения или их компоненты, и которые обрываются после конечного числа шагов (остаточная погрешность).
6. Погрешности дискретизации во времени и пространстве (конечная величина соответствующих шагов).

7. Погрешности округления при представлении чисел в ЭВМ. Один из источников погрешности — перевод чисел из десятичной в двоичную систему и обратно.

8. Погрешности действий с приближенными числами.

Погрешности, порождаемые первыми двумя причинами, от вычислителя и используемых им технических средств не зависят и считаются неустранимыми — во всяком случае, они не могут быть уменьшены изменением организации вычислений. Для вычислителя погрешность задачи следует считать неустранимой, хотя постановщик задачи иногда может ее изменить. Нет смысла решать задачу существенно точнее, чем это диктуется погрешностью исходных данных.

Влияние остальных источников может быть ослаблено выбором более точных методов или отдельных параметров методов, а также удлинением разрядной сетки. В курсах приближенных вычислений изучаются именно это вопросы.

Как правило, погрешность численного метода может быть уменьшена до любого разумного значения путем выбора некоторого параметра (количества учтенных членов ряда, степени аппроксимирующего полинома, величины шага, числа итераций и т. п.). Погрешность метода стараются сделать несколько меньшей погрешности исходных данных.

В практических вычислениях желательна такая организация счета, когда требуемая величина представлена как малая поправка к хорошему приближению.

Перейдем к способам оценки погрешностей.

4.2. Абсолютные и относительные погрешности

Пусть A — точное значение некоторой величины, a — число, принятое за ее приближенное значение. В этом случае пишут $A \approx a$. Если известно, что $a < A$ ($a > A$), то a называют приближением по недостатку (по избытку).

Разность $A - a$ называют *погрешностью* приближенного значения a, модуль погрешности — абсолютной погрешностью приближенного значения a. Будем обозначать ее через $\bar{\Delta}$ ($\bar{\Delta}a$):

$$\bar{\Delta} = |A - a|.$$

Нахождение $\bar{\Delta}$ невозможно без определения точного значения A. Поэтому вводится понятие об оценке погрешности. *Оценкой* абсолютной погрешности приближенного значения a называют подходящее число Δ, не меньшее абсолютной погрешности, т. е. такое, что

$$\Delta \geq \bar{\Delta} = |A - a|.$$

Отсюда следует, что между числами A, a и Δ имеет место неравенство

$$A - \Delta \leq a \leq A + \Delta,$$

что принято записывать в виде

$$A = a \pm \Delta.$$

Замечание. Число Δ желательно выбирать близким к $\bar{\Delta}$. Обычно в записи Δ сохраняют две-три значащих цифры.

Пример 4.1. Известно, что число π имеет следующие 50 первых десятичных знаков:

$$\pi = 3.14159265358979323846264338327950288419716939937 51...$$

В практических расчетах его приближенное значение принимается в зависимости от задачи. Ясно, что при баллистических расчетах траекторий космических аппаратов его следует брать точнее, чем при раскрое листового материала для боковой поверхности консервной банки. В последнем случае достаточно положить $\pi \approx 3.14$. В качестве оценки погрешности Δ можно выбирать числа 0.002; 0.0016; 0.00160; 0.001593, а Δ принять равным 0.002 или 0.0016.

Пусть приближенное значение $a \neq 0$. *Относительной погрешностью* приближенного значения a называется отношение абсолютной погрешности к модулю a. Обозначая относительную погрешность через $\bar{\delta}$, будем иметь

$$\bar{\delta} = \bar{\Delta}/|a|.$$

Оценкой относительной погрешности назовем число δ, не меньшее, чем $\bar{\delta}$: $\delta \geq \bar{\delta}$. Из этого определения следует, что в качестве δ можно выбрать отношение

$$\delta = \Delta/|a|,$$

так что

$$\Delta = |a|\delta. \qquad (4.2.1)$$

Формула (4.2.1) связывает оценки абсолютной и относительной погрешностей.

Обычно оценка погрешности производится для наихудших вариантов, и фактическая погрешность может оказаться значительно меньше теоретической.

В заключение этого раздела отметим наличие у расчетчика некоторой степени свободы. Если требуется найти решение с погрешностью 10^{-2}, обычно подразумевается *порядок* погрешности: $1.5 \cdot 10^{-2}$ скорее всего вполне устроят заказчика.

4.3. Особенности машинной арифметики

В машинных вычислениях:

- числа представлены с ограниченным числом разрядов;
- выполняется огромное число шагов, что приводит к заметному росту *накопленных* ошибок;
- расчетчик обычно не видит промежуточных результатов;
- большая размерность задач может потребовать принципиально новых методов (см. формулы Крамера).

Точность представления чисел в ЭВМ зависит не от самого маленького по модулю числа, представимого в машине, но от длины разрядной сетки — точнее, количества разрядов, отводимого для хранения мантиссы числа с плавающей точкой. Последнее определяет $\varepsilon_{\text{маш}}$ — наименьшее число с плавающей точкой, которое при сложении с единицей дает число, большее 1. Им ограничивается относительная погрешность сложения чисел с плавающей точкой. В стандартном случае работы на ПЭВМ с одинарной точностью $\varepsilon_{\text{маш}} = 1.2 \cdot 10^{-7}$, так что нельзя рассчитывать более чем на 7 верных десятичных знаков (в формате двойного слова — на 16).

Стандарт IEEE рекомендует, чтобы компьютеры выполняли арифметику с большей разрядностью (арифметика с плавающей точкой — с внутренней разрядностью 80 битов).

К этому следует добавить, что результаты каждой отдельной операции как правило не подвергаются правильному округлению: во избежание

дополнительных задержек из-за переносов единицы в старшие разряды младшие разряды результата просто отбрасываются. При наличии соответствующих аппаратных возможностей и их учете в языке программирования и трансляторе способом округления можно управлять.

При счете на ЦВМ приходится считаться и с ограниченным *диапазоном* чисел, представимых в разрядной сетке машины. Пороги переполнения и потери значимости определяются количеством разрядов, отводимых на представление двоичного порядка числа. При программировании нужно выбирать такую последовательность действий, при которой исключаются переполнение разрядной сетки и обращение промежуточных результатов умножения в машинные нули. В первом случае вычисления завершатся «авостом» (машина, разумеется, не сломается); во втором теоретически ненулевое произведение окажется нулем.

В некоторых системах программирования для стандартного представления целых чисел со знаком отводятся 16 двоичных разрядов, что ограничивает максимальный модуль таких чисел величиной 32767. Этого практически достаточно для организации циклов и выборки элементов массивов. Однако при реализации целочисленной арифметики могут возникнуть затруднения. Например, для программирования на Паскале была предложена задача вычисления π с помощью степенного ряда

$$\sum_{k=0}^{\infty} \frac{1}{(4k+1)(4k+3)(4k+5)} = \pi/16 - 1/8$$

с выводом пошаговой погрешности для 20 шагов. Расчет знаменателя в целых числах уже на восьмом шаге привел к переполнению (должно получиться 42375) — без выдачи сигнальной информации, что затруднило диагностику ситуации. В таких случаях следует перейти либо к удлиненному формату целых (в Паскале — `longint`), либо к числам с плавающей точкой. Заметим, что в Паскале вещественная переменная не может быть параметром цикла. В Фортране последнего ограничения нет, а стандартный формат целых допускает значения свыше $2 \cdot 10^9$.

Увеличение разрядности современных ЭВМ уменьшает роль последовательности операций; рост быстродействия — их суммарной трудоемкости; увеличение объема оперативной памяти — связности алгоритма (объема памяти под промежуточные результаты). Все это оправдывает применение более простых методов и тем повышает надежность программ. Эти же факторы позволяют решать задачи большей сложности, с большей точностью и за более короткие сроки, что позволяет внедрять в автоматические системы более сложные алгоритмы управления.

4.4. Погрешности функций

Пусть величина u определена как функция переменных x_1 и x_2: $u = f(x_1, x_2)$. Будем предполагать, что $f(x_1, x_2)$ имеет непрерывные частные производные всех порядков, которые потребуются для дальнейшего рассмотрения. Найдем зависимость абсолютной погрешности и от погрешностей задания x_1 и x_2.

Теорема 4.1. Пусть x_1^0 и x_2^0 суть приближенные значения x_1 и x_2, а Δx_1 и Δx_2 — оценки их абсолютных погрешностей. Обозначим через M_1 и M_2 максимумы $|\frac{\partial f}{\partial x_1}|$ и $|\frac{\partial f}{\partial x_2}|$ в прямоугольнике

$$\begin{aligned} x_1^0 - \Delta x_1 &\le x_1 \le x_1^0 + \Delta x_1, \\ x_2^0 - \Delta x_2 &\le x_2 \le x_2^0 + \Delta x_2. \end{aligned}$$

Тогда абсолютная погрешность величины $\bar{\Delta} u$ величины $u = f(x_1, x_2)$ не превосходит $M_1 x_1 + M_2 x_2$:

$$\bar{\Delta} u \le M_1 \Delta x_1 + M_2 \Delta x_2.$$

Доказательство. Пусть $u^0 = f(x_1^0, x_2^0)$. Используя теорему о среднем, имеем

$$\begin{aligned} u - u^0 &= f(x_1, x_2) - f(x_1^0, x_2^0) \\ &= [f(x_1, x_2) - f(x_1^0, x_2)] + [f(x_1^0, x_2) - f(x_1^0, x_2^0)] \\ &= f'_{x_1}(x_1', x_2)(x_1 - x_1^0) + f'_{x_2}(x_1^0, x_2')(x_2 - x_2^0). \end{aligned}$$

Следовательно,

$$\bar{\Delta} u = |u - u_0| \le |f'_{x_1}(x_1', x_2)(x_1 - x_1^0)| + |f'_{x_2}(x_1^0, x_2')(x_2 - x_2^0)| \le M_1 \Delta x_1 + M_2 \Delta x_2,$$

а потому в качестве оценки абсолютной погрешности можно принять

$$\Delta u \le M_1 \Delta x_1 + M_2 \Delta x_2.$$

▲

Замечание. При малых Δx_1 и Δx_2 обычно считают

$$\Delta u = |f'_{x_1}(x_1^0, x_2^0)| \Delta x_1 + |f'_{x_2}(x_1^0, x_2^0)| \Delta x_2.$$

Для функций n переменных

$$\Delta u = \sum_{k=1}^{n} |f'_{x_k}(x_1^0, x_2^0, \dots x_n^0)| \Delta x_k. \qquad (4.4.1)$$

Следствия.

1. Если $u = x_1 \pm x_2 \pm \ldots \pm x_n$, то $\partial f/\partial x_k = \pm 1$, и $M_k = 1$, $k = \overline{1,n}$. Таким образом, абсолютная погрешность алгебраической суммы нескольких величин не превосходит суммы оценок абсолютных погрешностей слагаемых.

2. Пусть
$$u = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n},$$
где $\{\alpha_i\}$ — константы. Тогда
$$\frac{\partial u}{\partial x_k} = x_1^{\alpha_1} \ldots x_{k-1}^{\alpha_{k-1}} \alpha_k x_k^{\alpha_k - 1} x_{k+1}^{\alpha_{k+1}} \ldots x_n^{\alpha_n} = \frac{\alpha_k}{x_k} u,$$
и для оценки абсолютной погрешности имеем
$$\Delta u = |u| \sum_{k=1}^{n} \left| \frac{\alpha_k}{x_k} \right| \Delta x_k.$$
Поэтому оценка относительной погрешности
$$\delta u = \frac{\Delta u}{|u|} = \sum_{k=1}^{n} |\alpha_k| \delta x_k. \qquad (4.4.2)$$

В частности, оценка относительной погрешности *произведения* равна сумме относительных погрешностей сомножителей. Оценка относительной погрешности частного равна сумме оценок относительных погрешностей делимого и делителя.

Оценка относительной погрешности *суммы* при одинаковых знаках слагаемых не превосходит наибольшей оценки относительной погрешности слагаемых.

В литературе по вычислительной математике обычно указывается, что при суммировании с малой разрядностью нужно по возможности начинать с малых по модулю слагаемых: в противном случае они могут оказаться несоизмеримыми с накопленной суммой и не окажут на нее должного влияния. Обсуждаемый эффект усугубляется отбрасыванием (вместо округления) дополнительных разрядов результата. В частности, в [10] предлагается тестовый пример
$$\sum_{i=1}^{1000000} 1/i^2.$$

Автор проверил предполагаемый эффект на задачах

$$\sum_{i=1}^{10000} 1/i^2$$

и

$$\sum_{i=1}^{10000} 1/(10001-i)^2$$

с помощью математического пакета Scientific WorkPlace, позволяющего менять разрядность вычислений. Предельное теоретическое значение этой суммы

$$\sum_{i=1}^{\infty} 1/i^2 = \pi^2/6 = 1.644934066848226.$$

Разница в единицу четвертого знака была обнаружена только при счете в четырехразрядной сетке. Этого и следовало ожидать, поскольку слагаемые в данном случае убывают слишком быстро и сумма стабилизируется рано. В иных ситуациях с упомянутым эффектом следует считаться.

В случае *разных* знаков следует избегать вычитания близких чисел, при котором происходит катастрофическая потеря верных цифр. Можно, например, воспользоваться тождеством

$$a-b = \frac{a^2-b^2}{a+b}.$$

Его можно применить к решению квадратного уравнения $ax^2+bx+c=0$, когда $\sqrt{D} \approx b$ и расчет одного из корней сводится к вычитанию близких чисел. При $b>0$

$$x_2 = \frac{\sqrt{D}-b}{2a}\frac{\sqrt{D}+b}{\sqrt{D}+b} = \frac{D-b^2}{2a(\sqrt{D}+b)} = \frac{-4ac}{2a(\sqrt{D}+b)} = -\frac{2c}{\sqrt{D}+b}.$$

Второй корень можно получить с помощью одного из условий

$$x_1+x_2 = -b/a, \quad x_1 \cdot x_2 = c/a.$$

При возведении в степень приближенного числа его относительная погрешность умножается на показатель степени.

Для функции одной переменной как частный случай (4.4.1) имеем

$$\Delta u = |f'(x)|\Delta x.$$

Если воспользоваться теоремой о среднем значении, то получим $\bar{\Delta}u = |f'(\xi)|\bar{\Delta}$. Поскольку модуль производной таких функций, как $\sin, \cos, \operatorname{arctg}, \operatorname{arcctg}$, не превосходит единицы, то для этих функций их абсолютная погрешность не превосходит абсолютной погрешности аргумента; по аналогичным соображениям абсолютная погрешность функций $\operatorname{tg}, \operatorname{ctg}, \arcsin, \arccos$ *не меньше* абсолютной погрешности аргумента.

4.5. Обратная задача теории погрешностей

Под обратной задачей теории погрешностей понимается нахождение предельных оценок погрешностей аргументов, при которых погрешность функции не превзойдет допустимой величины. Способ решения обратной задачи для функции одного аргумента очевиден. Но для функции n аргументов одна и та же суммарная оценка погрешности, вычисляемая по формуле (4.4.1), может быть получена при различных распределениях оценок абсолютных погрешностей аргументов. По этой причине для решения обратной задачи теории погрешностей необходимо привлечение дополнительной информации. Обычно пользуются принципом «равных влияний» и принимают все слагаемые в формуле (4.4.1) равными. Тогда

$$\frac{\Delta u}{n} = \left|\frac{\partial f}{\partial x_k}\right| \Delta x_k, \qquad k = \overline{1,n}$$

и

$$\Delta x_k = \frac{\Delta u}{n} \Big/ \left|\frac{\partial f}{\partial x_k}\right|.$$

Пример 4.2. Имеется цилиндрический бак с диаметром основания $D = 2.2$ м и высотой $H = 4$ м. Каковы допустимые погрешности в измерении D и H и с какой точностью нужно задать π, чтобы объем бака $V = \pi D^2 H/4$ можно было вычислить с точностью 0.1 м^3?

Решение. Прежде всего отметим, что в данной задаче, поскольку мы должны выбрать приближенное значение π, имеет смысл считать π переменной величиной. Вычислим производные

$$\begin{array}{rcrcl} \partial V/\partial \pi & = & D^2 H/4 & \approx & 4.84; \\ \partial V/\partial H & = & \pi D^2/4 & \approx & 3.80; \\ \partial V/\partial D & = & \pi D H/2 & \approx & 13.80 \end{array}$$

и найдем допустимые значения составляющих погрешности:

$$\frac{\Delta V}{3} = \frac{0.1}{3} = 0.0333.$$

Отсюда

$$\begin{array}{rcll} \Delta\pi & = & 0.1/(3 \cdot 4.84) & \approx \ 0.0069; \\ \Delta H & = & 0.1/(3 \cdot 3.80) & \approx \ 0.00925 \text{ [м]}; \\ \Delta D & = & 0.1/(3 \cdot 13.8) & \approx \ 0.0024 \text{ [м]}. \end{array}$$

Возьмем диаметр с точностью 2 мм, высоту — 1 см, а $\pi \approx 3.14$. Тогда $\Delta V \leq 4.84 \cdot 0.0016 + 3.80 \cdot 0.01 + 13.80 \cdot 0.002 = 0.063$ м^3, и заданное условие выполнено.

С другой стороны, значение π нам известно практически точно, а для измерения бака под рукой может не оказаться достаточно удобного инструмента. Попытаемся снизить потребную точность измерений. Теперь

$$\begin{array}{rcll} \Delta H & = & 0.1/(2 \cdot 3.80) & \approx \ 0.0133 \text{ [м]}; \\ \Delta D & = & 0.1/(2 \cdot 13.8) & \approx \ 0.0036 \text{ [м]}. \end{array}$$

Возьмем высоту с точностью 1 см, а диаметр — с точностью 4 мм. Тогда $\Delta V \leq 3.80 \cdot 0.01 + 13.80 \cdot 0.004 = 0.094$ м^3, и требуемая точность вычисления объема достигается.

4.6. Гарантированная и вероятностная оценки погрешности

Многие дедуктивные рассуждения проигрывают в эффективности из-за того, что они ориентированы на справедливость во всех случаях, в том числе самых неблагоприятных. Была бы желательна разработка рабочих оценок, справедливых «в среднем».

Выше было показано, что оценка абсолютной погрешности суммы n слагаемых равна сумме оценок абсолютных погрешностей слагаемых:

$$\Delta u = \Delta x_1 + \Delta x_2 + \ldots + \Delta x_n.$$

Если оценки абсолютных погрешностей слагаемых равны, то $\Delta u = n\Delta x$. В этом случае получают максимально возможную погрешность, которая не будет превышена даже при условии, что все слагаемые одновременно взяты по избытку или по недостатку. Обычно дело обстоит не так

плохо, и ошибки отдельных слагаемых частично компенсируются. В теории вероятностей доказывается, что дисперсия суммы n независимых слагаемых $D_u = \sum_{i=1}^{n} D_i$. При равных точностях $D_u = nD_x$. Соответственно, среднеквадратическое отклонение $\sigma_u = \sigma_x\sqrt{n}$. Часто считают, что предельное отклонение $\Delta \le 3\sigma$ (правило «трех сигм»). Умножая предыдущее равенство на 3, получаем $\Delta u = \Delta x\sqrt{n}$. Таким образом, с увеличением числа слагаемых абсолютная погрешность суммы растет пропорционально $\sqrt{n}$. Аналогичный результат может быть установлен для относительной погрешности произведения.

4.7. Число верных цифр результата

Всякое число a может быть представлено бесконечной или конечной десятичной дробью вида $\alpha_m\alpha_{m-1}\ldots\alpha_{m-n+1}\ldots$, значение которой

$$a = \alpha_m 10^m + \alpha_{m-1}10^{m-1} + \ldots + \alpha_{m-n+1}10^{m-n+1} + \ldots$$

Здесь $\alpha_m, \alpha_{m-1}, \ldots$ — цифры этого числа (мы взяли старшие n цифр). Будем считать $\alpha_m \ne 0$ первой значащей цифрой числа a, α_{m-n+1} — n-й значащей цифрой. Очевидно,

$$\alpha_m 10^m \le |a| \le (\alpha_m + 1)10^m.$$

Пример 4.3. Если a = 47.31964..., то $m = 1$, $\alpha_m = 4$. Говорят, что приближенное значение a величины A имеет n верных первых цифр, если абсолютная погрешность не превосходит половины единицы n-го разряда в записи числа a:

$$\bar{\Delta} = |a - A| \cdot 10^{m-n+1}/2.$$

Если некоторое число имеет n верных цифр, то за оценку его абсолютной погрешности будем принимать половину единицы n-го разряда.

В последующих примерах считаем, что A записано в виде десятичной дроби.

Пример 4.4. Пусть A = 3.785734... Тогда a =3.785 имеет 4 верных цифры, так как $|A - a| = 0.00031\ldots < 0.0005$, т. е. меньше половины единицы *четвертого* слева разряда.

Пример 4.5. Пусть A=10.00237 ... , a = 9.99999. В этом случае $|A-a|$=0.00238...<0.005, т. е. меньше половины единицы *третьего* слева разряда. Значит, здесь три верных знака.

Существует связь между числом верных цифр и оценкой относительной погрешности числа. Можно показать, что при $n \geq 2$ для оценки относительной погрешности приближенной записи числа имеет место формула

$$\delta = 1/(2\alpha_m 10^{n-1}).$$

Точность приближенного числа зависит не от количества заданных цифр, а от числа *верных*. При решении прикладных задач исходные данные обычно имеют разную точность, и погрешность результата зачастую определяется наибольшей из погрешностей участвующих в операции чисел (операндов). При машинном счете сохранение во всех операциях максимального числа значащих цифр хотя и не увеличивает трудоемкость вычислений, но не приводит к повышению точности и затрудняет подготовку исходных данных. Отсюда вытекает целесообразность округления исходных данных, т. е. сохранения минимально необходимого количества верных цифр.

4.8. Корректность математической модели

4.8.1. Корректность задач и методов

По Адамару, задача считается корректно поставленной, если ее решение:

- существует,
- единственно,
- устойчиво по отношению к возмущениям.

К середине XX века накопилось множество содержательных некорректных задач, требующих математического обоснования и создания устойчивых методов их решения. Таковыми являются задачи восстановления сигнала, обратные задачи теплопроводности, геофизики, астрофизики и др. Математические сложности заключаются в том, что обратный

оператор A^{-1}, заданный на всей своей области определения, не является непрерывным. Примером неустойчивых моделей являются задачи линейной алгебры с плохо обусловленными матрицами, решение некоторых видов интегральных уравнений и др. Такие модели должны быть подвергнуты регуляризации: замене точного решения уравнения на приближенное в смысле минимума среднеквадратической невязки левой и правой частей; замене задачи корректной с дополнительным малым параметром, при устремлении которого к нулю решение новой задачи переходит в решение исходной; замене неизвестных функций на функции известного типа, задаваемые небольшим числом параметров. На языке техники регуляризация равносильна некоторой фильтрации: несущественно искажая гармоники с малой частотой, она сильно ослабляет высокочастотные ([10, с. 210]).

Итак, смысл имеет только решение устойчивой задачи. Однако иногда пытаются получить его *неустойчивым методом*, приводящим к систематическому накоплению погрешности. Неустойчивость может порождаться и неправильным выбором параметров метода — например, шага изменения независимой переменной при численном решении дифференциальных уравнений. Опять же классическим примером здесь является вычисление серии интегралов

$$I_n = \int_0^1 x^n e^{x-1}\,dx, \qquad n = 1, 2, \ldots$$

по рекуррентной формуле

$$I_n = x^n e^{x-1}\Big|_0^1 - n\int_0^1 x^{n-1} e^{x-1}\,dx = 1 - nI_{n-1}.$$

Отправляясь от начального значения $I_1 = 1/e \approx 0.367879$, мы довольно быстро приходим к заведомо абсурдным отрицательным значениям $\{I_n\}$. Дело в том, что модуль малой ошибки $4 \cdot 10^{-7}$ в определении I_1 к моменту вычисления I_n умножается на $n!$.

Неустойчивым методам вычисления, как правило, есть альтернативы. Корректным методом решения приведенной выше задачи будет *обратная* рекурсия $I_{n-1} = (1 - I_n)/n$, которая при достаточно большом n может быть начата с $I_n = 0$.

4.8.2. Корректность систем линейных уравнений

При описании физических моделей крайне редко используются системы

$$\tilde{A}\tilde{x} = \tilde{b} \tag{4.8.1}$$

с точными исходными данными. Обычно задаются приближенные уравнения

$$Ax = b \tag{4.8.2}$$

с указанием погрешности в исходных данных:

$$||\tilde{A} - A|| = ||\Delta A|| \le \varepsilon_A, \qquad ||\tilde{b} - b|| = ||\Delta b|| \le \varepsilon_b. \tag{4.8.3}$$

В качестве формального решения задачи (4.8.1–4.8.3) может быть взят любой вектор, который обращает уравнение (4.8.1) с матрицами A' и правыми частями b', удовлетворяющими неравенствам (4.8.3), в тождество.

Точное решение системы с точными, но неизвестными исходными данными назовем *физическим* решением; точное решение системы (4.8.2) — *математическим* решением; полученное численным методом с погрешностями перевода в двоичную систему и обратно, погрешностями метода и округления — *машинным*.

Необходимо определить понятие искомого решения, построить устойчивый алгоритм этого решения, оценить в ходе решения погрешность машинной реализации (близость машинного и математического решений), а также оценить наследственную погрешность (близость физического и математического решений).

Отыскание классического решения x системы (4.8.2), т. е. вектора x, для которого невязка $r = b - Ax$ тождественно равна нулю, будет корректно поставленной задачей, если $m = n$, $\det A \neq 0$ и

$$||\Delta A|| \cdot ||A^{-1}|| < 1$$

при произвольном возмущении ΔA из окрестности, определяемой условием (4.8.3).

Глава 5. Вычисление значений функции

В настоящей главе рассматриваются приемы сведения вычисления некоторых функций к повторяющимся арифметическим операциям со степенными рядами, многочленами и дробями. Иногда удобно приближать функции тригонометрическими полиномами или аппроксимировать $\ln f(x)$.

5.1. Вычисление многочленов

Рассмотрим многочлен

$$P_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n.$$

Его можно преобразовать к виду

$$P_n(x) = (\ldots((a_0 x + a_1)x + a_2)x + \ldots + a_{n-1})x + a_n.$$

Эта схема сводится к циклическим операциям умножения предыдущего результата на x и прибавления очередного коэффициента многочлена. Эти операции повторяются, пока не будет прибавлен коэффициент a_n. Начальным значением результата должен служить старший коэффициент a_0:

```
p=a(0)
do i=1,n
   p=p*x+a(i)
end do
```

Описанный метод — так называемая схема Горнера (1819 г.), предложенная И. Ньютоном на 100 лет раньше [37, с. 121], — требует всего *одной* ячейки памяти ЦВМ для запоминания промежуточных результатов.

По аналогичной схеме организуется и вычисление многочленов, содержащих *обобщенные* степени

$$x^{[k]} = x(x-1)\dots(x-k+1).$$

5.2. Рекурсивные и рекуррентные вычисления

Многие задачи можно свести к совокупности более простых, те подвергнуть дальнейшему упрощению и т. д. — пока не придем к задаче с известным решением. После этого начинается обратный процесс движения к исходной задаче. Классическим (и всем надоевшим) примером такой проблемы является вычисление факториала $k! = k \cdot (k-1)!$, $1! = 1$. Более интересна сортировка большого массива через его разбиения и слияние отсортированных частей. Достоинством рекурсии является ее логическая прозрачность и как следствие — повышенная надежность программной реализации (в умелых руках).

Приведем пример рекурсивного расчета многочленов Чебышева (см. разд. 10.4.2). Эти многочлены вычисляются по формулам

$$\begin{aligned} T_0(x) &= 1, \quad T_1(x) = x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x), \quad k = 2, 3, \dots \end{aligned}$$

— см. программу:

```
program chebrec
   real x /2/,y
   integer k
   interface
      recursive function cheb(k,x) result(tch)
         integer k
         real    tch
      end function cheb
   end interface
   do k=0,3
```

```
      y=cheb(k,x)
      print *, 'k = ',k, ' y = ',y ! 1, 2, 7, 26
   end do
end program chebrec

recursive function cheb(k,x) result(tch)
   integer       k
   real,save ::  t0
   real          t1,tch
   t0=1.0; t1=x
   if (k==0) then
      tch=t0
   else if (k==1) then
          tch=t1
        else
          t1=cheb(k-1,x)
          tch=2*x*t1-t0
          t0=t1
        end if
   end if
end function cheb
```

В рекурсивной функции cheb принципиальную роль играет атрибут SAVE переменной T0, обеспечивающий сохранение ее значений между вызовами. Для таких переменных инициализация выполняется только при первом вызове процедуры.

Многочисленные примеры рекурсивных программ вычислительных задач на языке Пролог и соответствующие упражнения приводятся в [83]. Приведем записанную на Прологе программу быстрого возведения заданного числа в целую положительную степень. Четная степень получается возведением в квадрат половинной степени, а нечетная — умножением на основание предыдущей нечетной.

```
domains
    n=integer
    x,p=real
predicates
    fastpow(x,n,p)
clauses
    fastpow(X,0,1).    /* Нулевая степень  X - единица  */
```

```
    fastpow(X,N,R):-   /* Общий случай                  */
    Q=N mod 2,         /* Если N - четное              */
    M=N div 2,         /* Половинная степень           */
    fastpow(X,M,Q),    /* Возведение в нее X           */
    R=Q*Q.             /* Квадрат результата, конец    */
    fastpow(X,N,R):-   /* Иначе                        */
    M=N-1,             /* Уменьшение степени на 1      */
    fastpow(X,M,Q),    /* Возведение X в эту степень   */
    R=Q*X.             /* Домножение на X, конец       */
goal
    fastpow(5,8,P),    /* Возведение 5 в степень 8     */
    write("P = %e",P).
```

Результат 590625 выводится на печать с плавающей точкой. Он же был получен при последовательном умножении.

Реализация рекурсии связана с повторным входом в блок (процедуру) до выхода из предыдущего вхождения. Соответственно, она порождает одновременно существующие копии блока в количестве, определяемом кратностью выполняемой рекурсии, и может быть весьма накладной по расходу памяти и времени счета. Если рекурсию удается заменить рекуррентным счетом, такую возможность нужно обязательно использовать. Приведем *рекуррентный* вариант программы решения той же задачи о многочленах Чебышева:

```
program recurr
   real x /2/,y
   integer k
   interface
      real function cheb(k,x)
         integer k
         real    x
      end function cheb
   end interface
   do k=0,3
      y=cheb(k,x)
      print *, 'k = ',k, ' y = ',y ! 1, 2, 7, 26
   end do
end program recurr
```

```
real function cheb(k,x)
   integer   k
   real      t0,t1
   t0=1.0;   t1=x
   if (k==0) then
      cheb=t0
   else if (k==1) then
      cheb=t1
   else
      do n=2,k
         t2=2*x*t1-t0
         t0=t1
         t1=t2
      end do
      cheb=t2
   end if
end function cheb
```

5.3. Расчет степенных рядов

5.3.1. Общие сведения

Прежде всего отметим, что во многих случаях решение удается получить в замкнутой форме. Неожиданно часто суммируются некоторые специальные ряды — например, содержащие биномиальные коэфициенты. К примеру,

$$\sum_{i=0}^{n} \binom{n}{i} = (1+1)^n = 2^n.$$

Советуем читателю, прежде чем обращаться к вычислительной машине, попытаться просуммировать ряд «руками» (точнее, головой).

Многие трансцендентные функции могут быть представлены суммами вида

$$f(x) = \sum_{k=0}^{\infty} a_k x^k = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k\,!} x^k$$

(ряд Маклорена). Оборвав разложение на n-м члене, можно применять усеченный ряд для приближенного определения значения функции

$f(x)$. Вопрос о номере члена, на котором можно прекратить вычисления, решается на основе оценки погрешности суммы ряда.

Разложения в ряд Тейлора часто оказываются неэффективными — например, из-за быстрого роста сложности производных с увеличением их порядка. Соответственно увеличивается время счета. Другим их недостатком является неравномерная (при фиксированном числе членов) точность приближения, резко падающая с увеличением модуля аргумента. При аппроксимации рядом Тейлора нужно начинать с аппроксимаций невысокого (2–3) порядка и убедиться в отсутствии разрывов функции и ее производных в окрестности базовой точки.

5.3.2. Признаки сходимости рядов

Прежде чем считать степенной ряд, убедитесь в его сходимости.

Согласно *признаку Лейбница*, ряд со знакочередующимися членами сходится, если модули членов ряда строго убывают. Погрешность R_n суммы n его членов по абсолютной величине не превосходит модуля первого отброшенного[1].

При отсутствии чередования знаков следует попытаться подобрать мажорирующую члены остатка ряда по модулю *убывающую* геометрическую прогрессию и с ее помощью оценить сверху упомянутый остаток. Так, в разложении

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \qquad (5.3.1)$$

члены ряда подсчитываются согласно $b_k = x^k/k!$, откуда следует возможность вычислять их по формуле

$$b_k = b_{k-1} \cdot x/k, \qquad k = 1, 2, \dots$$

При $k > |x|$ члены ряда станут убывать по модулю. Пусть n_0 — некоторое значение k, отвечающее этому условию. Тогда при $k > n_0$ будет $|x|/k < |x|/n_0$, и модуль остатка ряда

$$\begin{aligned} |R_{n_0}| &= \Big| \sum_{k=n_0+1}^{\infty} \frac{|x|^k}{k!} \Big| < \frac{|x|^{n_0}}{n_0!} \left[\frac{|x|}{n_0} + \left(\frac{|x|}{n_0}\right)^2 + \left(\frac{|x|}{n_0}\right)^3 + \dots \right] \\ &= |b_{n_0}| \cdot (|x|/n_0)/(1 - |x|/n_0). \end{aligned}$$

[1] Разумеется, если этот член вычислен, его все же следует прибавить к накопленной сумме.

Если $|x/n_0| < 0.5$, то $|R_{n_0}| \le |b_{n_0}|$, и погрешность частичной суммы не превзойдет модуля последнего учтенного члена.

Признак Даламбера основан на аналогии с суммой бесконечной убывающей геометрической прогрессии. Если

$$|q| = \lim_{n\to\infty} \frac{a_{n+1}}{a_n} < 1,$$

то ряд сходится.

Для рядов с положительными членами известен более сильный *интегральный признак Коши*: если интеграл $\int_1^\infty f(n)\,dn$, где $f(n)$ — выражение для n-го члена, сходится, то сходится и степенной ряд.

Нужно иметь в виду: если ряд не является абсолютно сходящимся, то перестановка его членов может изменить его сумму и даже сделать ряд расходящимся.

5.3.3. Расчет членов рядов

Рассмотрим особенности расчета членов степенных рядов. Возьмем, например, ряд для неполной гамма-функции

$$\gamma(x,a) = \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{k\,!(k+a)}.$$

Было бы неумно, имея вычисленным шестой этого ряда $x^6/6\,!(a+6)$, заново считать $(-1)^7$, x^7 и $7!$. Поэтому при программировании выражений для расчета общего члена ряда стремятся выделить элементы, связанные с аналогичными элементами предыдущего члена простыми рекуррентными (пересчетными) зависимостями, как это было сделано в предыдущем пункте. В данном случае целесообразно представить общий член ряда в виде $b_k = c_k/d_k$, где $c_k = c_{k-1}(-x/k)$, $d_k = d_{k-1} + 1$, $k = 1, 2, \ldots$, при очевидных начальных значениях $c_0 = 1$, $d_0 = a$. Заметим, что пересчет степеней и факториалов производится с помощью умножения, а сумм — путем сложения. Разумеется, новые элементы нужно засылать в те же ячейки памяти ЦВМ, в которых были их аналоги на предыдущем шаге (иначе говоря, сохранять имена переменных).

Покажем, как по этой схеме рассчитывается распределение числа событий простейшего потока за случайный интервал времени. Обозначим через λ интенсивность входящего потока. Тогда вероятность появления ровно j событий потока за случайное время, подчиненное распределе-

нию $B(t)$, должна вычисляться согласно

$$q_j = \int_0^\infty \frac{(\lambda t)^j}{j\,!} e^{-\lambda t}\, dB(t), \qquad j = 0, 1, \ldots \tag{5.3.2}$$

Указанные вероятности играют важную роль в расчете сложных систем обслуживания, и необходимо уметь эффективно их вычислять для возможно более широкого круга распределений. Специальный интерес вызывает расчет вероятности q_0, которая может рассматриваться как преобразование Лапласа с параметром λ от распределения $B(t)$.

Для гамма-распределения с параметром формы r плотность распределения

$$b(t) = \frac{\mu(\mu t)^{r-1}}{\Gamma(r)} e^{-\mu t}.$$

В этом случае

$$\begin{aligned} q_j &= \int_0^\infty \frac{(\lambda t)^j}{j\,!} e^{-\lambda t} \frac{\mu(\mu t)^{r-1}}{\Gamma(r)} e^{-\mu t}\, dt \\ &= \frac{\lambda^j \mu^r}{j\,!\,\Gamma(r)} \int_0^\infty t^{r+j-1} e^{-(\lambda+\mu)t}\, dt \\ &= \frac{\lambda^j \mu^r}{j\,!\,(\lambda+\mu)^{r+j}\,\Gamma(r)} \int_0^\infty u^{r+j-1} e^{-u}\, du \\ &= \Big(\frac{\mu}{\lambda+\mu}\Big)^r \Big(\frac{\lambda}{\lambda+\mu}\Big)^j \frac{\Gamma(r+j)}{j\,!\,\Gamma(r)}, \qquad j = 0, 1, \ldots \end{aligned} \tag{5.3.3}$$

Получим рекуррентные формулы вычисления $\{q_j\}$ при времени обслуживания, подчиненном гамма-распределению. Прежде всего, из (5.3.3) следует $q_0 = (\mu/(\lambda+\mu))^r$. Имея в виду, что $\Gamma(m+1) = m\Gamma(m)$, выводим

$$\frac{q_j}{q_{j-1}} = \frac{\lambda}{\lambda+\mu} \frac{\Gamma(r+j)\cdot(j-1)\,!}{j\,!\,\Gamma(r+j-1)} = \frac{\lambda}{\lambda+\mu} \frac{r+j-1}{j}.$$

Итак, при гамма-распределении

$$\begin{aligned} q_0 &= \Big(\frac{\mu}{\lambda+\mu}\Big)^r, \\ q_j &= q_{j-1} \frac{\lambda}{\lambda+\mu} \frac{r+j-1}{j}, \qquad j = 1, 2, \ldots \end{aligned} \tag{5.3.4}$$

Оказывается, для расчета интересующих нас вероятностей даже не требуется обращаться к гамма-функции!

Весьма поучителен фрагмент программы расчета произведения Валлиса

$$\frac{\pi}{2} = \frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdots$$

```
. . . . . .
    p=1.0; c=2.0; d=1.0;
    b=3.0; k=1
    do while(abs(b-1.0)>eps)
        b=c/d; p=p*b
        if (k>0) then
            d=d+2
        else
            c=c+2
        end if
        k=-k
    end do
. . . . . .
```

Эту же идею с «кувыркающейся единицей» k можно применять при работе со знакочередующимися членами и при вычислении *двойных* факториалов, где дополнительный сомножитель приходится приписывать через шаг, например при вычислении

$$b_n = (-1)^n \frac{x^{2n+1}\cdot(2n-1)!!}{[(2n)!!]^2}.$$

Еще один эффектный пример — ряд Рамануджана для вычисления $1/\pi$:

$$\frac{1}{\pi} = 12\sum_{n=0}^{\infty}(-1)^n \frac{(6n)\,!}{(n\,!)^3(3n)\,!}\,\frac{13591409+545140134n}{(640320^3)^{n+1/2}}.$$

Члены этого ряда имеет смысл представить в виде $b_n = c_n\cdot d_n$, где d_n есть числитель второй дроби, а c_n — вся остальная часть общего члена. Полагая $f = 640320^3 \approx 2.62\cdot 10^{17}$, имеем $c_0 = 12/\sqrt{f}$ и пересчетный множитель

$$\frac{c_n}{c_{n-1}} = \frac{(-1)^n(6n)\,!\times[(n-1)\,!]^3(3n-3)\,!f^{n-1/2}}{(n\,!)^3(3n)\,!f^{n+1/2}\times(-1)^{n-1}(6n-6)\,!}$$

$$= -\frac{6n(6n-1)(6n-2)(6n-3)(6n-4)(6n-5)}{n^3 \cdot 3n(3n-1)(3n-2) \cdot f}$$
$$= -8(6n-1)(6n-3)(6n-5)/(n^3 \cdot f).$$

Очевидно, d_0 =13591409, $d_n = d_{n-1}$ +545140134.

5.3.4. Ускорение сходимости рядов

Ряд Рамануджана сходится поразительно быстро. Уже величина, обратная нулевому члену, отличается от π на $-5.92{\cdot}10^{-14}$. Первый член ряда составляет $-5.98{\cdot}10^{-15}$, и погрешность вычисления π уменьшается до $-7.18{\cdot}10^{-17}$. Второе слагаемое равно $3.12{\cdot}10^{-29}$. Это и последующие слагаемые выталкиваются за пределы удвоенной разрядной сетки и на результат не влияют.

К сожалению, рассмотренный пример является редким исключением. Очень многие ряды (в особенности при больших по модулю значениях аргумента) обладают крайне медленной сходимостью, и для достижения требуемой точности приходится брать большое число членов. В таких случаях применяются различные способы ускорения сходимости рядов, из которых мы рассмотрим уменьшение модуля аргумента и метод Эйлера—Абеля.

Уменьшение модуля аргумента базируется на специальных свойствах вычисляемой функции. Обозначим через $E(x)$ наибольшее целое число, не превосходящее x, и положим $\tilde{x} = x - E(x)$. Тогда для $x > 0$

$$e^x = e^{E(x)}e^{\tilde{x}} = \underbrace{e \cdot e \cdot e \ldots e}_{E(x)\text{ раз}} \cdot e^{\tilde{x}},$$

а при $x < 0$

$$e^x = \underbrace{e^{-1} \cdot e^{-1} \cdot e^{-1}}_{-E(x)\text{ раз}} \cdot e^{\tilde{x}}.$$

В обоих случаях $0 < \tilde{x} < 1$, и степенной ряд, обеспечивающий расчет последнего сомножителя, быстро сходится. Первая группа сомножителей вычисляется прямым умножением.

Попутно укажем способ экономии числа умножений. Пусть, например, оказалось $E(x) = 11$. Тогда $e^{11} = e^1 \cdot (e^2) \cdot ((e^2)^2)^2$. Записав двоичное представление показателя $(11)_{10} = (1011)_2$, убеждаемся, что расстановка нулей и единиц в нем указывает необходимый подбор сомножителей. В данном случае экономия может показаться несущественн-

ной. Однако указанная идея может быть (и была) использована при возведении в целую степень матриц и выполнении сверток распределений в моментах (для задачи о регулярном прореживании потоков).

При вычислении тригонометрических функций выполняется приведение к функции острого угла. На первом этапе вычисляется

$$x^* = \begin{cases} E(x/2\pi) & \text{для } \sin x, \quad \cos x; \\ E(x/\pi) & \text{для } \operatorname{tan} x, \quad \operatorname{ctg} x. \end{cases}$$

Далее тригонометрическая функция остатка $\beta = x - 2\pi x^*$ для синуса и косинуса, $\beta = x - \pi x^*$ для тангенса приводится к функции острого угла по формулам из табл. 5.1. Например, $\cos(\pi/2 + \alpha) = -\sin\alpha$, а $\cos(\pi/2 - \alpha) = \sin\alpha$.

Таблица 5.1. Приведение тригонометрических функций к функции острого угла

Функция	$\beta = \pi/2 \pm \alpha$	$\beta = \pi \pm \alpha$	$\beta = 3\pi/2 \pm \alpha$	$\beta = 2\pi - \alpha$
$\sin\beta$	$+\cos\alpha$	$\mp\sin\alpha$	$-\cos\alpha$	$-\sin\alpha$
$\cos\beta$	$\mp\sin\alpha$	$-\cos\alpha$	$\pm\sin\alpha$	$+\cos\alpha$
$\operatorname{tg}\beta$	$\mp\operatorname{ctg}\alpha$	$\pm\operatorname{tg}\alpha$	$\mp\operatorname{ctg}\alpha$	$-\operatorname{tg}\alpha$
$\operatorname{ctg}\beta$	$\mp\operatorname{tg}\alpha$	$\pm\operatorname{ctg}\alpha$	$\mp\operatorname{tan}\alpha$	$-\operatorname{ctg}\alpha$

Повторное применение формул первого столбца этой таблицы позволяет свести аргумент к диапазону $0 \le \alpha \le \pi/4$. Так, для $0 < \alpha < \pi/4$ имеем $\cos(\pi/4 + \alpha) = \sin(\pi/2 - (\pi/4 + \alpha)) = \sin(\pi/4 - \alpha)$.

Уменьшение аргумента является важным условием не только ускорения сходимости, но и (в некоторых случаях) получения осмысленных результатов работы. Приведем (табл. 5.2) взятые из работы [6] результаты вычисления $\sin x$ для $x = \pi/6 + 2k\pi, \quad k = 0, 1, \ldots$ с помощью степенного ряда $\sin x = x - x^3/3! + x^5/5! - \ldots$ Расчет членов ряда выполнялся с 8 десятичными знаками до тех пор, пока модуль очередного члена не оказывался меньше 10^{-8}. Точное значение синуса равно 0.5.

Таблица 5.2. Расчет $\sin x$ с помощью степенного ряда

x в градусах	x в радианах	Сумма ряда
30	0.52359878	0.49999999
390	6.80678415	0.49999993
750	13.08996952	0.50013507
1110	19.37315488	0.51658490
1470	25.65634012	24.25401855
1830	31.93952560	14380.23767090

При $x \geq 1470^\circ$ они кажутся совершенно неожиданными. Рассмотрим этот случай более подробно. Можно убедиться, что здесь наибольший по модулю член ряда b=5503768000, причем два последних нуля соответствуют потерянным из-за ограниченной длины мантиссы значащим цифрам. Ошибка в вычислении этого члена может достигать половины единицы младшего разряда мантиссы, т. е. $0.5 \cdot 100 = 50$. При таких громадных ошибках трудно рассчитывать на хороший окончательный результат, если он по модулю не должен превосходить единицы. Это и объясняет содержимое табл. 5.2.

Метод Эйлера—Абеля. Рассмотрим степенной ряд

$$S(x) = \sum_{n=0}^{\infty} a_n x^n \tag{5.3.5}$$

с конечным радиусом сходимости. Предположим, что существует предел

$$\rho = \lim_{n\to\infty} (a_{n+1}/a_n). \tag{5.3.6}$$

Тогда

$$\lim_{n\to\infty} \frac{a_{n+1} - \rho a_n}{a_n} = \lim_{n\to\infty} \left(\frac{a_{n+1}}{a_n} - \rho \right) = \rho - \rho = 0. \tag{5.3.7}$$

Умножив ряд (5.3.5) на $1 - \rho x$, получим

$$\begin{aligned} (1-\rho x)S(x) &= \sum_{n=0}^{\infty} a_n x^n - \sum_{n=0}^{\infty} \rho a_n x^{n+1} = a_0 + \sum_{n=0}^{\infty} a_{n+1} x^{n+1} - \sum_{n=0}^{\infty} \rho a_n x^{n+1} \\ &= a_0 + \sum_{n=0}^{\infty} (a_{n+1} - \rho a_n) x^{n+1} = a_0 + x \sum_{n=0}^{\infty} (a_{n+1} - \rho a_n) x^n \end{aligned}$$

и, следовательно,

$$S(x) = \frac{a_0}{1-\rho x} + \frac{x}{1-\rho x} \sum_{n=0}^{\infty} a_n \left(\frac{a_{n+1}}{a_n} - \rho \right) x^n, \tag{5.3.8}$$

причем в силу (5.3.7) последний ряд сходится быстрее ряда (5.3.5).

Подобное преобразование может быть применено повторно. В частном случае $\rho = 1$ имеем $a_{n+1} - \rho a_n = a_{n+1} - a_n = \Delta a_n$, и (5.3.8) переходит в

$$S(x) = \frac{a_0}{1-\rho x} + \frac{x}{1-\rho x} \sum_{n=0}^{\infty} (\Delta a_n) x^n,$$

т. е. коэффициенты нового ряда суть конечные разности коэффициентов исходного ряда.

Пример 5.1. Известно, что

$$\ln(1-x) = -\left(x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots + \frac{x^n}{n} + \ldots\right), \tag{5.3.9}$$

и этот ряд сходится (при $-1 \le x < 1$) чрезвычайно медленно. В частности, для получения погрешности не более 10^{-4} при $x = -1$ требуется взять 10 000 членов ряда (5.3.9). Преобразуем этот ряд к виду

$$\ln(1-x) = -x\left(1 + \frac{x}{2} + \frac{x^2}{3} + \ldots + \frac{x^n}{n+1} + \ldots\right)$$

и рассмотрим функцию

$$\varphi(x) = 1 + \frac{x}{2} + \frac{x^2}{3} + \ldots + \frac{x^n}{n+1} + \ldots$$

Легко видеть, что здесь $\rho = 1$. Вычислим для ряда (5.3.9) разность

$$a_{n+1} - \rho a_n = a_{n+1} - a_n = \frac{1}{n+2} - \frac{1}{n+1} = \frac{n+1-n-2}{(n+1)(n+2)} = -\frac{1}{(n+1)(n+2)}.$$

Следовательно, однократное преобразование Эйлера—Абеля приводит ряд (5.3.9) к виду

$$\ln(1-x) = -x\left(\frac{1}{1-x} - \frac{x}{1-x}\sum_{n=0}^{\infty}\frac{x^n}{(n+1)(n+2)}\right). \tag{5.3.10}$$

Для членов полученной суммы вновь имеем $\rho = 1$, и

$$a'_{n+1} - \rho a'_n = \frac{1}{(n+2)(n+3)} - \frac{1}{(n+1)(n+2)} = -\frac{2}{(n+1)(n+2)(n+3)},$$

откуда

$$\ln(1-x) = \frac{-x}{1-x} + \frac{x^2}{1-x}\left[\frac{1}{2(1-x)} - \frac{x}{1-x}\sum_{n=0}^{\infty}\frac{x^n}{(n+1)(n+2)(n+3)}\right]. \tag{5.3.11}$$

При отрицательном x (напомним, что расчет ведется для $x = -1$) члены этих рядов имеют чередующиеся знаки; следовательно, для получения требуемой точности должно выполняться условие $|b_{n*+1}| < \varepsilon$. Здесь

через $\{b_n\}$ обозначены члены преобразованного ряда, а n^* есть граница усечения его. При этом необходимо учесть множитель перед суммой, равный $1/2$ в выражении (5.3.10) и $1/4$ в (5.3.11). Итак, для (5.3.10) при $\varepsilon = 10^{-4}$ необходимо потребовать

$$\frac{1}{2}\frac{1}{(n^*+2)(n^*+3)} \le 10^{-4},$$

или $(n^*+2)(n^*+3) \ge 5000$, откуда $n^* \ge 68$.

Для ряда (5.3.11) получаем условие

$$(n^*+2)(n^*+3)(n^*+4) \ge 2500,$$

или $n^* \ge 11$. После трехкратного преобразования Эйлера—Абеля достаточно потребовать

$$(n^*+2)(n^*+3)(n^*+4)(n^*+5) \ge 1250,$$

откуда $n^* \ge 3$. Таким образом, преобразование Эйлера—Абеля в данном случае оказалось весьма эффективным.

5.4. Дробно-рациональные приближения

Дробно-рациональная *аппроксимация Паде* записывается в виде

$$f(x) = \frac{b_0 + b_1 x + b_2 x^2 + b_3 x^3}{1 + c_1 x + c_2 x^2 + c_3 x^3}.$$

Для определенности и упрощения последующих рассуждений мы выбрали составляющие полиномы третьей степени. Ясно также, что выбор $c_0 = 1$ общности не уменьшает. Расчет этих полиномов опять же организуется по схеме Горнера и (при равных степенях) внутри *одного* цикла.

Аппроксимация Паде часто более эффективна, чем разложение Тейлора, и имеет бóльшую область сходимости. Она особенно полезна в диапазонах «немногочленного» поведения функции — например, асимптотического. Покажем, как получить такую аппроксимацию. Заменим $f(x)$ ее многочленным приближением (например, разложением в ряд Тейлора) с коэффициентами $\{a_j\}$ и учетом степеней до 6-й включительно. Тогда

$$b_0+b_1x+b_2x^2+b_3x^3 = (1+c_1x+c_2x^2+c_3x^3)\cdot(a_0+a_1x+a_2x^2+a_3x^3+a_4x^4+a_5x^5+a_6x^6).$$

Перемножив многочлены, из условия равенства коэффициентов при одинаковых степенях x в левой и правой частях уравнения имеем систему

$$\begin{aligned} b_0 &= a_0, \\ b_1 &= a_1 + a_0c_1, \\ b_2 &= a_2 + a_1c_1 + a_0c_2, \\ b_3 &= a_3 + a_2c_1 + a_1c_2 + a_0c_3, \\ 0 &= a_4 + a_3c_1 + a_2c_2 + a_1c_3, \\ 0 &= a_5 + a_4c_1 + a_3c_2 + a_2c_3, \\ 0 &= a_6 + a_5c_1 + a_4c_2 + a_3c_3. \end{aligned} \qquad (5.4.1)$$

Решая эту систему, находим коэффициенты дробно-линейной аппроксимации $\{b_i\}$ и $\{c_i\}$.

5.5. Непрерывные дроби

Дробно-рациональную функцию легко представить в виде цепной дроби. Если

$$f(x) = \frac{c_{10} + c_{11}x + c_{12}x^2 + \dots}{c_{00} + c_{01}x + c_{02}x^2 + \dots},$$

то с помощью элементарных преобразований имеем

$$f(x) = \frac{1}{\dfrac{c_{00}}{c_{10}} + \dfrac{c_{00} + c_{01}x + c_{02}x^2 + \dots}{c_{10} + c_{11}x + c_{12}x^2 + \dots} - \dfrac{c_{00}}{c_{10}}} = \frac{c_{10}}{c_{00} + xf_1(x)},$$

где

$$\begin{aligned} f_1(x) &= \frac{c_{20} + c_{21}x + c_{22}x^2 + \dots}{c_{10} + c_{11}x + c_{12}x^2 + \dots}, \\ c_{2,k} &= c_{10}c_{0,k+1} - c_{00}c_{1,k+1}, \qquad k = 0, 1, \dots \end{aligned}$$

Аналогично

$$f_1(x) = \frac{c_{20}}{c_10 + xf_2(x)},$$

где

$$\begin{aligned} f_2(x) &= \frac{c_{30}+c_{31}x+c_{32}x^2+\ldots}{c_{20}+c_{21}x+c_{22}x^2+\ldots}, \\ c_{3,k} &= c_{20}c_{1,k+1}-c_{10}c_{2,k+1}, \qquad k=0,1,\ldots \end{aligned}$$

и т. д. Поскольку на каждом шаге наивысшая степень многочлена понижается на единицу, процесс завершится через конечное число шагов и приведет к цепной дроби

$$\cfrac{c_{10}}{c_{10}+\cfrac{c_{20}x}{c_{20}+\cfrac{c_{30}x}{c_{30}+\cfrac{c_{40}x}{c_{40}+\cdots}}}}$$

Правила расчета коэффициентов многочленов удобно задать определителем

$$c_{j,k}=-\begin{vmatrix} c_{j-2,0} & c_{j-2,k+1} \\ c_{j-1,0} & c_{j-1,k+1} \end{vmatrix}.$$

Указанным образом получено, например, разложение

$$\ln(1+x)= k_0+\cfrac{x}{k_1+\cfrac{x}{k_2+\cfrac{x}{k_3+\cfrac{x}{k_4+\cfrac{x}{k_5}}}}}.$$

Здесь для интервала $0\le x\le 1$ и

$$\begin{aligned} &k_0=0.0000000894, \quad k_1=1.0000091365, \quad k_2=2.0005859000, \\ &k_3=3.0311932666, \quad k_4=1.0787748225, \quad k_5=8.8952784060 \end{aligned}$$

абсолютная погрешность результата не превосходит 10^{-7} [99, с. 49].

Цепная дробь считается *снизу вверх* по схеме, аналогичной схеме Горнера:

```
p=k(5)
do i=4,0,-1
   p=k(i)+x/p
end do
```

Глава 6.
Решение уравнений с одним неизвестным

6.1. Постановка задачи

Пусть $f(x)$ — непрерывная функция, определенная в некотором промежутке $[a,b]$. Будем изучать способы, позволяющие найти приближенное значение корня ξ уравнения

$$f(x)=0. \tag{6.1.1}$$

В общем случае состояние процесса решения уравнения (6.3.1) на k-м шаге характеризуется тремя величинами:

- **невязка** $f(x_k)$;
- **ошибка** $(\xi - x_k)$;
- **поправка** $(x_k - x_{k-1})$.

В последующем предполагается, что функция $f(x)$ на $[a,b]$ знак меняет, а производная $f'(x)$ знак сохраняет. Из этого предположения следует существование единственного корня ξ уравнения (6.3.1) на $[a,b]$. Предположим также, что модуль производной функции $f(x)$ ограничен снизу и сверху положительными числами m_1 и M_1:

$$0 < m_1 \le |f'(x)| \le M_1 < \infty. \tag{6.1.2}$$

Пусть теперь $x_0 \in [a,b]$ — произвольное число. Покажем, что имеет место неравенство

$$|x_0 - \xi| \le \frac{|f(x_0)|}{m_1}. \tag{6.1.3}$$

Действительно, учитывая равенство $f(\xi) = 0$ и используя теорему Лагранжа, получаем

$$|f(x_0)| = |f(x_0) - f(\xi)| = |f'(c)(x_0 - \xi)| = |f'(c)||x_0 - \xi| \geq m_1|x_0 - \xi|,$$

т. е. $m_1|x_0 - \xi| \leq |f(x_0)|$, откуда следует (6.1.3).

Неравенство (6.1.3) дает оценку абсолютной погрешности приближенного равенства

$$\xi \approx x_0$$

через «невязку» $f(x_0)$.

Пусть x^* удовлетворяет уравнению $f(x^*) = 0$. Будем говорить, что $\bar{x}$ «решает» уравнение $f(x) = 0$, если выполнено одно из условий

$$|f(\bar{x})| \approx 0; \qquad |\bar{x} - x^*| \approx 0.$$

Недостатком первого подхода к оценке корня (по невязке левой части) является влияние на процесс масштабирования уравнения: умножение обеих частей его на достаточно малое по модулю число заставит процесс завершиться очень далеко от корня. Второй подход (по ошибке) для оценки погрешности точного решения требует знания искомого корня. Последнее затруднение снимается при немонотонном характере приближений (здесь погрешность мажорируется модулем поправки) или использовании аппроксимаций типа Δ^2-процесса Эйткена (см. разд. 6.8).

6.2. Отделение корней

Приближенное нахождение корней уравнения обычно складывается из двух этапов: отделение корней (установление возможно более тесных границ промежутков, в каждом из которых находится один и только один корень) и доведение каждого из приближенных значений до требуемой точности.

Одним из простых методов отделения корней является графический. Он удобен при выводе графиков на экран дисплея.

Для отделения корней обычно используется имеемая дополнительная информация (см. разд. 6.9 о решении полиномиальных уравнений), а также процесс последовательного деления на две части (бисекция, дихотомия) исходного отрезка и его «подозрительных» фрагментов с проверкой каждого такого фрагмента на перемену знака. Если непрерывная

функция $f(x)$ на границах отрезка $[a,b]$ принимает значения разных знаков, что проще всего проверяется произведением $f(a)f(b)<0$, то в этом отрезке содержится нечетное число корней уравнения $f(x)=0$ с учетом их кратностей (по крайней мере один). В противном случае корней либо нет, либо их число четно. Эта теорема (Больцано—Коши) в условиях теоремы не дает ответа на вопрос о количестве нулей и бесполезна при четном их количестве.

При тех же условиях непрерывная *строго монотонная* функция имеет на отрезке $[a,b]$ нуль, и притом единственный.

6.3. Метод половинного деления

Метод проб (он же — половинного деления) состоит в том, что сначала вычисляется $x_0=(a+b)/2$. Затем определяют $f(x_0)$ и — в зависимости от знака результата — устанавливают, в какой части прежнего интервала ($[a,x_0]$ или $[x_0,b]$) находится корень. К новому интервалу применяется та же процедура — до тех пор, пока не обнаружится $f(x)\approx 0$ (с точностью, допускаемой условиями задачи или возможностями машины), или длина промежутка, в котором заключен корень ξ, не окажется меньше допустимой погрешности. На каждой итерации метода половинного деления приобретается один бит точности.

Достоинством метода проб является то, что он обязательно приводит к цели (процесс сходится). При этом всегда

$$|\xi-x_n|\le (b-a)/2^n. \tag{6.3.1}$$

Неравенство (6.3.1) позволяет априорно оценить число шагов n, достаточное для достижения $|\xi-x_n|\le\Delta$. Из условия $\Delta\le(b-a)/2^n$ находим $2^n\le(b-a)/\Delta$. Логарифмируя это неравенство по основанию 2, получаем

$$n\le\log_2\frac{b-a}{\Delta}.$$

6.4. Метод хорд

Пусть для определенности $f(a) < 0$ и $f(b) > 0$. Логично предположить, что в классе методов дихотомии можно достичь лучших результатов, если делить область локализации корня не пополам, а в отношении $-f(a) : f(b)$. Это даст нам приближенное значение корня

$$x_1 = a + h_1,$$

где

$$h_1 = \frac{-f(a)}{-f(a) + f(b)}(b - a) = -\frac{f(a)}{f(b) - f(a)}(b - a).$$

Таким образом,

$$x_1 = a - \frac{f(a)}{f(b) - f(a)}(b - a).$$

Далее описанный метод применяется к тому из интервалов $[a, x_1]$, $[x_1, b]$, на котором функция $f(x)$ меняет знак. Процесс повторяется, пока корень ξ не будет получен с требуемой точностью. При постоянстве знака $f'(x)$ на интервале $[a, b]$ метод хорд обеспечивает монотонное стремление результатов очередного шага x_n к корню ξ. В частности, при $f'(x)f''(x) > 0$, как на рис. 6.1,а),

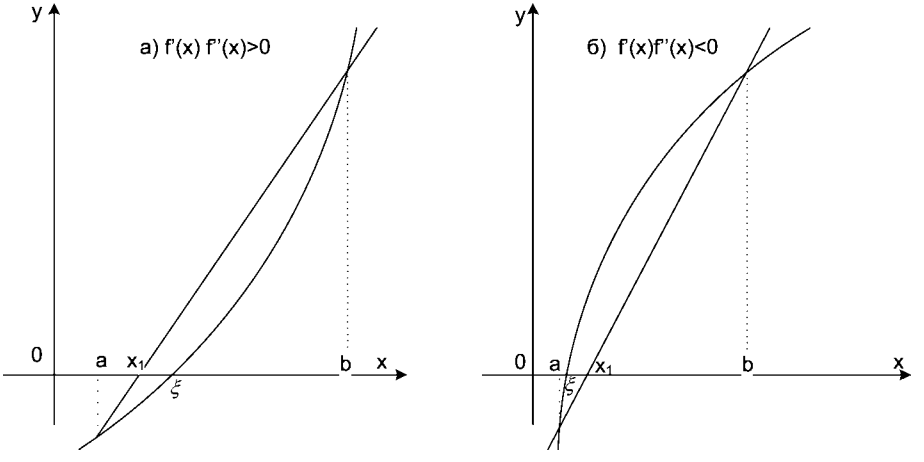

Рис. 6.1. Решение уравнения методом хорд

последовательные приближения будут получаться по формуле

$$x_{n+1} = x_n - \frac{f(x_n)}{f(b) - f(x_n)}(b - x_n), \qquad n = 1, 2, \ldots \tag{6.4.1}$$

(двигается левый конец интервала). При $f'(x)f''(x) < 0$ — рис. 6.1 б) — двигается правый конец интервала, и

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(a)}(x_n - a), \qquad n = 1, 2, \ldots \tag{6.4.2}$$

Найдем оценку погрешности через разность двух последовательных приближений. Примем для определенности, что последовательные приближения вырабатываются согласно формуле (6.4.1). Тогда

$$a < x_1 < x_2 < \ldots < x_{n-1} < x_n < \ldots < \xi < b$$

и

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b) - f(x_{n-1})}(b - x_{n-1}).$$

Это равенство может быть преобразовано к виду

$$f(x_{n-1}) = \frac{f(b) - f(x_{n-1})}{b - x_{n-1}}(x_{n-1} - x_n) = f'(\tilde{x}_n)(x_{n-1} - x_n),$$

где $\tilde{x}_n \in [x_{n-1}, b]$. Следовательно,

$$f(x_{n-1}) = f'(\tilde{x}_n)(x_{n-1} - x_n).$$

С другой стороны,

$$\begin{aligned} f(x_{n-1}) &= -f(\xi) + f(x_{n-1}) = f'(c_n)(x_{n-1} - \xi) \\ &= f'(c_n)(x_{n-1} - x_n + x_n - \xi) \\ &= f'(c_n)(x_{n-1} - x_n) + f'(c_n)(x_n - \xi), \end{aligned}$$

где $c_n \in [x_{n-1}, \xi]$.

Сопоставив полученные результаты, будем иметь

$$f'(\tilde{x}_n)(x_{n-1} - x_n) = f'(c_n)(x_n - \xi) + f'(c_n)(x_{n-1} - x_n),$$

откуда

$$|\xi - x_n| = \frac{|f'(\tilde{x}_n) - f'(c_n)|}{|f'(c_n)|}|x_n - x_{n-1}|.$$

Поскольку производная $f'(x)$ на $[a, b]$ по предположению не меняет знака, можно утверждать, что $|f'(\tilde{x}_n) - f'(c_n)| \le M_1 - m_1$, и поэтому

$$|\xi - x_n| \le \frac{M_1 - m_1}{m_1}|x_n - x_{n-1}|. \tag{6.4.3}$$

Этот же результат получается и в случае приближения к корню справа, т. е. согласно (6.4.2).

Если интервал $[a, b]$ столь узок, что $M_1 \leq 2m_1$, то

$$|\xi - x_n| \leq |x_n - x_{n-1}|.$$

В этом случае вычисления можно прекратить при $|\xi - x_n| \leq \Delta$, где Δ есть допустимая абсолютная погрешность вычисления корня уравнения. Она должна выбираться с учетом абсолютной величины корня — из условия получения требуемого количества верных цифр.

Метод хорд, как правило, сходится быстрее метода проб, причем в обоих случаях успех гарантирован. Возможны, однако, случаи, когда половинное деление обеспечит более быструю сходимость — см. рис. 6.2.

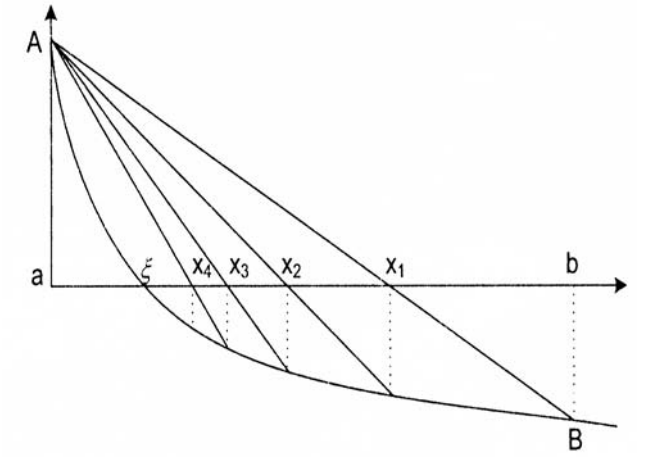

Рис. 6.2. Плохая сходимость метода хорд

Эффективна *комбинация* бисекции и половинного деления: бисекция рекомендуется, если полученная точка оказалась внутри текущего интервала и не слишком близко к его концам.

6.5. Параболическая аппроксимация

Данный подход (метод Мюллера) обобщает метод секущих с двух интерполяционных точек на три. В общем случае имеются три точки $\{x_{n-2}, x_{n-1}, x_n\}$ и соответствующие им значения функции $\{y_{n-2}, y_{n-1}, y_n\}$.

По ним составляется квадратичная функция $z(x)$, и в качестве x_{n+1} берется тот из нулей $z(x)$, который ближе к x_n. Далее операции повторяются с тремя последними точками.

Итак, мы заменяем уравнение $f(x)=0$ на

$$Ax^2+Bx+C=0. \tag{6.5.1}$$

Левой границей локализации корня считается x_0, правой — x_2, точка x_1 сначала помещается в середине интервала. Обозначив $y_i=f(x_i)$, имеем для нахождения коэффициентов аппроксимации уравнения

$$\begin{aligned} A(x_0^2-x_1^2)+B(x_0-x_1)&=y_0-y_1,\\ A(x_2^2-x_1^2)+B(x_2-x_1)&=y_2-y_1. \end{aligned}$$

Частные определители этой системы

$$\begin{aligned} \Delta_A &= \begin{vmatrix} y_0-y_1 & x_0-x_1 \\ y_2-y_1 & x_2-x_1 \end{vmatrix} = (y_0-y_1)(x_2-x_1)-(y_2-y_1)(x_0-x_1),\\ \Delta_B &= \begin{vmatrix} x_0^2-x_1^2 & y_0-y_1 \\ x_2^2-x_1^2 & y_2-y_1 \end{vmatrix} = (y_2-y_1)(x_0^2-x_1^2)-(y_0-y_1)(x_2^2-x_1^2), \end{aligned}$$

а главный определитель

$$\Delta=(x_0^2-x_1^2)(x_2-x_1)-(x_2^2-x_1^2)(x_0-x_1).$$

Соответственно,

$$A=\Delta_A/\Delta,\quad B=\Delta_B/\Delta,\quad C=y_2-(Ax_2+B)x_2.$$

Теперь можно найти корни уравнения (6.5.1); выбрать x_3 — тот из них, который находится внутри интервала $[x_0,x_2]$; вычислить $y_3=f(x_3)$. Далее x_3 в зависимости от знака y_3 становится левой или правой границей интервала. Процесс прекращается при $|x_1-x_3|<\varepsilon_1$ или $|f(x_3)|<\varepsilon_2$. Приведем соответствующий фрагмент программы:

```
y0=f(x0); y2=f(x2)
dx=abs(x1-x2); y3=1e0
do while(abs(dx)>eps1 .and. abs(y3)>eps2)
   da=(y0-y1)*(x2-x1)-(y2-y1)*(x0-x1)
   db=(y2-y1)*(x0*x0-x1*x1)-(y0-y1)*(x2*x2-x1*x1)
   d=(x0*x0-x1*x1)(x2-x1)-(x2*x2-x1*x1)*(x0-x1)
   a=da/d; b=db/d; c=y2-(a*x2+b)*x2
   d=b**2-4*a*c; d=sqrt(d); a=a/2
```

```
      x3=(b+d)/a; x4=(-b-d)/a
      if (abs(x1-x4)<(abs(x1-x3)) x3=x4
      dx=abs(x1-x3); y3=f(x3)
      if (y3*y1<0) then
         if (x3<x1) then
            x0=x3; y0=y3
         else
            x2=x3; y2=y3
         end if
      else
         if (y3*y2)>0) then
            x2=x1; x1=x3; y2=y1; y1=y3
         else
            x0=x1; x1=x3; y0=y1; y1=y3
         end if
      end if
   end do
```

В дальнейшем на каждом шаге процесса значение функции вычисляется только один раз — в точке x'. Эта точка заменяет собой одну из исходных при условии сохранения перемены знака. Соответственно заменяется и значение функции. Процесс продолжается до получения решения с требуемой точностью.

В случае сходимости метода старший коэффициент $z(x) \to 0$, и в ближайшей окрестности корня необходим переход к линейной интерполяции.

Та же аппроксимация может быть применена для приближенного нахождения *минимума* функции. При $A < 0$ минимум «прямой» параболы соответствует точке $x^* = -B/2A$. Итак,

$$x^* = \frac{(y_0 - y_1)(x_2^2 - x_1^2) - (y_2 - y_1)(x_0^2 - x_1^2)}{2[(y_0 - y_1)(x_2 - x_1) - (y_2 - y_1)(x_0 - x_1)]}.$$

Этот подход очень полезен при определении минимума по направлению в методе скорейшего спуска — см. разд. 11.4.3.

При необходимости получения новых корней переходят к функции $f_1(x) = f(x)/(x - x^*)$, где x^* — найденный корень.

Другим вариантом нахождения корня является переход к обратной функции. Здесь нужно поменять ролями x и y и подставить $y = 0$. Естественно, значение аппроксимирующей параболы будет равно ее свободному члену.

6.6. Метод Ньютона

Пусть корень ξ уравнения

$$f(x) = 0 \tag{6.6.1}$$

отделен на отрезке $[a, b]$, причем $f'(x)$ и $f''(x)$ непрерывны и сохраняют свои знаки при $a \le x \le b$. Найдя какое-либо n-е приближение к корню, мы можем уточнить его следующим образом. Положим

$$\xi = x_n + h_n,$$

где h_n — малая поправка. Из условия

$$0 = f(x_n + h_n) \approx f(x_n) + h_n f'(x_n)$$

находим поправку

$$h_n = -f(x_n)/f'(x_n).$$

Таким образом,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, \ldots$$

На рис. 6.3 приведена графическая иллюстрация метода Ньютона.

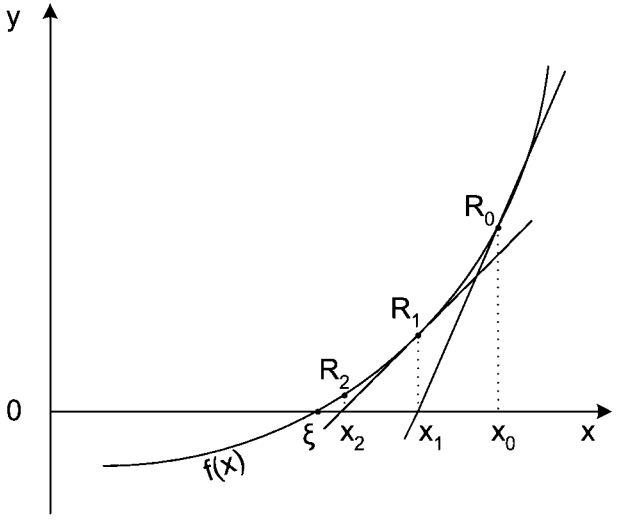

Рис. 6.3. Метод Ньютона

Сходимость метода Ньютона к корню уравнения будет тем быстрее, чем ближе функция к линейной и чем круче ее график пересекает ось абсцисс. Имеет смысл требовать, чтобы по модулю вторая производная была ограничена сверху, а первая снизу.

При затруднениях с вычислением производной последнюю заменяют конечно-разностным отношением:

$$x_{k+1} = x_k - \frac{f(x_k) \cdot h}{f(x_k + h) - f(x_k)}.$$

В непосредственной окрестности корня имеет смысл принять $h = f(x_k)$. Тогда последняя формула переходит в

$$x_{k+1} = x_k - \frac{f^2(x_k)}{f(x_k + f(x_k)) - f(x_k)}$$

(метод Стеффенсена).

С вычислительной точки зрения более экономен метод, в котором для оценки производной используется $f'(x_k) = (f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})$. Получаем *двухшаговый* метод секущих

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}.$$

При его реализации на каждом шаге вычисляется только одно новое значение функции.

Перейдем к оценкам точности результатов, получаемых методом Ньютона.

Теорема 6.1. Пусть функция $f(x)$ для всех $x \in [a, b]$ удовлетворяет условиям

$$\begin{aligned} |f'(x)| &\geq m_1 > 0, \\ |f''(x)| &\geq m_2 > 0. \end{aligned}$$

Если все последовательные приближения $\{x_k\}$, получаемые методом Ньютона, при любом фиксированном k принадлежат к отрезку $[a, b]$ и сходятся к корню ξ уравнения (6.6.1), то справедливы неравенства

$$\begin{aligned} |\xi - x_{k+1}| &\leq \frac{m_2}{2m_1}|\xi - x_k|^2, \\ |\xi - x_{k+1}| &\leq \frac{m_2}{2m_1}|x_{k+1} - x_k|^2. \end{aligned} \qquad (6.6.2)$$

Первое из этих неравенств позволяет считать метод Ньютона по скорости убывания невязки методом *второго* порядка, а второе может служить для остановки процесса вычислений. Доказательство см. [23, с. 127].

Начальное приближение следует выбирать из условия Фурье

$$f(x_0)f''(x_0) > 0.$$

Это гарантирует принадлежность к упомянутому интервалу последующих приближений.

Для оценки погрешности n-го приближения можно воспользоваться общей формулой

$$|\xi - x_n| \le \frac{|f(x_n)|}{m_1}, \tag{6.6.3}$$

где m_1 — наименьшее значение $|f'(x)|$ на $[a,b]$. С другой стороны, по формуле Тейлора

$$f(x_n) = f(x_{n-1}) + f'(x_{n-1})(x_n - x_{n-1}) + \frac{1}{2}f''(\xi)(x_n - x_{n-1})^2, \tag{6.6.4}$$

где $\xi_{n-1} \in (x_{n-1}, x_n)$. С учетом правила определения поправки для перехода от x_{n-1} к x_n формула (6.6.4) переходит в

$$|f(x_n)| \le \frac{1}{2}M_2(x_n - x_{n-1})^2.$$

Подставляя эту оценку в (6.6.3), окончательно имеем

$$|\xi - x_n| \le \frac{M_2}{2m_1}(\xi_n - x_{n-1})^2. \tag{6.6.5}$$

Эта формула обещает быструю сходимость процесса Ньютона, если начальное приближение x_0 таково, что

$$\frac{M_2}{2m_1}|\xi - x_0| \le q < 1.$$

В частности, если $M_2/(2m_1) \le 1$ и $|\xi - x_n| < 10^{-m}$, то оценка погрешности следующего приближения будет меньше 10^{-2m}. Фактически это означает, что *в окрестности корня* на каждой итерации погрешность возводится в квадрат, т. е. число верных знаков удваивается. Но в эту окрестность еще надо попасть; поэтому на практике для выхода в непосредственную близость к корню применяют гарантирующий сходимость медленный метод, а затем переходят на метод Ньютона.

Меры предосторожности при использовании метода Ньютона основаны на двух идеях:

- нужна гарантия приближения к решению на каждом шаге;
- не следует делать слишком больших шагов.

Вычисленный шаг принимается, если норма невязки уменьшается. Иначе шаг сокращается.

Недостатки метода Ньютона:

- может иметь малую область сходимости и потому требует хороших начальных приближений;
- требует вычисления не только значений функции, но и ее производных.

Последнего можно избежать, аппроксимируя производную конечно-разностным отношением.

В *модифицированном* методе Ньютона первоначальное значение производной не перевычисляется. Это уменьшает трудоемкость каждого шага, но увеличивает число шагов.

6.7. Метод итераций

6.7.1. Понятие о методе итераций

Выпишем вновь исходное уравнение

$$f(x) = 0. \tag{6.7.1}$$

Пусть оно имеет равносильную форму

$$x = \varphi(x), \tag{6.7.2}$$

где $\varphi(x)$ — непрерывная функция, и x_0 — некоторое число, которое назовем начальным приближением к корню уравнения (6.7.2). Пользуясь формулой

$$x_n = \varphi(x_{n-1}), \qquad n = 1, 2, \ldots, \tag{6.7.3}$$

будем находить числа $x_1, x_2, \ldots, x_n, \ldots$ Если эта последовательность при $n \to \infty$ имеет конечный предел $\xi = \lim_{n\to\infty} x_n$, то ξ является корнем уравнения (6.7.2). При нахождении ξ указанным приемом говорят, что

уравнение (6.7.2) решается методом *итерации*. Итерацией называют как шаг метода, так и полученный на нем результат.

Рассмотрим геометрическую иллюстрацию метода итераций. Построим графики $y = x$ и $y = \varphi(x)$, показанные на рис. 6.4, а).

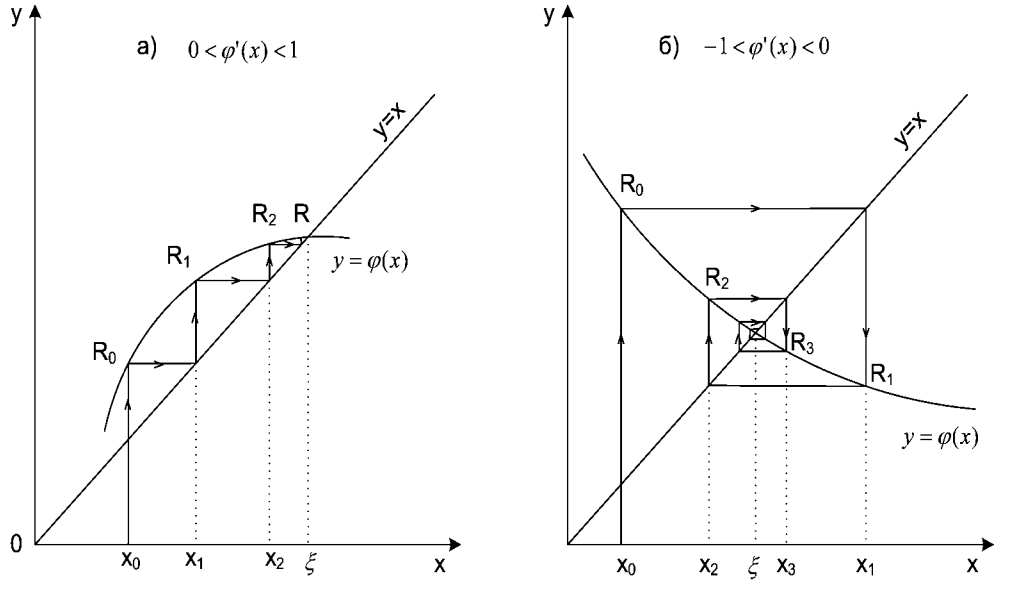

Рис. 6.4. Сходящийся итерационный процесс решения уравнения $x = \varphi(x)$

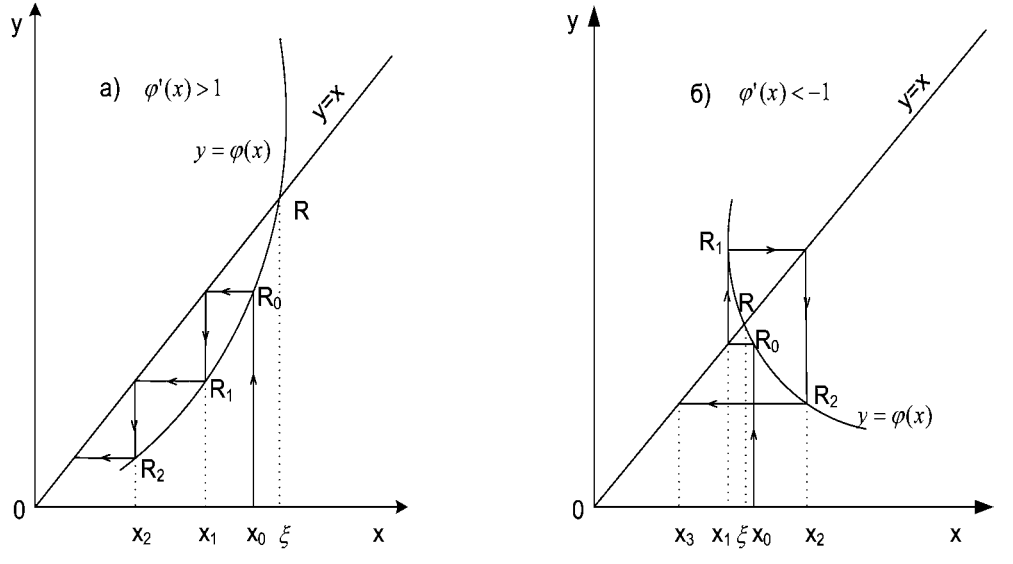

Рис. 6.5. Расходящийся итерационный процесс решения уравнения $x = \varphi(x)$

Абсцисса ξ точки R пересечения этих графиков является корнем уравнения (6.7.2). Пусть x_0 — произвольное число. Тогда $\varphi(x_0)$ есть ордината точки R_0, и на графике точка R'_0 имеет ту же ординату и абсциссу x_1, равную $f(x_0)$. Продолжая этот процесс, видим, как точки $R_0, R_1, R_2, \ldots$ неограниченно приближаются к R, а их абсциссы — к корню уравнения ξ.

Рассмотренный рис. 6.4, а) соответствует $0 < \varphi'(x) < 1$. При $-1 < \varphi'(x) < 0$ приближения чередуются, подходя к корню то справа, то слева — рис. 6.4, б). Если модуль производной превышает единицу, то итерации оказываются *расходящимися* — см. рис. 6.5, а) для положительной производной и рис. 6.5, б) для отрицательной.

6.7.2. Условия сходимости итераций

Условия сходимости итераций могут быть установлены аналитически. Имеет место

Теорема 6.2. Если $\varphi(x)$ определена и дифференцируема на всей числовой оси и существует число $q, \quad 0 < q < 1$, такое, что для всех x модуль производной $|\varphi'(x)| \le q$, то уравнение (6.7.2) имеет единственный корень, и процесс итерации сходится к этому корню независимо от выбора начального приближения x_0.

Доказательство. Построим в соответствии с равенством (6.7.3), исходя из x_0, последовательность чисел $x_1, x_2, \ldots, x_n = \varphi(x_{n-1})$. Тогда на основании теоремы Лагранжа

$$x_{n+1} - x_n = \varphi(x_n) - \varphi(x_{n-1}) = \varphi'(c_n)(x_n - x_{n-1})$$

и, следовательно,

$$|x_{n+1} - x_n| = |\varphi'(c_n)| \cdot |x_n - x_{n-1}| \le q|x_n - x_{n-1}|.$$

Значит,

$$\begin{aligned} |x_2 - x_1| &\le q|x_1 - x_0|, \\ |x_3 - x_2| &\le q|x_2 - x_1| \le q^2|x_1 - x_0|, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ |x_{n+1} - x_n| &\le q^n|x_1 - x_0|. \end{aligned}$$

Отсюда следует, что ряд

$$x_0 + (x_1 - x_0) + (x_2 - x_1) + \ldots + (x_n - x_{n-1}) + \ldots \qquad (6.7.4)$$

сходится абсолютно, так как его члены по модулю не превосходят членов убывающей геометрической прогрессии. Обозначим предел частной суммы ряда (6.7.4) через ξ. Но частная сумма этого ряда есть x_n. Следовательно, $\lim_{n\to\infty} x_n = \xi$, т. е. корню уравнения (6.7.2). Легко установить и единственность получаемого корня.

Аналогичным образом можно установить, что при $|\varphi'(x)| \geq 1$ на всей числовой оси точки $\{x_n\}$ с ростом n не будут приближаться к корню ξ (если только в начальный момент не оказалось $x_0 = \xi$). ▲

Обычно с уравнением (6.7.2) имеют дело не на всей числовой оси, а на некотором конечном отрезке $[a, b]$. В этом случае предъявляется дополнительное требование, чтобы точки $x_n = \varphi(x_{n-1})$, $n = 1, 2, \ldots$, не покидали $[a, b]$.

Процесс итераций в условиях теоремы является *самоисправляющимся*: ошибка в одной из итераций (если результат не вышел из области сходимости) может рассматриваться как задание нового начального приближения. Таким образом, ошибки накапливаются только в пределах одного шага итераций.

Оценим погрешность результата очередной итерации. Пусть на отрезке $[a, b]$ модуль производной правой части $|\varphi'(x)| \leq q < 1$ и существует корень ξ уравнения (3.3.2). Тогда

$$\begin{aligned} |x_n - \varphi(x_n)| &= |(x_n - \xi) - [\varphi(x_n) - \varphi(\xi)]| = |(x_n - \xi) - (x_n - \xi)\varphi'(c)| \\ &= |x_n - \xi| \cdot |1 - \varphi'(c)| \geq |x_n - \xi| \begin{cases} 1 - q & \text{при } \varphi'(x) > 0; \\ 1 & \text{при } \varphi'(x) \leq 0 \end{cases} \end{aligned}$$

и, следовательно,

$$|x_n - \xi| \leq |x_n - \varphi(x_n)| \begin{cases} 1/(1-q) & \text{при } \varphi'(x) > 0; \\ 1 & \text{при } \varphi'(x) \leq 0. \end{cases}$$

С другой стороны,

$$\begin{aligned} |x_n - \varphi(x_n)| &= |\varphi(x_{n-1}) - \varphi(x_n)| = |\varphi'(c_1)(x_{n-1} - x_n)| \\ &= |\varphi'(c_1)| \cdot |x_{n-1} - x_n| \leq q|x_n - x_{n-1}|. \end{aligned}$$

Отсюда следует, что

$$|x_n - \xi| \leq |x_n - x_{n-1}| \begin{cases} q/(1-q) & \text{при } \varphi'(x) > 0; \\ q & \text{при } \varphi'(x) \leq 0. \end{cases} \qquad (6.7.5)$$

Ранее было доказано, что $|x_n - x_{n-1}| \leq q^{n-1}|x_1 - x_0|$. Значит,

$$|x_n - \xi| \leq q^n |x_1 - x_0| \begin{cases} 1/(1-q) & \text{при } \varphi'(x) > 0; \\ 1 & \text{при } \varphi'(x) \leq 0. \end{cases} \qquad (6.7.6)$$

Формулы (6.7.5) и (6.7.6) позволяют оценить близость к корню ξ n-го приближения x_n.

В практике вычислений по итерационным зависимостям часто считают: если два последовательных приближения x_{n-1} и x_n совпали между собой с заданной точностью ε, то с той же точностью справедливо равенство $\xi \approx x_n$. Как видно из формулы (6.7.5), такой подход допустим при $\varphi'(x) < 0$. В случае $\varphi'(x) > 0$ им можно пользоваться только при $q/(1-q) \le 1$, т. е. когда $q \le 1/2$. Если же $\varphi'(x) > 1/2$, то при q, близком к единице, ошибка от прекращения вычислений на n-й итерации может быть недопустимо велика независимо от модуля разности между смежными приближениями. В случае $q > 1/2$ вместо метода итераций рекомендуется применять метод половинного деления.

Приведенные выше теоремы о сходимости итераций доказывались в предположении, что вычисления $x_{n+1} = \varphi(x_n)$ выполняются абсолютно точно. В действительности счет ведется с определенным числом значащих цифр, и фактический процесс порождает последовательность чисел

$$\tilde{x}_{n+1} = \varphi(\tilde{x}_n) + \gamma_n, \qquad n = 0, 1, \ldots,$$

где $\{\gamma_n\}$ — ошибки округления (усечения). Последовательность $\{\tilde{x}_n\}$ сходится к корню уравнения лишь при условии, что погрешности $\{\gamma_n\}$ убывают по крайней мере как члены геометрической прогрессии. При практических расчетах гарантируется лишь ограниченность $\{\gamma_n\}$ сверху, определяемая длиной разрядной сетки и правилами округления. Доказано, что в этих условиях погрешность итераций имеет некоторое предельное (ненулевое) значение, и неограниченное продолжение итераций не обеспечивает неограниченного приближения к корню: будут наблюдаться осцилляции в ближайшей окрестности последнего.

Приведем пример решения уравнения методом итерации.

Пример 6.1. Найти корень уравнения $x = \cos x$. Из рис. 6.6 видно, что уравнение имеет единственный корень ξ, причем $0.5 < x < 1$. Сходимость процесса гарантируется, поскольку $\cos' x = -\sin x$ и в окрестности корня уравнения модуль производной меньше единицы.

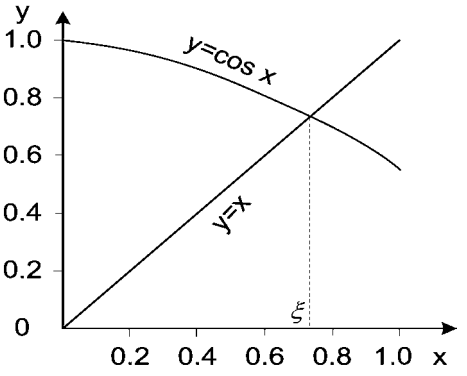

Рис. 6.6. Решение уравнения $x = \cos x$

Положим $x_0 = 0.7500$. Применяя формулу (6.7.3), последовательно находим:

$$x_1 = 0.7317, \quad x_2 = 0.7440, \quad x_3 = 0.7357,$$
$$x_4 = 0.7413, \quad x_5 = 0.7376, \quad x_6 = 0.7401,$$
$$x_7 = 0.7364, \quad x_8 = 0.7395, \quad x_9 = 0.7387.$$

В качестве корня с тремя верными цифрами после запятой можно принять 0.739.

6.7.3. Обеспечение сходимости итераций

Из расходимости итераций не следует отсутствие решения: просто процесс мог быть построен неудачно. Уравнение (6.7.1) может иметь много форм вида (6.7.2). Нас интересуют только те из них, для которых $|\varphi'(x)| < 1$ хотя бы в некоторой окрестности корня. Рассмотрим приемы преобразования (6.7.1) или (6.7.2) в такие уравнения.

Переход к обратной функции. Пусть уравнение (6.7.1) имеет равносильное уравнение вида (6.7.2), но $|\varphi'(x)| > 1$. Тогда функция $y = \varphi(x)$ имеет однозначную обратную функцию $x = \psi(y)$, для которой $|d\psi/dy| = 1/|\varphi(x)| < 1$. Если ξ — корень уравнения $x = \varphi(x)$, то $\xi = \varphi(\xi)$. В этом случае $\psi(\xi) = \psi(\varphi(\xi)) = \xi$; значит, ξ является и корнем уравнения $y = \psi(y)$. Последнее удовлетворяет условиям сходимости итераций.

Пример 6.2. Найти наименьший положительный корень уравнения $x = \operatorname{tg} x$.

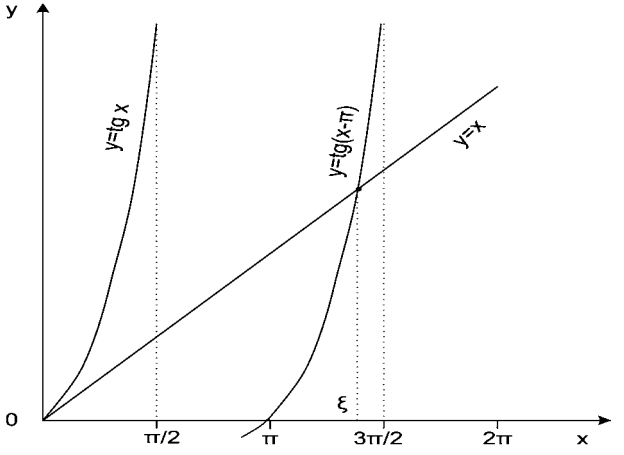

Рис. 6.7. К отысканию наименьшего положительного корня уравнения $x = \operatorname{tg} x$

Из рис. 6.7 видно, что корень, удовлетворяющий предъявленным требованиям, лежит в пределах $\pi < \xi < 3\pi/2$. Но $\varphi'(x) = 1/\cos^2 x > 1$, и итерации по исходной формуле окажутся расходящимися. Учитывая положение корня, перепишем исходное уравнение в виде $x = \operatorname{tg}(x - \pi)$. Взяв арктангенс от обеих частей последней записи, получим уравнение

$$x = \pi + \operatorname{arctg} x.$$

Модуль производной правой части не превосходит 1/11, так как

$$|(\pi + \arctan x)'| = 1/(1 + x^2) < 1/(1 + \pi^2) \approx 1/11.$$

Поэтому для нового уравнения процесс итерации будет сходящимся. Полагая $x_0 = \frac{1}{2}(\pi + 3\pi/2) = 5\pi/4$, имеем $x_0 = 3.92699081$; $x_1 = 4.46304061$; $x_2 = 4.49196705$; $x_3 = 4.49334136$; $x_4 = 4.49340624$; $x_5 = 4.49340930$. Отметим, что результат шестой итерации дает девять верных десятичных цифр: $x_6 = 4.49340945$.

Получим теоретическую оценку близости третьего приближения к корню, считая $q = 1/11$ и $x_1 - x_0 = 0.536$. На основании формулы (6.7.6) для случая $\varphi'(x) > 0$ имеем

$$|x_3 - \xi| \le \frac{(1/11)^3}{1 - 1/11} \cdot 0.536 = \frac{0.536}{1210} < 0.0005.$$

Следовательно, старшие четыре цифры x_3 верны, что подтверждается сравнением x_3 с x_6.

Подбор множителя. Пусть на отрезке $[a,b]$ функция $g(x) \neq 0$. Умножим уравнение $f(x) = 0$ на $g(x)$: $f(x)g(x) = 0$. Вычитая обе части последнего уравнения из x, получаем

$$x = x - f(x)g(x).$$

Условие сходимости итераций имеет вид

$$|1 - [g(x)f(x)]'| < 1$$

или

$$0 \leq [g(x)f(x)]' < 2.$$

Функцию $g(x)$ можно выбирать различными способами.

а) $g(x) = \text{const}$. Пусть на отрезке $[a,b]$ производная $f(x)$ ограничена сверху и снизу: $0 < m_1 \leq f'(x) \leq M_1$. Положим $g(x) = \lambda = 2/(M_1 + m_1)$. Тогда

$$|\varphi'(x)| = \left|1 - \frac{2}{M_1 + m_1} f'(x)\right| \leq 1 - \frac{2m_1}{M_1 + m_1} = \frac{M_1 - m_1}{M_1 + m_1} < 1,$$

и процесс итерации по формуле

$$x = x - \frac{2}{M_1 + m_1} f(x)$$

сходится.

Пример 6.3. Пусть $f(x) = x^3 - x^2 - 1000 = 0$. В промежутке $10 \leq x \leq 11$ имеется единственный корень этого уравнения. Производная $f'(x) = 3x^2 - 2x$ на отрезке $[10, 11]$ монотонно возрастает. Значит, $m_1 = f'(10) = 280$ и $M_1 = f'(11) = 341$. Отсюда

$$\lambda = \frac{2}{280 + 341} = \frac{2}{621} \approx \frac{1}{300}.$$

Для итераций воспользуемся уравнением

$$x = x - \frac{x^3 - x^2 - 1000}{300}.$$

Полагая $x_0 = 10$, находим $x_1 = 10.3333333$; $x_2 = 10.3446913$; $x_3 = 10.3446910$. Уже третья итерация дает девять верных знаков.

б) g(x)=1/f'(x). Если выбрать $g(x) = 1/f'(x)$, то

$$\varphi'(x) = \left(x - \frac{f(x)}{f'(x)}\right)' = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} = f(x)\frac{f''(x)}{[f'(x)]^2}.$$

В достаточно близкой окрестности корня $|f(x)| \approx 0$, и модуль производной $\varphi(x)$ весьма мал. Итак, в варианте б) решается уравнение

$$x = x - f(x)/f'(x).$$

Соответствующая расчетная схема эквивалентна методу Ньютона.

6.7.4. Метод Вегстейна

Для запуска метода Вегстейна [59] необходимо вычислить $x_1 = \varphi(x_0)$ и положить $y_1 = x_1$. На каждом шаге метода $y_k = \varphi(x_{k-1}), \quad k = 2, 3\ldots$ вычисляется однократно.

Пусть x_{k-1} и x_k — результаты двух последних итераций. Вычислим $y_{k-1} = \varphi(x_{k-2})$ и $y_k = \varphi(x_{k-1})$ и отметим на кривой $y = \varphi(x)$, показанной на рис. 6.8, точки (x_{k-2}, y_{k-1}) и (x_{k-1}, y_k).

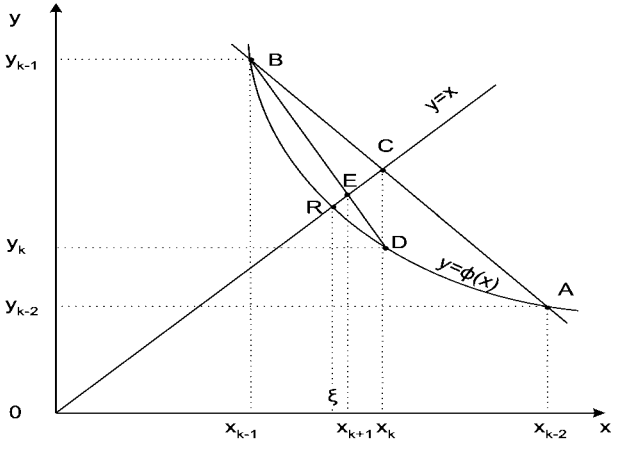

Рис. 6.8. Геометрический смысл метода Вегстейна

Дж. Вегстейн предложил в качестве очередного приближения x_k принимать абсциссу точки C пересечения хорды AB с прямой $y = x$.

Получим необходимые расчетные соотношения. Уравнение прямой, проходящей через точки A и B, есть

$$\frac{y - y_A}{y_B - y_A} = \frac{x - x_A}{x_B - x_A}$$

или

$$\frac{y - y_{k-1}}{y_k - y_{k-1}} = \frac{x - x_{k-2}}{x_{k-1} - x_{k-2}}.$$

Поскольку мы ищем точку пересечения прямой AB с прямой $y = x$, то, заменив y на x в каждом равенстве, придем к уравнению

$$\frac{x - y_{k-1}}{y_k - y_{k-1}} = \frac{x - x_{k-2}}{x_{k-1} - x_{k-2}}.$$

Решая его относительно $x = x_k$, находим

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

Выражение для x_k может быть записано в более удобной форме:

$$x_k = y_k - \frac{(y_k - y_{k-1})(y_k - x_{k-1})}{(y_k - y_{k-1}) - (x_{k-1} - x_{k-2})}, \qquad k = 2, 3, \ldots$$

Замечание. Мы применяли метод Вегстейна для решения уравнения $x = \varphi(x)$. Его можно использовать и для уравнений вида $f(x) = 0$. В этом случае полагают $x_0 = a$, $x_1 = b$. Приближения $\{x_k\}$, $k = 2, 3, \ldots$ находят применением метода хорд к отрезку с концами x_{k-2} и x_{k-1}.

Пример 6.4. Используем метод Вегстейна для решения уже встречавшегося нам уравнения $x = \cos x$ при том же начальном приближении $x_0 = 0.7500$. Для сравнения воспроизведем и результаты, полученные методом простой итерации (табл. 6.1). Результаты первой итерации, как это предполагается в методе Вегстейна, являются общими. Уже первое применение основного алгоритма Вегстейна дает четыре верных знака результата, которые в методе простой итерации не были получены за 9 шагов. Полученный по Вегстейну x_4 имеет 9 верных знаков.

Сходимость метода Вегстейна обеспечивается при любом значении $\varphi'(x)$. Если же $|\varphi'(x)| < 1$, то его применение, как это видно из табл. 6.1, существенно ускоряет сходимость.

Таблица 6.1. Сравнение итерационных методов

Результаты	Итерации	
	простая	по Вегстейну
x_0	0.7500	0.7500
x_1	0.7317	0.7317
x_2	0.7440	0.7391
x_3	0.7357	0.7391
x_4	0.7413	-
x_5	0.7376	-
x_6	0.7401	-
x_7	0.7384	-
x_8	0.7395	-
x_9	0.7389	-

6.7.5. Применение метода итераций для приближенного вычисления значения функций

Всякую функцию $y = f(x)$ можно представить (не единственным образом) как неявно заданную уравнением

$$F(x, y) = 0. \tag{6.7.7}$$

Может оказаться, что его решение относительно y методом итераций удобнее, чем вычисление функции $f(x)$ каким-либо другим путем. В частности, применяя к (6.7.7) метод Ньютона, получаем

$$y_{n+1} = y_n - \frac{F(x, y_n)}{F'_y(x, y_n)}, \qquad n = 0, 1, \ldots \tag{6.7.8}$$

Исходя из некоторого $y_0(x)$, будем находить $y_1, y_2, \ldots, y_n, \ldots$ Если итерации сходятся, то после конечного числа шагов в качестве $f(x)$ можно принять y_n: $f(x) \approx y_n(x)$.

Проиллюстрируем этот подход на примере вычисления фунции $y = \sqrt{x}$, которая на некоторых специализированных ЦВМ не входит в число элементарных машинных операций. Запишем эту функцию в неявном виде: $F(x, y) = x - y^2 = 0$. Тогда

$$y_{n+1} = y_n - \frac{x - y_n^2}{-2y_n}.$$

После упрощений имеем

$$y_{n+1} = y_n/2 + x/(2y_n).$$

Окончательно

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right), \qquad n = 0, 1, \ldots \tag{6.7.9}$$

Связанный с равенством (6.7.9) итерационный процесс показан на рис. 6.9.

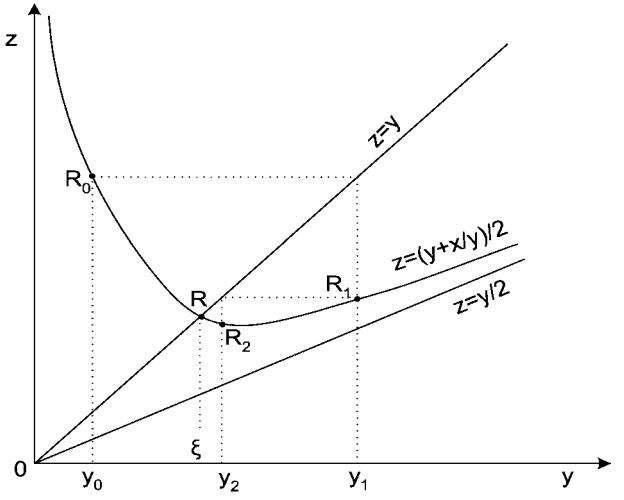

Рис. 6.9. Извлечение квадратного корня методом итераций

Дифференцируя первую часть (6.7.9), убеждаемся, что $\varphi'(y) = (1 - x/y^2)/2$, и при $\sqrt{x/2} < y < \infty$ модуль производной $|\varphi'(y)| < 1/2$. Если $y_0 < \sqrt{x}$, то $\varphi'(y_0) < 0$ и очередное приближение окажется по другую сторону точки $\xi = \sqrt{x}$ (может быть, дальше от нее, чем было начальное приближение). Однако в последующем будет иметь место монотонная сходимость итераций. Таким образом, лучше выбирать $y_0 > \sqrt{x}$.

Пример 3.5. Покажем описанный процесс для $x = 2$, $y_0 = 1$. Тогда

$$\begin{aligned} y_1 &= 0.5(1 + 2/1) = \frac{3}{2} = 1.5000; \\ y_2 &= 0.5(3/2 + 2 \cdot 2/3) = \frac{17}{12} = 1.4166; \\ y_3 &= 0.5(17/12 + 2 \cdot 12/17) = \frac{577}{408} = 1.41425. \end{aligned}$$

С девятью верными знаками после запятой $\sqrt{2} = 1.414213562$. Итак, уже третье приближение обеспечило нам 6 верных знаков результата.

6.8. Метод Эйткена

Пусть результаты некоторого процесса

$$\begin{aligned} r_0 &= r + e, \\ r_1 &= r + ef, \\ r_2 &= r + ef^2, \\ &\cdots\cdots\cdots\cdots \end{aligned} \tag{6.8.1}$$

Таким образом, погрешности представляются членами убывающей геометрической прогрессии. Тогда конечные разности

$$\begin{aligned} \Delta_0 &= e(f-1), \\ \Delta_1 &= ef(f-1). \end{aligned}$$

Теперь ясно, что $f = \Delta_1/\Delta_0$. Далее,

$$e = \frac{\Delta_0}{f-1} = \frac{\Delta_0}{\Delta_1/\Delta_0 - 1} = \frac{(\Delta_0)^2}{\Delta_1 - \Delta_0}.$$

Отсюда следует, что исправленный результат должен вычисляться согласно

$$r = x_0 - \frac{(\Delta_0)^2}{\Delta_1 - \Delta_0}.$$

Эта формула определяет второе название данного алгоритма (Δ^2-процесс). Он имеет многочисленные применения и обсуждается в самых авторитетных учебниках по численным методам [29, 59].

Δ^2-метод должен применяться к трем смежным членам *линейно* (со скоростью геометрической прогрессии) сходящейся последовательности. При необходимости продолжения следует получить новую тройку результатов.

6.9. Решение полиномиальных уравнений

Отдельного рассмотрения заслуживает нахождение корней *полинома*. По сравнению с трансцендентными функциями полиномы имеют то преимущество, что наперед точно известно число их корней. Для полиномов существуют также оценки числа положительных, отрицательных и комплексных корней.

Свойства алгебраических уравнений

$$a_n x^n + a_{n-1} x_{n-1} + \ldots + a_1 x + a_0 = 0$$

резюмируются следующим образом:

1. Алгебраическое уравнение порядка n имеет n корней, которые могут быть действительными или комплексными. Если для такого уравнения получена $(n+1)$ перемена знака, то все его корни отделены.

2. Если все коэффициенты алгебраического уравнения действительные, то все комплексные корни образуют сопряженные пары. Алгебраическое уравнение нечетной степени имеет по крайней мере один действительный корень.

3. Число положительных действительных корней не превосходит числа перемен знака в последовательности коэффициентов.

4. Число отрицательных действительных корней не превосходит числа перемен знака в последовательности коэффициентов при замене x на $-x$.

Согласно теореме Декарта, число положительных корней алгебраического уравнения с учетом их кратностей равно числу перемен знака в системе коэффициентов $\{a_0, a_1, \ldots, a_n\}$ или меньше этого количества на четное число. Число отрицательных корней с учетом их кратностей равно числу *постоянств* знака в той же системе или меньше этого количества на четное число.

Грубая оценка модулей всех корней уравнения дается неравенством

$$|x_k| < 1 + \frac{A}{|a_0|} = R,$$

где A — максимальный из модулей остальных коэффициентов уравнения. С другой стороны, если $B = \max\{|a_0|, |a_1|, \ldots, |a_{n-1}|\}$, то

$$|x_k| > \frac{1}{1 + B/|a_n|} = r.$$

Эти неравенства указывают, что все корни уравнения на комплексной плоскости заключены в круговое кольцо, $r < |x| < R$. С их помощью легко устанавливаются границы для положительных и отрицательных корней.

За верхнюю границу положительных корней многочлена $P_n(x)$ с $a_0 > 0$ можно принять число

$$R = 1 + \sqrt[m]{B/a_0},$$

где m — индекс первого отрицательного коэффициента из $a_1, a_2, \ldots, a_n$, а B — наибольший из модулей отрицательных коэффициентов.

Согласно Вестерфильду, модули всех корней приведенного (с единичным старшим коэффициентом) многочлена $P_n(x)$ лежат в круге, радиус которого не превосходит суммы двух наибольших из чисел $\sqrt[m]{|a_m|}$, $m = \overline{1,n}$.

Перечисленные результаты не только облегчают решение уравнений, но и используются для анализа устойчивости систем автоматического управления.

Прямые («формульные») методы нахождения корней применяются для уравнений до четвертой (практически — до третьей) степени включительно. Уравнения степени 5 и выше в *общем* случае неразрешимы в радикалах. Численные методы решения алгебраических уравнений высоких степеней обычно связаны с последовательным понижением степени уравнения. Согласно теореме Безу, при любом x_0

$$a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n = (x - x_0)(b_0x_{n-1} + b_1x^{n-2} + \ldots + b_{n-2}x + b_{n-1}) + b_n,$$

причем $b_n = P_n(x_0)$. Выполняя перемножение и приравнивая коэффициенты при одинаковых степенях x, получаем

$$b_0 = a_0, \quad b_1 = a_1 + x_0b_0, \quad b_2 = a_2 + x_0b_1, \quad \ldots, \quad b_n = a_n + x_0b_{n-1}.$$

Если x_0 — корень исходного многочлена, то $b_n = 0$.

Найдя каким-либо методом корень x^* и разделив уравнение на $(x - x^*)$, можно понизить эту степень на 1 и применить к новому уравнению ту же процедуру. При решении полиномиальных уравнений коэффициенты многочлена

$$P_{n-1}(x) = P_n(x)/(x - x*)$$

определяются по рекуррентным формулам

$$b_n = a_0, \qquad b_j = a_j + b_{j-1}x^*, \quad j = \overline{1, n-1}.$$

Специальные методы отыскания *комплексных* корней обычно связаны с выделением из исходного уравнения квадратичного множителя:

$$(x^2 + px + q)(x^{n-2} + b_{n-1}x^{n-3} + \ldots + b_3x + b_2) + b_1x + b_0 = 0,$$

где $b_1x + b_0$ — линейный остаточный член. Если $b_0 = b_1 = 0$, то деление выполняется без остатка. Сравнивая коэффициенты при одинаковых степенях x в обоих формах записи уравнения, приходим к

$$\begin{aligned}
b_{n-1} &= a_{n-1} - p, \\
b_{n-2} &= a_{n-2} - pb_{n-1} - q, \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
b_{n-j} &= a_{n-j} - pb_{n+1-j} - qb_{n+2-j}, \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
p &= (a_1 - qb_3)/b_2, \\
q &= a_0/b_2.
\end{aligned}$$

Сначала задаются вероятные значения p и q. Далее подряд выполняются вышеуказанные вычисления, в конце которых p и q пересчитываются и оценивается значимость уточнений. Процедура сводится к решению системы двух уравнений с двумя неизвестными методом простой итерации. В методе Берстоу для решения аналогичной системы используется метод Ньютона.

Поскольку в подавляющем большинстве случаев корни вычисляются приближенно, при понижении степени алгебраического уравнения неизбежна потеря точности, с которой могут быть найдены последующие корни. Известна крайняя неустойчивость корней некоторых полиномов как функций от их коэффициентов (см. уже упоминавшийся пример Уилкинсона). Даже небольшая погрешность в исходных данных приводит к большим погрешностям в решении или вовсе неверному результату. Так, уравнение

$$x^2 - 2x + 1 = 0$$

имеет кратный вещественный корень $x_1 = x_2 = 1$, а

$$x^2 - 2x + 1.01 = 0$$

— пару комплексных корней $x_1 = 1 + 0.1i$, $x_2 = 1 - 0.1i$.

В общем, задачи, включающие в себя нахождение нулей полиномов высокой степени, требуют либо предельно точного вычисления коэффициентов, либо совершенно иного подхода. Примером является задача вычисления собственных чисел матрицы (см. разд. 8.6–8.8).

6.10. Сопоставление методов

Важной характеристикой процесса решения уравнения является *скорость сходимости*. В общем случае говорят, что метод сходится со скоростью r, если предельное отношение погрешностей на смежных шагах

$$\lim_{i\to\infty} \frac{|e_{i+1}|}{|e_i|^r} = C,$$

где C — некоторая конечная ненулевая константа. Для метода половинного деления $r = 1$ (линейная сходимость), $C = 1/2$. Случай $r = 2$ соответствует квадратичной сходимости. Практически важны значения не только r, но и C: при малой C линейная сходимость может оказаться предпочтительнее квадратичной с большой константой. Важную роль играет также качество начального приближения.

Метод секущих фактически эквивалентен методу Ньютона с конечно-разностной аппроксимацией производной. Он сходится сверхлинейно с показателем $r = (1 + \sqrt{5})/2$ =1.618. Трудности при его применении возникают в случаях нулевой производной и кратных корней. При кратных корнях методы Ньютона и секущих будут сходиться с линейной скоростью. Если кратность корня известна, можно восстановить быструю сходимость.

Скорость сходимости обратной квадратичной интерполяции равна 1.839.

Методы Ньютона и секущих можно использовать и в комплексных областях.

Глава 7. Вычислительные методы линейной алгебры

К численным методам алгебры традиционно относят методы работы с матрицами: обращения, решения систем линейных алгебраических уравнений, вычисления определителей, нахождения собственных значений и собственных векторов, а также поиск нулей многочленов.

7.1. Матрицы

Пусть m и n — целые числа, бóльшие или равные 1. Рассмотрим таблицу чисел $\{a_{i,k}\}$, содержащую m строк и n столбцов:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

Эта совокупность чисел называется *матрицей* вида (m, n). Матрицы являются компактным и удобным средством описания математических моделей — в особенности линейных. Обычно считают, что изобилие (по количеству и начинке) пакетов соответствующих подпрограмм снимает все проблемы. Однако возникает вопрос о *рациональном* использовании имеемых возможностей и при «самодеятельном программировании» — учете «источников повышенной опасности».

Для краткости в дальнейшем будем обозначать матрицы большими буквами $A(m, n)$, при необходимости указывая в скобках порядок матрицы по измерениям.

Числа $\{a_{ik}\}$ суть *элементы* матрицы. Если $m = 1$, то матрица называется *вектором-строкой*, если $n = 1$ — *вектором-столбцом*. При $m = n$ матрица называется *квадратной* порядка n. С каждой квадратной матрицей связывается ее *определитель* — число, получаемое по известным правилам из элементов матрицы.

7.2. Операции над матрицами

Транспонированная матрица A^T вида (n, m) получается из исходной A заменой строк на столбцы:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \dots & \dots & \dots & \dots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix}.$$

В частности, вектор-строка

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \end{bmatrix}$$

после транспонирования становится вектором-столбцом, и наоборот.

Имеет смысл привести фрагмент программы, записывающий транспонированную матрицу на месте исходной (квадратной):

```
do i=2,n
   do j=1,i-1
      b=a(i,j); a(i,j)=a(j,i); a(j,i)=b
   end do
end do
```

Укажем часто допускаемые здесь ошибки:

- без использования промежуточной переменной B матрица превратится в симметричную;
- задание для внутреннего цикла верхней границы n выполнит транспонирование *дважды*, следствием чего явится ... исходная матрица.

Суммой $C = A + B$ двух матриц одного вида (m, n) называется матрица C того же вида с элементами $c_{ik} = a_{ik} + b_{ik} = b_{ik} + a_{ik}$. Отсюда следует, что $A + B = B + A$.

Произведением матрицы A *на число* λ является матрица того же вида (m,n) с элементами $\{\lambda a_{ik}\}$. Она обозначается λA или $A\lambda$. Нетрудно видеть, что справедливы свойства:

$$\begin{aligned} \lambda(A+B) &= \lambda A + \lambda B; \\ (\lambda+\mu)A &= \lambda A + \mu A; \\ (A+B)+C &= A+(B+C). \end{aligned}$$

Произведением $C = A \times B$ *матриц* вида (m,n) и (n,k) называется матрица $C(m,k)$, для которой

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}, \qquad i = \overline{1,m}; \quad j = \overline{1,k},$$

т. е. элемент c_{ij} равен сумме произведений элементов i-й строки матрицы A на соответствующие элементы j-го столбца матрицы B. Подчеркнем, что протяженность исходных матриц по «внутреннему» измерению обязательно должна совпадать. Приведем фрагмент программы перемножения матриц:

```
do i=1,m
   do j=1,k
      s=0
      do l=1,n; s=s+a(i,l)*b(l,j); end do
      c(i,j)=s
   end do
end do
```

Ответный элемент в этом фрагменте можно рассматривать как *скалярное произведение* — сумму парных произведений одноименных проекций строки и столбца. Запишем программу перемножения тех же матриц с использованием сечений массивов и встроенной функции `dot_product`:

```
do i=1,m
   do j=1,k
      c(i,j)=dot_product(a(i,:),b(:,j))
   end do
end do
```

Для того чтобы наряду с произведением AB имело смысл и произведение BA, необходимо, чтобы $m = k$. Тогда матрица AB будет квадратной порядка m, а матрица BA — квадратной порядка n. Если

A и B — квадратные матрицы порядка n, то произведения AB и BA также являются квадратными матрицами порядка n, в общем случае не равными друг другу. Например,

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 \\ -1 & 3 \end{bmatrix}, \quad AB = \begin{bmatrix} 1 & 7 \\ -3 & 9 \end{bmatrix}, \quad BA = \begin{bmatrix} 2 & 7 \\ -2 & 8 \end{bmatrix}.$$

Таким образом, для произведения матриц коммутативный (переставновочный) закон в общем случае неверен.

Часто приходится иметь дело с *диагональным* матричным сомножителем. Умножение квадратной матрицы на диагональную справа эквивалентно умножению столбцов, а слева — строк на соответствующие элементы диагонали.

Имеют место следующие свойства действий над матрицами:

$$\begin{aligned} [A(m,n) + B(m,n)]C(n,k) &= AC + BC, \\ C(m,n)[A(n,k) + B(n,k)] &= CA + CB, \\ (\lambda A)B = \lambda(AB) &= A(\lambda B). \end{aligned}$$

Если соседние члены некоторой последовательности матриц имеют по своим измерениям протяженность, при которой их произведения существуют, то для произведения матриц справедлив ассоциативный закон. Так,

$$\underbrace{A(m,n) \times B(n,k)}_{D(m,k)} \times C(k,l) = A(m,n) \times \underbrace{[B(n,k) \times C(k,l)]}_{F(n,l)} = G(m,l).$$

Это позволяет выписывать произведения матриц без указания группировки сомножителей (но не нарушая порядок их следования).

Теперь мы можем выписать полный набор свойств операций транспонирования:

1. Транспонированная симметричная матрица равна исходной.
2. Дважды транспонированная матрица совпадает с исходной.
3. Транспонированная сумма матриц равна сумме транспонированных слагаемых.
4. Транспозиция произведения равна произведению транспозиций, взятому в обратном порядке: $(AB)^T = B^T A^T$.

5. Произведение матрицы на транспонированную ей дает квадратную симметричную матрицу, так как $(AA^T)^T = (A^T)^T A^T = AA^T$.

6. Транспонированная обратная матрица обратна транспонированной исходной.

С помощью операции транспонирования скалярное произведение векторов записывается в терминах обычного матричного умножения: для столбцов $(a, b) = a^T b$, для строк $(a, b) = ab^T$.

Пусть дана действительная квадратная $n \times n$ матрица A и n-мерные векторы x и y. Скалярное произведение

$$(Ax, y) = \sum_{i=1}^{n} y_i \Big(\sum_{j=1}^{n} a_{ij} x_j \Big) = \sum_{j=1}^{n} x_j \Big(\sum_{i=1}^{n} a_{ij} y_i \Big) = (x, A^T y).$$

Значит, матричный множитель в скалярном произведении можно переставлять, заменяя его транспонированным.

В комплексном случае транспонирование сопровождается заменой всех элементов матрицы сопряженными числами. Результат обозначается A^*.

Определитель произведения квадратных матриц равен произведению определителей сомножителей.

Естественным развитием операции произведения является операция возведения матрицы в целую степень. Она применима только к квадратным матрицам (почему?). Целые *отрицательные* степени получаются возведением в степень обратной матрицы.

Для быстрого возведения квадратной матрицы в целую степень можно использовать последовательные возведения в квадрат и управляемое двоичным разложением показателя степени накопление произведения. Упомянутое разложение формируется из остатков от деления на 2 требуемой кратности и последовательно получаемых частных — пока очередное частное не окажется нулем. Пусть нужная степень равна 19. Читая справа налево равенство $19_{10} = 10011_2$, убеждаемся, что в произведение должны войти первая, вторая и шестнадцатая степени исходной матрицы.

Общий алгоритм возведения в степень k матрицы A имеет следующий вид:

1. Положить произведение $P = I$ (единичная матрица), $Q = A$, $m = k$.

2. Вычислить $\tilde{m} = [m/2]$ (целая часть), $r = m - 2\tilde{m}$ (остаток).

3. Если $r = 1$, копить произведение $P = P * Q$.

4. Если $\tilde{m} = 0$, перейти к этапу 6.

5. Положить $Q = Q * Q$, $m = \tilde{m}$. Перейти к этапу 2.

6. Конец алгоритма.

С помощью степеней матриц можно определять матричные функции от матриц и вычислять суммы соответствующих сходящихся рядов. Простейшим случаем является матричная геометрическая прогрессия, широко используемая, например, в теории марковских цепей и теории очередей. Ее сумма

$$S = \sum_{k=0}^{\infty} A^k = (I - A)^{-1}$$

(понятия единичной и обратной матриц определены в разд. 7.4). Эта формула справедлива при условии, что норма матрицы $||A|| < 1$. Проницательный читатель без труда получит ее по аналогии с суммой скалярной геометрической прогрессии.

7.3. Ранг матрицы

Пусть A есть матрица вида (m, n) и дано целое число r, $0 < r \leq \min(m, n)$. Элементы матрицы, стоящие в каких-либо r строках и r столбцах матрицы A, образуют квадратную матрицу порядка r. Так как r из m рядов можно выбрать C_m^r различными способами, то различных квадратных матриц порядка r в матрице A имеется $C_m^r \cdot C_n^r$. Пусть все определители этих матриц равны нулю. Поскольку определитель порядка $(r+1)$ есть сумма произведений некоторых чисел на определители r-го порядка, то все определители порядка $(r + 1)$ и выше окажутся равными нулю.

Рангом матрицы называется наивысший порядок r выделяемого из нее ненулевого определителя. В этом случае в ней обязательно должны найтись ненулевые определители порядка $r - 1, r - 2, \ldots, 1$. Матрице, все элементы которой равны нулю, приписывается ранг 0.

7.4. Неособая, единичная, обратная матрицы

Пусть A — квадратная матрица n-го порядка. Квадратную матрицу с нулевым определителем называют *особой* (вырожденной) матрицей. Матрица A считается неособой, если ее определитель отличен от нуля. Неособая матрица порядка n имеет полный ранг n, т. е. наибольший из возможных для нее рангов.

Единичной матрицей порядка n называют матрицу размера (n, n), у которой на главной диагонали стоят единицы, а остальные элементы равны нулю. Единичную матрицу обозначают буквами I или E (мнемоника в обоих случаях очевидна). Умножение произвольной матрицы (m, n) как справа, так и слева на соответствующих размеров единичную дает исходную матрицу. Таким образом, при матричном умножении I играет роль, аналогичную единице в умножении чисел.

Обозначим через $|A|$ определитель матрицы A. Пусть A_{ik} есть *алгебраическое дополнение* элемента a_{ik} — подматрица, получаемая из исходной вычеркиванием i-й строки и k-го столбца и умножением на $(-1)^{i+k}$. Считая матрицу A неособой, введем для нее *обратную* матрицу A^{-1}, определяемую равенством

$$A^{-1} = \begin{bmatrix} \dfrac{A_{11}}{|A|} & \dfrac{A_{21}}{|A|} & \cdots & \dfrac{A_{n1}}{|A|} \\ \dfrac{A_{12}}{|A|} & \dfrac{A_{22}}{|A|} & \cdots & \dfrac{A_{n2}}{|A|} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{A_{1n}}{|A|} & \dfrac{A_{2n}}{|A|} & \cdots & \dfrac{A_{nn}}{|A|} \end{bmatrix},$$

т. е. элемент в i-й строке и k-м столбце A^{-1} есть частное от деления на определитель $|A|$ алгебраического дополнения элемента исходной матрицы (иначе говоря, дополнение берется к элементу транспонированной матрицы).

Докажем, что произведение матриц A и A^{-1} есть единичная матрица. Обозначим $C = AA^{-1}$. Тогда

$$\begin{aligned} c_{ik} &= a_{i1}\frac{A_{k1}}{|A|} + a_{i2}\frac{A_{k2}}{|A|} + \ldots + a_{in}\frac{A_{kn}}{|A|} \\ &= \frac{1}{|A|}(a_{i1}A_{k1} + a_{i2}A_{k2} + \ldots + a_{in}A_{kn}). \end{aligned}$$

Известно, что сумма произведений элементов некоторого ряда матрицы на их алгебраические дополнения равна определителю, а на алгебраические дополнения другого параллельного ряда — нулю. Таким образом, диагональные элементы $\{c_{ii}\}$ окажутся равными единице, а все прочие будут нулями. Но это и означает, что $C = I$, т. е. $AA^{-1} = I$. Аналогично доказывается равенство $A^{-1}A = I$.

Обратная матрица всегда неособая. Матрица, обратная по отношению к обратной, есть исходная матрица.

Если A и B — неособые матрицы порядка n, то AB — также неособая матрица (ее определитель как произведение ненулевых определителей сомножителей отличен от нуля), и

$$(AB)^{-1} = B^{-1}A^{-1}.$$

Если матрица A — квадратная и невырожденная, то операции обращения и транспонирования перестановочны: транспонированная обратная матрица обратна транспонированной исходной. Их совместное выполнение обозначается A^{-T}.

Определитель обратной матрицы обратен определителю исходной. Обратное произведение равно произведению обращенных сомножителей, взятому в обратном порядке: $(AB)^{-1} = B^{-1}A^{-1}$.

В заключение приведем пример расчета обратной матрицы.

Пример 7.1. Пусть дана матрица

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 0 & 2 \\ -1 & 3 & -2 \end{bmatrix}.$$

Раскрывая ее определитель по элементам второго столбца, имеем

$$|A| = 3 \cdot [(-1)^{1+2}(-2+2) + (-1)^{3+2} \cdot (4-1)] = 3 \cdot (-3) = -9;$$

$$\begin{array}{lll} A_{11} = \begin{bmatrix} 0 & 2 \\ 3 & -2 \end{bmatrix} = -6; & A_{12} = -\begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix} = 0; & A_{13} = \begin{bmatrix} 1 & 0 \\ -1 & 3 \end{bmatrix} = 3; \\ A_{21} = -\begin{bmatrix} 3 & 1 \\ 3 & -2 \end{bmatrix} = 9; & A_{22} = \begin{bmatrix} 2 & 1 \\ -1 & -2 \end{bmatrix} = -3; & A_{23} = -\begin{bmatrix} 2 & 3 \\ -1 & 3 \end{bmatrix} = -9; \\ A_{31} = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} = 6; & A_{32} = -\begin{bmatrix} 2 & 1 \\ 1 & -2 \end{bmatrix} = -3; & A_{33} = \begin{bmatrix} 2 & 3 \\ 1 & 0 \end{bmatrix} = -3. \end{array}$$

Таким образом, обратная к A матрица

$$A^{-1} = -\frac{1}{9}\begin{bmatrix} -6 & 9 & 6 \\ 0 & -3 & -3 \\ 3 & -9 & -3 \end{bmatrix} = \begin{bmatrix} 2/3 & -1 & -2/3 \\ 0 & 1/3 & 1/3 \\ -1/3 & 1 & 1/3 \end{bmatrix}.$$

7.5. Матрицы специальной структуры

Важно различать два типа матриц: хранимые полностью и разреженные. Значительная часть элементов последних — нули; поэтому их ненулевые элементы хранятся как данные специальной структуры или генерируются по мере необходимости (пример — матрица Гильберта, в которой элемент $a_{ij} = 1/(i+j-1)$).

Под матрицами специальной структуры будем понимать матрицы с *закономерным* расположением ненулевых элементов: блочные (клеточные), треугольные, ленточные и диагональные. В *трехдиагональной* матрице равны нулю все элементы, кроме лежащих на главной диагонали и двух смежных с ней. Такие матрицы часто получаются при дискретизации дифференциальных уравнений (обыкновенных или с частными производными) и в иных задачах, включающих некоторую сеточную структуру. Классы диагональных и треугольных матриц замкнуты относительно линейных операций и умножения, а также (для невырожденных матриц) — обращения.

Преимущество работы со «структурными» матрицами состоит в экономии памяти под хранение данных и исключении операций, дающих заведомо нулевые результаты. Поскольку «Программа=алгоритм +структуры данных», учет нестандартной организации данных усложняет программирование. Для минимизации числа ошибок мы рекомендуем следующую технологию:

- записать матричную операцию покомпонентно для «развернутых» матриц;
- сузить границы циклов с учетом расположения ненулевых операндов;
- переписать операторы циклов и индексные выражения, отображая выбор данных на структуру их хранения.

К матрицам специальной структуры можно отнести также обсуждаемые ниже в специфических контекстах симметричные и ортогональные.

Численные методы, применяемые для обычных («хранимых», «плотных») матриц, весьма отличаются от методов, приспособленных для упакованных разреженных матриц регулярной структуры (треугольных, ленточных, блочных). Первые более универсальны; вторые в частных случаях намного эффективнее и позволяют решать задачи большей размерности.

7.5.1. Клеточные матрицы

Пусть дана некоторая матрица A. Разобьем ее на подматрицы меньших порядков (клетки, или блоки) с помощью горизонтальных и вертикальных перегородок, идущих вдоль всей матрицы. Например,

$$A = \left[\begin{array}{cc|c} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \hline a_{31} & a_{32} & a_{33} \end{array}\right],$$

где клетками являются матрицы

$$P = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}; \qquad Q = \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix}; \qquad R = \begin{bmatrix} a_{31} & a_{32} \end{bmatrix}; \qquad S = [a_{33}].$$

Тогда матрицу A можно рассматривать как составную, элементами которой служат клетки:

$$A = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}.$$

Понятно, что разбиение матрицы на блоки может производиться по-разному и определяется смыслом задачи. В *теории марковских цепей* вероятности переходов между состояниями описываются блочной матрицей вышеприведенного вида, причем $R = 0$. Здесь подматрица P описывает переходы между состояниями транзитивной группы, Q — из транзитивной группы в «невозвратную», S — внутри невозвратной. Заметим, что выход из невозвратной группы имеет нулевые вероятности.

Важным частным случаем клеточных матриц являются блочно-диагональные. Здесь вдоль главной диагонали располагаются квадратные блоки (в общем случае неодинакового размера), а прочие элементы являются нулевыми.

Действия над клеточными матрицами формально совершаются *применительно к клеткам* по тем же правилам, что над обычными — при условии совместимости клеток применительно к данной операции. Если, например,

$$A = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}, \qquad B = \begin{bmatrix} T & U \\ V & W \end{bmatrix},$$

то произведение

$$AB = \begin{bmatrix} PT + QV & PU + QW \\ RT + SV & RU + SW \end{bmatrix}.$$

Здесь необходимо, чтобы число столбцов у клеток первой матрицы равнялось числу строк у клеток второй.

Блочно-диагональную матрицу целесообразно хранить в виде трехмерного массива, при ссылках на который первый индекс указывает номер блока, а второй и третий — номера строк и столбцов в пределах блока. Границы строк и столбцов назначаются по их максимальным значениям.

Клеточные методы могут использоваться для решения больших систем линейных уравнений, когда матрица и вектор правых частей целиком не помещаются в оперативной памяти.

7.5.2. Треугольные матрицы

Квадратная матрица называется верхней (нижней) треугольной, если ее элементы, стоящие ниже (соответственно, выше) главной диагонали, равны нулю.

Определитель треугольной матрицы равен произведению ее диагональных элементов. Поэтому треугольная матрица является неособой, если все элементы ее диагонали отличны от нуля. Можно доказать, что сумма и произведение треугольных матриц одинаковой структуры (соответственно, верхних или нижних) есть треугольная матрица той же структуры.

Матрица, обратная к треугольной, есть треугольная матрица той же структуры. Это облегчает обращение треугольных матриц.

Всякую квадратную матрицу

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots a_{1n} \\ a_{21} & a_{22} & \dots a_{2n} \\ \dots\dots\dots\dots\dots\dots \\ a_{n1} & a_{n2} & \dots a_{nn} \end{bmatrix},$$

имеющую отличные от нуля главные диагональные миноры

$$\Delta_1 = a_{11}, \quad \Delta_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}, \quad \ldots, \quad \Delta_n = |A|,$$

можно представить в виде произведения LU нижней L и верхней U треугольных матриц. Это представление будет единственным, если заранее зафиксировать диагональные элементы одной из треугольных матриц (например, положить их равными 1).

Перемножая эти матрицы, получаем

$$a_{ij} = \sum_{k=1}^{j} l_{ik} u_{kj}, \quad i \geq j,$$

$$a_{ij} = \sum_{k=1}^{i} l_{ik} u_{kj}, \quad i < j.$$

Эти системы легко решаются с точностью до диагональных элементов. Для определенности можно принять $r_{ii} = 1, \quad i = \overline{1,n}$. Заметим, что

$$A^{-1} = U^{-1} L^{-1}.$$

Поскольку обращение треугольных матриц делается очень просто, обсуждаемое представление квадратной матрицы может быть полезным для ее обращения.

Для *упакованного* хранения треугольной матрицы ее построчно записывают в линейный массив. При этом вместо n^2 единиц памяти достаточно использовать $n(n+1)/2$ — асимптотически вдвое меньше. Перемещение вдоль строки идет с шагом 1. Сложение треугольных матриц (как и произвольных) применительно к линейным массивам можно выполнять в однократном цикле на общее число элементов в слагаемом. Кстати отметим, что в Ф90 имеются встроенные средства переформатирования матричных объектов.

Значительно сложнее обстоит дело при *умножении*. Продемонстрируем технологию работы с упакованными верхними треугольными матрицами «по разделениям». Нарисуем сомножители, заштрихуем их ненулевые части и линейно пронумеруем оставшиеся элементы, двигаясь по строкам — рис. 7.1.

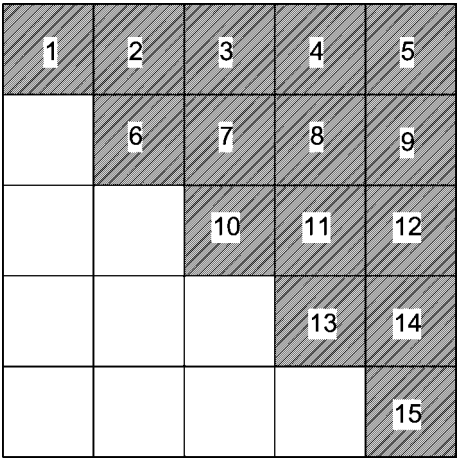

Рис. 7.1. Преобразование треугольной матрицы в линейный массив

Установим границы второго и внутреннего циклов и запишем вариант для матриц, хранимых обычным способом:

```
do i=1,n
   do j=i,n
      s=0
      do l=i,j; s=s+a(i,l)*b(l,j); end do
   end do
end do
```

Для верхней треугольной матрицы перемещению вдоль любого столбца от i-й к $(i+1)$-й строке соответствует приращение адреса $(n-i)$, начальный адрес $(i+1)$-й строки отличается от начального адреса предыдущей на $(n+1-i)$. Это позволяет записать процедуру перемножения верхних треугольных матриц:

```
subroutine trimult(n,a,b,c)
   integer, intent(in) :: n
   real, intent(in),  dimension(:) :: a,b
   real, intent(out), dimension(:) :: c
   real s

   integer i,j,k,l,m,r
   l=0; r=1
   do i=1,n
      do j=i,n
```

```
            s=0; m=l+j
            do k=i,j
               s=s+a(l+k)*b(m); m=m+n-k
            end do
            c(r)=s; r=r+1
         end do
            l=l+n-i
      end do
   end subroutine trimult
```

Рекомендуем читателю нарисовать *нижние* треугольные матрицы для конкретного n, проставить номера элементов и получить для них аналогичные соотношения.

Теперь рассмотрим *обращение* верхней треугольной матрицы. Обозначим $\{b_{ij}\}$ элементы исходной матрицы, $\{z_{ij}\}$ — элементы обратной. Матрица, обратная к верхней треугольной, также является верхней треугольной. Это позволяет переписать формулу для расчета элементов матрицы-произведения

$$c_{im} = \sum_{r=1}^{n} b_{ir} z_{rm} = \delta_{im}, \qquad m = \overline{1,n},$$

где δ_{im} — символ Кронекера, в форме

$$\delta_{im} = \sum_{r=i}^{m} b_{ir} z_{rm},$$

откуда

$$z_{im} = \Big(\delta_{im} - \sum_{r=i+1}^{m} b_{ir} z_{rm}\Big) \Big/ b_{ii}, \quad m = n, n-1, \ldots, 2; \quad i = m, m-1, \ldots, 1. \tag{7.5.1}$$

Элементы обратной матрицы можно записывать вместо элементов прямой, если формировать их по столбцам, начиная с последнего, а в пределах столбцов — снизу вверх. В каждом из столбцов сначала находится диагональный элемент, обратный соответствующему элементу исходной.

Ниже приводится процедура обращения верхней треугольной матрицы, записанной вместе с аналогичными ей в общий линейный массив.

```
   subroutine intr(n,b,e)
!*******************************************/
!* Обращение верхней треугольной матрицы,  */
!* записанной в одномерном исходном массиве */
!* по строкам, на месте исходной           */
!*     Разработчик Ю. И. Рыжиков           */
!*                  (c)                    */
!*         Версия 20.02.1995               */
!*******************************************/

      integer          n  !!  размер матрицы в неупаков.виде
      double precision b  !!  массив, хранящий матрицу
      integer          e  !!  адрес последнего эл-та матрицы
                          !!  в массиве b
      dimension        b(*)
      integer          i,lim,lir,lii,lmm,lrm,lrs,m,r
      double precision d

      lmm=e
      do m=n,1,-1              !! строки обратной матрицы
         lii=lmm
         lim=lmm+m-n-1
         b(lmm)=1d0/b(lmm)
         lrs=lmm
         lii=lmm+m-n-2
         do i=m-1,1,-1         !! столбцы
             lir=lii+1
             lrm=lrs
             d=0
             do r=i+1,m
                d=d-b(lir)*b(lrm)
                lir=lir+1
                lrm=lrm+n-r
             end do            !! по r
             b(lim)=d/b(lii)
             lim=lim+i-1-n
             lii=lii+i-2-n
             lrs=lrs-n+i
         end do                !! по i
```

```
        lmm=lmm+m-n-2
      end do                    !! по m
    end                         !! intr
```

7.5.3. Ленточные матрицы

Многие задачи (в частности, решения уравнений в частных производных) приводят к ленточным матрицам, ненулевые элементы которых сосредоточены в полосе, окаймляющей главную диагональ. Здесь достаточно хранить только эту ленту в матрице размером $s \times n$, где s — ширина ленты.

Ленточная матрица не обязательно располагается симметрично относительно главной диагонали.

7.5.4. Диагональные матрицы

Диагональные матрицы замечательны возможностью их экономичного хранения (в виде векторов соответствующей длины) и удобством работы с ними. Из общих правил перемножения матриц следует, что умножение квадратной матрицы на диагональную слева эквивалентно умножению ее строк, а справа — столбцов на соответствующие элементы диагонали. Эти правила можно перенести и на блочно-диагональные матрицы.

7.5.5. Нерегулярные разреженные матрицы

Нерегулярными считаются разреженные матрицы, в размещении ненулевых элементов которых нет простой закономерности. Большие матрицы такого вида удобно хранить в виде *списковых структур*. Например, с каждой строкой матрицы можно связать список ненулевых элементов с указанием номера столбца, значения элемента и адреса следующего элемента списка. Такое представление матриц эффективно используется при решении больших систем линейных уравнений методом итераций. Списковое представление данных необходимо при расчете сетей обслуживания с неоднородными заявками, поскольку некоторые типы заявок минуют отдельные узлы. Уравнения баланса для них при *общем* множестве узлов могут содержать нулевые столбцы и строки, что гарантирует «авост» при попытке их решения.

Студентам, специализирующимся в области информационных технологий, мы рекомендуем попытаться написать программы сложения и перемножения матриц, представленных списками ненулевых элементов.

7.6. Нормы матрицы и вектора

Совокупность всех возможных n-мерных векторов с вещественными компонентами называется *линейным векторным пространством* R_n. В *эвклидовом* пространстве E_n дополнительно определено скалярное произведение: для $x, y \in E_n$ оно описывается формулой

$$(x, y) = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n.$$

В комплексном пространстве C_n

$$(x, y) = \bar{x}_1 y_1 + \bar{x}_2 y_2 + \ldots + \bar{x}_n y_n$$

(компоненты первого сомножителя[1] заменяются на комплексно сопряженные).

Для оценки матриц и векторов *в целом* применяется понятие нормы. Говорят, что введена *норма* вектора, если каждому вектору x поставлено в соответствие неотрицательное число, обладающее свойствами:

а) $\|x\| > 0$, причем $\|x\| = 0$ тогда и только тогда, когда все элементы вектора x равны нулю;

б) $\|x\| = |c| \cdot \|x\|$, где c — произвольное число;

в) $\|x\| + \|y\| \leq \|x\| + \|y\|$ (свойство треугольника).

Наиболее употребительны три нормы:

$$\begin{array}{lcl} ||x||_1 & = & |x_1| + |x_2| + \ldots + |x_n|; \\ ||x||_2 & = & (|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2)^{1/2}; \\ ||x||_\infty & = & \max_{1 \leq i \leq n} |x_i|. \end{array} \qquad (7.6.1)$$

Обозначения $1, 2, \infty$ объясняются тем, что все три нормы являются частными случаями гильбертова семейства норм:

$$||x||_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{1/p}.$$

Вторая норма есть обычная евклидова длина вектора.

[1] По некоторым источникам — второго.

К нормам *матриц* предъявляется дополнительное требование относительно норм произведения:

$$||AB|| \le ||A|| \cdot ||B||,$$

где A и B — произвольные матрицы, для которых произведение имеет смысл.

Нормы матрицы можно ввести различными способами. Здесь рассмотрим три из них:

- *первая норма* $\|A\|_1 = \max\limits_{1\le i\le m} \sum\limits_{j=1}^{n} |a_{ij}|$ равна максимальной из сумм модулей элементов матрицы *по строкам*;
- *вторая норма* $\|A\|_2 = \max\limits_{1\le j\le n} \sum\limits_{i=1}^{m} |a_{ij}|$ равна максимальной из сумм модулей элементов матрицы *по столбцам*;
- *третья норма* $\|A\|_3 = \sqrt{\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n} a_{ij}^2}$ есть квадратный корень из суммы квадратов элементов матрицы (для матриц, образованных действительными числами).

Можно доказать, что величины $\|A\|_1$, $\|A\|_2$ и $\|A\|_3$ удовлетворяют всем необходимым требованиям и потому являются нормами.

Нормы для матрицы и вектора должны быть *согласованы*: для любых матрицы и вектора должно выполняться

$$||Ax|| \le ||A|| \cdot ||x||.$$

Для этого следует определять

$$||A|| = \max_{x\ne 0} \frac{||Ax||}{||x||}.$$

Такие матричные нормы называются *операторными*, или подчиненными соответствующей норме.

Эвклидова матричная норма для вещественных матриц

$$||A||_E = \Big(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|\Big)^{1/2}$$

согласована с векторной нормой $||x||_2$, но операторной не является.

Вывод любого соотношения между нормами векторов и матриц предполагает *согласованность* этих норм.

7.7. Системы линейных уравнений

Системы линейных алгебраических уравнений возникают при обработке данных; дискретизации линейных дифференциальных задач методом конечных разностей или конечных элементов; решении краевых задач; расчете электрических цепей и сложных гидравлических систем; в некоторых экономических моделях и т. д.

Методы решения систем линейных уравнений делятся на точные и приближенные. *Точные* методы в предположении, что все вычисления ведутся без округлений, за конечное число шагов приводят к точному решению системы. *Приближенные* методы при том же допущении после любого конечного числа шагов дают лишь приближенное решение. К приближенным методам относятся итерационные, а к точным — метод Гаусса.

7.7.1. Матричная запись системы линейных уравнений

Пусть A — матрица вида (n,n), x, b — векторы-столбцы $(n,1)$. Тогда Ax есть вектор-столбец вида $(n,1)$, и матричная запись

$$Ax = b \tag{7.7.1}$$

эквивалентна развернутой форме

$$\begin{array}{llll} a_{11}x_1 & +a_{12}x_2 & +\ldots & +a_{1n}x_n = b_1, \\ a_{21}x_1 & +a_{22}x_2 & +\ldots & +a_{2n}x_n = b_2, \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1}x_1 & +a_{n2}x_2 & +\ldots & +a_{nn}x_n = b_n. \end{array}$$

Если считать A и b известными, то (7.7.1) можно рассматривать как систему из n линейных уравнений с неизвестными $x_1, x_2, \ldots x_n$. Назовем A матрицей коэффициентов, b — вектором правых частей, x — вектором неизвестных. Имеет место

Теорема 7.1 (Крамера). Если определитель $|A|$ матрицы коэффициентов системы из n линейных уравнений с n неизвестными отличен от нуля, то система имеет решение, и притом единственное.

Доказательство. Будем рассматривать систему в матричной записи (7.7.1). Так как $|A| \neq 0$, то A — неособая матрица и для нее существует обратная матрица A^{-1}. Положим $x = A^{-1}b$ и покажем, что x есть решение. Действительно,

$$Ax = A(A^{-1}b) = (AA^{-1})b = Ib = b.$$

Это решение единственное. Если бы существовало другое решение y, то мы имели бы $Ay = b$ и, следовательно, $A^{-1}(Ay) = A^{-1}b$. Но $A^{-1}(Ay) = (A^{-1}A)y = Iy = y$. Значит, $y = A^{-1}b$ и совпадает с $x = A^{-1}b$. ▲

Для того чтобы система (7.7.1) была совместна (имела хотя бы одно решение), необходимо и достаточно, чтобы были равны ранги матрицы коэффициентов A и расширенной матрицы A', получаемой присоединением к A столбца b свободных членов. Если $r = r'$, то решение будет единственным только при $n = r$, т. е. при совпадении числа неизвестных с рангом матрицы A. Если $n > r$, то система имеет множество решений, содержащих $n - r$ произвольных независимых величин.

Пример 7.2. Исследуем систему

$$\begin{array}{rcrcrcl} 2\cdot x_1 & + & 4\cdot x_2 & + & 1\cdot x_3 & = & 3, \\ 1\cdot x_1 & - & 2\cdot x_2 & + & 3\cdot x_3 & = & 2, \\ 4\cdot x_1 & + & 0\cdot x_2 & + & 7\cdot x_3 & = & 5. \end{array}$$

Ранг матрицы коэффициентов $r = 2$, так как

$$\begin{vmatrix} 2 & 4 \\ 1 & -2 \end{vmatrix} = -8 \neq 0,$$

а определитель полной матрицы коэффициентов

$$\begin{vmatrix} 2 & 4 & 1 \\ 1 & -2 & 3 \\ 4 & 0 & 7 \end{vmatrix} = (1/2)\begin{vmatrix} 2 & 4 & 1 \\ 2 & -4 & 6 \\ 4 & 0 & 7 \end{vmatrix} = (1/2)\begin{vmatrix} 2 & 4 & 1 \\ 4 & 0 & 7 \\ 4 & 0 & 7 \end{vmatrix} = 0.$$

При переходе к третьему определителю ко второй строке предыдущего была прибавлена первая, а окончательное заключение следует из равенства элементов двух параллельных рядов.

Ранг расширенной матрицы $r' = 3$ (один из определителей 3-го порядка не равен нулю):

$$\begin{vmatrix} 2 & 4 & 3 \\ 1 & -2 & 2 \\ 4 & 0 & 5 \end{vmatrix} = \begin{vmatrix} 4 & 0 & 7 \\ 1 & -2 & 2 \\ 4 & 0 & 5 \end{vmatrix} = -2\begin{vmatrix} 4 & 7 \\ 4 & 5 \end{vmatrix} = -2(-8) = 16 \neq 0.$$

Поскольку $r \neq r'$, система несовместна.

В дальнейшем будем предполагать, что матрица коэффициентов системы неособая и система имеет единственное решение $x = A^{-1}b$. Для нахождения x по этой формуле нужно вычислить матрицу A^{-1}.

Приведем интересную с точки зрения возможностей современного Фортрана программу решения системы линейных алгебраических уравнений $Ax = b$ по формуле $x = A^{-1}b$ для двух векторов b при известной обратной матрице коэффициентов с помощью матричной функции:

```
program vecfun
   interface
      function mvm(a,b) result (c)
         real a,b,c
         dimension a(:,:),b(:),c(size(a,1))
      end function mvm
   end interface

   real ainv,b1,b2,x1,x2
   dimension ainv(3,3),b1(3),b2(3),x1(3),x2(3)
   data ainv /1.0,2.6,3.4, -1.7,4.4,0.8, 3.6,-4.3,2.7/, &
          b1 /0.8,-4,2.5/, b2 /2.6,0.8,3.7/
   x1=mvm(ainv,b1)
   print *, x1     ! 16.60  -26.27   6.27
   x2=mvm(ainv,b2)
   print *, x2     ! 14.56   -5.63  19.47
end program vecfun

function mvm(a,b) result (c)
   real a,b,c
   dimension a(:,:),b(:),c(size(a,1))
   do i=1,size(a,1)
      c(i)=dot_product(a(i,:),b)
   end do
end function mvm
```

Здесь применена покомпонентная операция скалярного перемножения векторов `dot_product`. Отметим также обязательность в данном случае явного интерфейса.

Если вычислять обратную матрицу согласно данному при определении A^{-1} алгоритму, то для нахождения всех элементов A^{-1} потребуется вычислить n^2 определителей $(n-1)$-го порядка и один определитель n-го порядка. Для вычисления определителя n-го порядка надо вычислить $n!$ произведений по n сомножителей и эти произведения сложить.

Поэтому для нахождения всех элементов обратной матрицы придется проделать умножений

$$n^2(n-1)!(n-2)+n!(n-1)=nn!(n-2)+n!(n-1)=n!(n^2-n-1)\approx(n-1)^2n!.$$

При $n=10$ это число составляет около $300\cdot10^6$. Практически как для расчета определителей, так и для решения систем линейных алгебраических уравнений используются менее трудоемкие методы.

7.7.2. Погрешность прямых решений линейных систем

Точность решения оценивается нормами векторов невязки и ошибки решения. Эти показатели не обязательно малы одновременно. Относительная погрешность

$$\frac{\|\Delta x\|}{\|x\|}\le \mathrm{cond}(A)\frac{\|\Delta b\|}{\|b\|}.$$

То же самое справедливо относительно коэффициентов матрицы. Итак, относительная погрешность решения системы линейных уравнений пропорциональна относительным погрешностям матрицы коэффициентов системы и ее правой части. Константа пропорциональности есть *число обусловленности* $\mathrm{cond}(A)$, которое является мерой близости данной матрицы к множеству вырожденных матриц [37, с. 81–82].

Число обусловленности матрицы

$$\mathrm{cond}A=||A||\cdot||A^{-1}||$$

не может быть меньше единицы. Обусловленность — это *внутреннее* свойство матрицы, не связанное со способом решения системы уравнений. Логарифм числа обусловленности приближенно равен числу значащих цифр, теряемых в решении системы $Ax=b$ по отношению к машинной точности.

Матрицы с большими числами обусловленности называются *плохо обусловленными*. «Что такое хорошо и что такое плохо» — зависит от контекста: размера задачи, требований к точности решения, разрядной сетки машины и др. Решение считается ненадежным, если $\mathrm{cond}A\ge\delta/\varepsilon_{\text{маш.}}$, где δ — допустимая относительная погрешность результата. Легко показать, что масштабированием матрицы число ее обусловленности не изменить.

Классическим примером плохо обусловленной матрицы является матрица Гильберта с элементами

$$h_{ij} = 1/(i+j+1).$$

Уже при $n = 8$ cond $H \geq 10^{10}$, так что полученная при счете в одинарном формате обратная ей матрица может не иметь ни одного верного знака.

Малость невязки $r = b - A\tilde{x}$ плохо обусловленной системы еще не говорит о близости приближенного решения $\tilde{x}$ к точному.

Погрешность обычно пропорциональна наибольшей компоненте вектора решения. Это следует учитывать при существенных различиях в модулях компонент.

Точное значение числа обусловленности требуется редко. Обычно достаточно его разумной оценки. Для ее получения можно обойтись без вычисления обратной матрицы (см. [37, с. 88]). Матрица считается плохо обусловленной, если модуль ее определителя существенно меньше какой-либо из норм матрицы. Линейные системы с квадратными матрицами *некорректны*, если последние вырождаются в пределах точности коэффициентов, т. е. не выполнены условия

$$\|A^{-1}\| \cdot \|\Delta A\| \leq 1.$$

Здесь ΔA — матрица погрешностей коэффициентов. Понятие некорректной постановки задачи существует и в других разделах математики и практически означает крайнюю чувствительность решения задачи к малым изменениям исходных данных. Ошибки машинной реализации особенно опасны для плохо обусловленных систем. Увеличивая длину машинного слова, можно получить машинное решение, достаточно близкое к математическому решению задачи.

Двумерный случай допускает наглядную трактовку понятия обусловленности. Прямые, являющиеся геометрическими образами уравнений, пересекаются под очень острым углом. Небольшое искажение данных (параллельный перенос при изменении свободных членов, поворот прямых при возмущении матрицы коэффициентов) приводит к значительному перемещению точки пересечения, т. е. геометрического образа решения. По этой причине точка пересечения прямых, координаты которой образуют решение системы, определяется с большой ошибкой. Примером плохо обусловленной системы может служить

$$\begin{aligned} 5x + 7y &= 12, \\ 7x + 10y &= 17. \end{aligned} \tag{7.7.2}$$

Ее точное решение есть $x = 1, y = 1$. Однако при подстановке в левые части (7.7.2) значений неизвестных $x = 2.415; y = 0$ получаем $5x+7y = 12.075$ и $7x+10y = 16.905$, т. е. новый набор переменных также оказался приемлемым (по правым частям).

Регуляризация плохо обусловленных линейных систем состоит в замене их задачами вида $A^TAx = A^Tb$, которые эквивалентны минимизации $\|b - Ax\|$.

7.7.3. Расчетная схема метода Гаусса

Опишем метод Гаусса в его простейшей форме (схема «единственного деления»). Рассмотрим систему (7.7.1). Допустим, что $a_{11} \neq 0$. Умножив первое уравнение на $1/a_{11}$, получим первое уравнение преобразованной системы. Умножая полученное уравнение соответственно на $a_{21}, a_{31}, \ldots, a_{n1}$ и вычитая результаты почленно из 2-го, 3-го, ..., n-го уравнений системы (7.7.1), получаем преобразованную систему вида

$$\begin{array}{rrrrrr} x_1 & +a'_{12}x_2 & +a'_{13}x_3 & +\ldots & +a'_{1n}x_n & = b'_1; \\ & a'_{22}x_2 & +a'_{23}x_3 & +\ldots & +a'_{2n}x_n & = b'_2; \\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & & & \\ & a'_{n2}x_2 & +a'_{n3}x_3 & +\ldots & +a'_{nn}x_n & = b'_n. \end{array} \qquad (7.7.3)$$

Для перехода от (7.7.1) к (7.7.3) требуется $n(n+1)$ умножений и столько же вычитаний.

Если $a_{2,2} \neq 0$, то система (7.7.3) аналогично преобразуется к новой, где $a''_{22} = 1$, а для всех $k > 2$ будет $a''_{k2} = 0$. Продолжая этот процесс, придем к системе

$$\begin{array}{rrrrrr} x_1 & +a'_{12}x_2 & +a'_{13}x_3+ & \ldots & +a'_{1n}x_n & = b'_1; \\ & x_2 & +a''_{23}x_3+ & \ldots & +a''_{2n}x_n & = b''_2; \\ & & x_3+ & \ldots & +a'''_{2n}x_n & = b'''_3; \\ & & & \ldots\ldots\ldots\ldots\ldots\ldots & & \\ & & & & x_n & = b_n^{(n)}. \end{array} \qquad (7.7.4)$$

Преобразование (7.7.1) в систему (7.7.4) уравнений с треугольной матрицей коэффициентов называется *прямым* ходом метода Гаусса. Для прямого хода требуется умножений

$$\begin{array}{rcl} n(n+1) + (n-1)n + \ldots + 1 \cdot 2 & = & (n^2+n) + [(n-1)^2 + (n-1)] + \ldots \\ +(2^2+2) + (1^2+1) & = & (1^2 + 2^2 + \ldots + n^2) + (1 + 2 + \ldots + n) \\ = n(n+1)(2n+1)/6 & + & n(n+1)/2 \\ = n(n+1)/2((2n+1)/3+1) & = & n(n+1)(n+2)/3. \end{array}$$

Итак, умножений и вычитаний в прямом ходе будет $2n(n+1)(n+2)/3$.

Последнее уравнение системы (7.7.4) непосредственно дает нам x_n. Зная x_n, из предпоследнего уравнения (7.7.4) определяем x_{n-1} и т. д., пока не будет вычислен x_1. Эти вычисления называются *обратным* ходом метода Гаусса; для его выполнения нужно произвести $n(n-1)/2$ умножений и столько же вычитаний. Трудоемкость метода в основном определяется прямым ходом. Если $n = 10$, то общее число операций

$$N = \frac{2 \cdot 10 \cdot 11 \cdot 12}{3} + 10 \cdot 9 = 880 + 90 \approx 1000,$$

т. е. примерно в 300 тыс. раз меньше, чем при использовании формулы Крамера.

Следует отметить, что полученная выше оценка указывает количество только арифметических операций. В ней не учтены операции формирования и управления, необходимые для машинной реализации метода Гаусса. Анализ «бесхитростной» программы, некогда составленной автором на языке Алгол-60 и переведенной Альфа-транслятором в коды ЦВМ типа М-220 при конкретном значении n, для полной трудоемкости метода дал оценку $(8n^3+63n^2+277n-105)/6$ операций. Асимптотически (при достаточно большом n) она оказывается ровно вдвое выше обычно указываемой — $2n^3/3$. Современные ЭВМ благодаря более развитой системе команд и большому числу индексных регистров позволяют создавать более компактные машинные программы, но и для них неизбежны дополнительные издержки на управление циклами.

Практически метод Гаусса для повышения точности вычислений всегда реализуется с выбором на каждом этапе прямого хода главного (наибольшего по модулю) элемента-делителя и соответствующей перестановкой строк. Гауссово исключение с частичным выбором главного элемента гарантированно дает малые невязки. Трудоемкость выбора главных элементов зависит не только от размерности системы, но и от элементов матрицы — более того, даже от порядка следования строк.

Ниже приведены процедура решения системы линейных уравнений методом Гаусса и вызывающая ее главная программа.

```
program tgauss
    interface
       subroutine gauss(a,i0,x)
          real,dimension (1:,1:) :: a
          real x(i0:)
          integer i0
```

```
        end subroutine gauss
      end interface
      real  a(0:2,4),x(3),y(3)
      integer i

      data a/1,2,3,1,3,5,1,4,6,1,2,4/
      data y/0,2,-1/

      call  gauss(A,1,X)
      write (*,1) '   Solution is','By gauss'
    1 format (7X,A,10X,A)
      write (*,2) (y(i),x(i), i=1,3)
    2 format (1X,2e20.5)
  end !! tgauss

  subroutine gauss(a,i0,x)
!****************************************/
!* Решение системы линейных уравнений   */
!*         методом Гаусса               */
!* с выбором главного элемента по строкам */
!****************************************/
      real,dimension (1:,1:) :: a
      real x(i0:)            !  Ответ
      integer i0             !  Начальный индекс ответа

      real             max,p
      integer          f,i,j,l,m,n,t

      n=size(a,1); m=n+1
      do i=1,n
         max=abs(a(i,i)); t=i
         do j=i+1,n
            p=abs(a(j,i))
            if (p>max) then
               max=p; t=j
            end if
         end do
         if (max<1e-7) then
            print *, 'Matrix is singular'; stop
         end if
```

```
         if (t/=i) then
            do j=i,m
               p=a(i,j); a(i,j)=a(t,j); a(t,j)=p
            end do
         end if
         do j=m,i+1,-1
            a(i,j)=a(i,j)/a(i,i)
         end do
         do f=i+1,n
            do j=i+1,m
               a(f,j)=a(f,j)-a(f,i)*a(i,j)
            end do
         end do
      end do ! Прямого хода
      do i=n,1,-1
         x(i-1+i0)=a(i,m)
         do l=i-1,1,-1
            a(l,m)=a(l,m)-a(l,i)*x(i-1+i0)
         end do
      end do
  end subroutine gauss
```

Решение (с учетом погрешности представления данных для нулевой компоненты) совпало с заданным вектором y.

7.7.4. Уточнение решений

По малости полученного вектора $\xi^{(0)} = b - Ax^{(0)}$ можно (с осторожностью) судить о близости найденного решения к точному. Если невязка $\xi^{(0)}$ велика, то следует искать такую векторную поправку Δ, что $x^{(0)} + \Delta$ есть точное решение системы:

$$A(x^{(0)} + \Delta) = b.$$

Последнее уравнение равносильно уравнению

$$A\Delta = \xi^{(0)}.$$

Итак, нахождение поправки сводится к решению системы уравнений с другими свободными членами, но с прежней матрицей коэффициентов. Заново выполнять ее треугольное разложение необходимости нет.

7.7.5. Применение метода Гаусса для вычисления определителя

При решении системы (7.7.1) методом Гаусса мы заменяли исходную матрицу A треугольной матрицей C:

$$A = \begin{bmatrix} 1 & c_{12} & c_{13} & \dots & c_{1n} \\ 0 & 1 & c_{23} & \dots & c_{2n} \\ 0 & 0 & 1 & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}.$$

Элементы этой матрицы получались из элементов исходной матрицы A и последующих промежуточных матриц в результате операций:

- деления строки на «ведущие» элементы $a_{11}, a'_{22}, a''_{33}, \dots, a_{nn}^{(n-1)}$, которые предполагались ненулевыми;
- вычитания из нижележащих строк отмеченной строки, умноженной на постоянный для каждой строки множитель.

Первая из этих операций приводит к делению определителя матрицы на ведущий элемент, а вторая величины определителя не меняет. Следовательно,

$$\det C = \frac{1}{a_{11}a'_{22}a''_{33}\dots, a_{nn}^{(n-1)}} \det A.$$

Очевидно, что $\det C = 1$; значит, определитель матрицы равен произведению ведущих элементов схемы Гаусса.

7.7.6. Применение метода Гаусса для вычисления обратной матрицы

Пусть задана квадратная матрица A порядка n, причем $|A| \neq 0$. Тогда существует обратная матрица

$$A^{-1} = A' = \begin{bmatrix} a'_{11} & a'_{12} & \dots & a'_{1n} \\ a'_{21} & a'_{22} & \dots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a'_{n1} & a'_{n2} & \dots & a'_{nn} \end{bmatrix},$$

где $a'_{ij} = A_{ji}/|A|$.

Выберем из обратной матрицы j-й столбец $[a'_{1j}, a'_{2j}, \dots a'_{nj}]^T$. Если образовать произведение i-й строки исходной матрицы A на j-й столбец матрицы $A' = A^{-1}$, то сумма парных произведений

$$\sum_{k=1}^{n} a_{ik} a'_{kj} = \begin{cases} 0 & \text{при } i \neq j; \\ 1 & \text{при } i = j, \end{cases}$$

поскольку $AA^{-1} = I$. Таким образом, для нахождения элементов j-го столбца обратной матрицы нужно решить систему линейных уравнений с матрицей коэффициентов A и столбцом свободных членов с единицей в j-й строке и нулями в остальных. Всего необходимо решить n систем линейных уравнений с общей матрицей коэффициентов при неизвестных и различными столбцами свободных членов. Решать такие системы целесообразно методом Гаусса.

Для экономии вычислений можно одновременно преобразовывать по схеме Гаусса матрицу A и *все* правые части. Один раз выполнив прямой ход и n раз — обратный ход Гаусса, получим таблицу из n^2 чисел, образующих обратную матрицу.

7.8. Метод прогонки для трехдиагональных матриц

Будем искать решение такой (ленточной) системы, каждое уравнение которой связывает три «соседних» неизвестных:

$$a_i x_{i-1} + cb_i x_i + c_i x_{i+1} = d_i, \tag{7.8.1}$$

где $i = \overline{1,n}$; $a_1 = 0$, $c_n = 0$. Запишем систему уравнений в виде

$$\begin{array}{rrrl}
b_1 x_1 & +c_1 x_2 & +0 & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots = d_1, \\
a_1 x_1 & +b_2 x_2 & +c_2 x_3 & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots = d_2, \\
0 & +a_2 x_3 & +b_3 x_3 & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots = d_3, \\
0 & 0 & +a_3 x_3 & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots = d_4, \\
\dots\dots & \dots\dots & \dots\dots & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
 & & a_{n-1} x_{n-2} & +b_{n-1} x_{n-1} \quad +c_{n-1} x_n = d_{n-1}, \\
 & & & a_n x_{n-1} \quad +b_n x_n = d_n.
\end{array}$$

На главной диагонали матрицы здесь стоят элементы $b_1, b_2, \dots, b_n$, над ней — $c_1, c_2, \dots, c_{n-1}$, под ней — $a_2, a_3, \dots, a_n$. Как правило, все элементы главной диагонали ненулевые. Задачи этого типа встречаются при

решении систем дифференциальных уравнений в частных производных, при аппроксимации сплайнами и т. п.

Аналогично методу Гаусса прогонка имеет прямой и обратный ход. Прямая прогонка состоит в том, что каждое неизвестное x_i выражается через x_{i+1} с помощью прогоночных коэффициентов $\{A_i, B_i\}$:

$$x_i = A_i x_{i+1} + B_i, \qquad i = \overline{1, n-1}. \tag{7.8.2}$$

Из первого уравнения исходной системы найдем

$$x_1 = -\frac{c_1}{b_1} x_2 + \frac{d_1}{b_1}.$$

Ясно, что

$$A_1 = -c_1/b_1, \quad B_1 = d_1/b_1.$$

Далее,

$$a_2(A_1 x_2 + B_1) + b_2 x_2 + c_2 x_3 = d_2.$$

Отсюда

$$x_2 = \frac{-c_2 x_3 + d_2 - a_2 B_1}{a_2 A_1 + b_2},$$

так что

$$A_2 = -c_2/e_2, \quad B_2 = (d_2 - a_2 B_1)/e_2, \quad e_2 = a_2 A_1 + b_2.$$

Можно показать, что для любого номера $i \geq 2$

$$A_i = -c_i/e_i, \quad B_i = (d_i - a_i B_{i-1})/e_i, \quad e_i = a_i A_{i-1} + b_i.$$

Последний x найдем согласно

$$x_n = \frac{d_n - a_n B_{n-1}}{b_n + a_n A_{n-1}}.$$

Обратная прогонка состоит в последовательном вычислении неизвестных от бóльших индексов к меньшим согласно (7.8.2) с использованием найденных на этапе прямой прогонки коэффициентов $\{A_i, B_i\}$.

Трудоемкость метода прогонки в [15, с. 88] оценивается как $O(5n)$.

7.9. Итерационные методы решения систем линейных уравнений

7.9.1. Метод простой итерации

Пусть дана система из уравнений с n неизвестными, записанная в матричной форме:

$$A_1 x = b_1. \tag{7.9.1}$$

Допустим, что каким-либо способом найдена равносильная ей система

$$x = Ax + b. \tag{7.9.2}$$

В последующем будем обозначать через $x_i^{(k)}$ i-ю составляющую вектора $x^{(k)}$, где k — номер приближения. Пусть $x^{(0)}$ — произвольный вектор. Положив

$$x^{(k+1)} = Ax^{(k)} + b, \qquad k = 0, 1, \ldots, \tag{7.9.3}$$

получим последовательность векторов $x^{(1)}, x^{(2)}, \ldots x^{(k)}, \ldots$ Пусть при $k \to \infty$ компонента $x_i^{(k)}$ для каждого $i = \overline{1,n}$ имеет конечный предел $\hat{x}_i$. Вектор с компонентами $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n$ обозначим через $\hat{x}$ и будем записывать $x^{(k)} \to \hat{x}$. Тогда $\hat{x}$ является решением системы. В самом деле, левая часть (7.9.3) имеет пределом $\hat{x}$, а правая — $A\hat{x} + b$. Это означает, что $\hat{x}$ служит решением (7.9.2). Вектор $x^{(k)}$ называется k-й *итерацией* вектора $x^{(0)}$.

Может случиться, что хотя бы для одного i конечного предела $x_i^{(k)}$ при $k \to \infty$ не существует. Тогда процесс итераций считается расходящимся.

Достаточные условия сходимости этого процесса (метода Якоби) связаны непосредственно с исходной матрицей:

Теорема 7.2. Если какая-нибудь норма $\|A\|$ матрицы A удовлетворяет условию

$$\|A\| < 1, \tag{7.9.4}$$

то процесс итерации для системы (7.9.2) сходится к единственному решению при любом начальном приближении.

Для оценки нормы погрешности n-й итерации существуют формулы, аналогичные рассмотренным в главе 6:

$$\|\hat{x} - x^{(k)}\| \le \frac{\|A\|}{1 - \|A\|}\|x^{(k)} - x^{(k-1)}\|, \qquad k = 1, 2, \ldots \tag{7.9.5}$$

Этот же результат может быть выражен через норму разности начальных итераций:

$$\|\hat{x} - x^{(k)}\| \le \frac{\|A\|^k}{1 - \|A\|}\|x^{(1)} - x^{(0)}\|, \qquad (7.9.6)$$

где $\hat{x}$ — точное решение. Применение формул (7.9.5) или (7.9.6) предполагает использование нормы, для которой выполнено условие сходимости итераций (7.9.4), и пренебрежимую малость погрешностей округления.

При $\|A\| \le 1/2$ множитель $\|A\|/(1 - \|A\|) \le 1$; в этом случае итерации можно прекратить при выполнении условия $\|x^{(k)} - x^{(k-1)}\| \le \delta$ и положить $\hat{x} \approx x^{(k)}$.

7.9.2. Метод Зейделя

В отличие от метода простой итерации, в методе Зейделя вычисления проводятся по схеме

$$\begin{array}{llllllll}
x_1^{(k+1)} & = a_{1,1}x_1^{(k)} & +a_{1,2}x_2^{(k)} & +a_{1,3}x_3^{(k)} & +\ldots & +a_{1,n}x_n^{(k)} & +b_1; \\
x_2^{(k+1)} & = a_{2,1}x_1^{(k+1)} & +a_{2,2}x_2^{(k)} & +a_{2,3}x_1^{(k)} & +\ldots & +a_{2,n}x_n^{(k)} & +b_2; \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
x_n^{(k+1)} & = a_{n,1}x_1^{(k+1)} & +a_{n,2}x_2^{(k+1)} & +a_{n,3}x_3^{(k+1)} & +\ldots & +a_{n,n}x_n^{(k)} & +b_n.
\end{array}$$

Иначе говоря, только что полученная компонента новой итерации сразу же используется для расчета последующих компонент этой итерации.

Достоинством метода Зейделя является экономия n ячеек памяти для x: ответ обновляется покомпонентно, так что не требуется сохранять старые результаты $\{x^{(k-1)}\}$ до окончательного формирования $\{x^{(k)}\}$.

Необходимое и достаточное условие сходимости процесса Зейделя: модули всех корней уравнения, получаемого после раскрытия определителя в левой части равенства

$$\begin{vmatrix}
a_{11} - \lambda & a_{12} & a_{13} & \ldots & a_{1n} \\
a_{21}\lambda & a_{22} - \lambda & a_{23} & \ldots & a_{2n} \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
a_{n1}\lambda & a_{n2}\lambda & \ldots & a_{n,n-1}\lambda & a_{nn} - \lambda
\end{vmatrix} = 0, \qquad (7.9.7)$$

должны быть меньше единицы. Поскольку это условие отлично от условия теоремы 7.2, могут быть ситуации, в которых простые итерации расходятся, а зейделевы сходятся, и наоборот. Однако выполнение *достаточных* условий сходимости простых итераций (см. теорему 7.2) обеспечивает и сходимость процесса Зейделя.

Оценку погрешности n-й итерации метода Зейделя мы проведем только по первой норме. Пусть условие $\|A_1\|$ выполнено. В этом случае

$$\|\bar{x} - x^{(k)}\|_1 \le \frac{\mu}{1-\mu}\|x^{(k)} - x^{(k-1)}\|_1 \tag{7.9.8}$$

или

$$\|\bar{x} - x^{(k)}\|_1 \le \frac{\mu^k}{1-\mu}\|x^{(1)} - x^{(0)}\|_1. \tag{7.9.9}$$

Здесь

$$\mu = \max_i \frac{\sum\limits_{j=i}^{n} |a_{i,j}|}{1 - \sum\limits_{j=1}^{i-1} |a_{i,j}|}.$$

Поскольку $\sum\limits_{j=1}^{i-1} |a_{i,j}| + \sum\limits_{j=i}^{n} |a_{i,j}| \le \|A\|_1 < 1$, имеет место

$$\sum_{j=i}^{n} |a_{i,j}| \le \|A\|_1 - \sum_{j=1}^{i-1} |a_{i,j}|,$$

а потому

$$\frac{\sum\limits_{j=i}^{n} |a_{i,j}|}{1 - \sum\limits_{j=1}^{i-1} |a_{i,j}|} \le \frac{\|A\|_1 - \sum\limits_{j=1}^{i-1} |a_{i,j}|}{1 - \sum\limits_{j=1}^{i-1} |a_{i,j}|} < \frac{\|A\|_1 - \|A\|_1 \sum\limits_{j=1}^{i-1} |a_{i,j}|}{1 - \sum\limits_{j=1}^{i-1} |a_{i,j}|} = \|A\|_1.$$

Следовательно, $\mu < \|A\|_1$, и при $\|A\|_1 < 1$ из (7.9.9) следует сходимость итераций по Зейделю. Но неравенства (7.9.5)–(7.9.6) при замене $\|A\|_1$ на μ совпадают с неравенствами (7.9.8)–(7.9.9). Значит, при $\|A\|_1 < 1$ метод Зейделя имеет лучшую оценку погрешности итераций по первой норме. Метод Зейделя может обеспечить и фактическое ускорение сходимости (см. пример 7.3).

Для зейделевой итерации оценка по m-норме

$$|x_l - x_l^{(k)}| \le \frac{\mu^k}{1-\mu} \max_j |x_j^{(1)} - x_j^{(0)}|.$$

Здесь

$$\mu = \max_l \left\{ \Big(\sum_{j=i}^{n} |\alpha_{ij}| \Big) \Big/ \Big(1 - \sum_{j=1}^{i-1} |\alpha_{ij}| \Big) \right\} \le ||\alpha||_m.$$

При реализации метода Зейделя имеет смысл после полного тура переупорядочить уравнения по убыванию модулей поправок предыдущего тура.

Метод Зейделя, как и метод Якоби, обладает медленной сходимостью, и его нецелесообразно применять для решения систем высокого порядка и плохо обусловленных. Для частных случаев (например, для симметричных матриц) можно построить более быстрые схемы. Обычно метод Зейделя дает лучшую сходимость, чем метод простой итерации. Однако в принципе возможны и обратные ситуации (и даже расходимость зейделевой итерации при сходимости простой).

7.9.3. Обеспечение сходимости итераций

Линейные системы могут быть приведены к виду, обеспечивающему сходимость итерационного процесса решения, если они имеют доминирующие диагональные элементы. Для этого из заданной системы выделяют уравнения с коэффициентами, модули которых больше сумм модулей остальных коэффициентов уравнения:

$$|a_{ii}| > \sum_{j \neq i} |a_{jj}|, \quad i = \overline{1,n}. \tag{7.9.10}$$

Каждое выделенное уравнение записывается в строку результирующей системы с таким номером, чтобы отмеченный коэффициент оказался диагональным. Из оставшихся неиспользованными и выделенных уравнений составляют линейно независимые комбинации с целью максимизации модулей остальных диагональных коэффициентов. При этом каждое из оставшихся уравнений должно войти минимум в одну комбинацию.

Разрешая каждое i-е уравнение системы (7.7.1) относительно x_i, получаем

$$\begin{array}{lllllll} x_1 = & 0 \cdot x_1 & +a'_{12}x_2 & +a'_{1,3}x_3 & +\ldots & +a'_{1n}x_n & +b'_1; \\ x_2 = & a'_{21}x_1 & +0 \cdot x_2 & +a'_{13}x_3 & +\ldots & +a'_{2n}x_n & +b'_2; \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ x_n = & a'_{n1}x_1 & +a'_{n2}x_2 & +a'_{13}x_3 & +\ldots & +0 \cdot x_n & +b'_n, \end{array} \tag{7.9.11}$$

где

$$a'_{ij} = -\frac{a_{ij}}{a_{ii}} \quad (j \neq i); \quad b'_i = \frac{b_i}{a_{ii}}.$$

В силу неравенств (7.9.10) имеем

$$\sum_{j=1}^{n} |a'_{ij}| \le \sum_{j \ne i} \left| -\frac{a_{ij}}{a_{ii}} \right| = \frac{1}{a_{ii}} \sum_{j \ne i} |a_{ij}| < 1.$$

Следовательно, первая норма $\|A\|_1 < 1$ и итерации, определяемые системой (7.9.11), будут сходиться.

Пример 7.3. Рассмотрим систему

$$\begin{array}{rrrr} 5x_1 & +3x_2 & -x_3 & = 15; \\ x_1 & -4x_2 & +2x_3 & = -2; \\ 2x_1 & +6x_2 & -10x_3 & = -18 \end{array}$$

с решением $x_1 = 2, x_2 = 3, x_3 = 4$. Нетрудно видеть, что условия (7.9.10) для нее выполнены, и полученная из этой системы запись обеспечивает сходящийся итерационный процесс. В табл. 7.1 показан ход процесса для метода простой итерации и метода Зейделя при начальных значениях $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 1$.

Таблица 7.1. Решение системы из примера 7.3 итерационными методами

Номер шага	Зейделевы			Простые		
	x_1	x_2	x_3	x_1	x_2	x_3
1	2.6000	1.6500	3.3100	2.6000	1.2500	2.6000
2	2.6720	2.8230	4.0282	2.7700	2.4500	3.0700
3	2.1118	3.0421	4.0476	2.1440	2.7275	3.8240
4	1.9843	3.0199	4.0088	2.1283	2.9480	3.8653
5	1.9898	3.0018	3.9991	2.0043	2.9647	3.9945
6	1.9987	2.9992	3.9993	2.0200	2.9983	3.9797
7	2.0003	2.9997	3.9999	1.9970	2.9949	4.0030
8	2.0001	3.0000	4.0000	2.0037	3.0007	3.9963
9	2.0000	3.0000	4.0000	1.9988	2.9991	4.0012

Точность 0.0001 в методе простой итерации достигается только на 14-м шаге.

7.9.4. Сверхрелаксация

Общая идея метода релаксации (ослабления) состоит в уменьшении на каждом шаге очередной невязки изменением соответствующей компоненты приближения.

Пусть на очередном k-м шаге получено новое приближение к i-й компоненте решения $\tilde{x}_i^{(k)}$. Представим результат шага в виде

$$x_i^{(k)} = x_i^{(k-1)} + \omega(\tilde{x}_i^{(k)} - x_i^{(k-1)}),$$

где ω — параметр релаксации. При $\omega = 1$ метод эквивалентен методу Зейделя. В методе *верхней сверхрелаксации* ω выбирается из интервала $(1, 2)$, т. е. коррекция может быть сделана «более энергичной» и сходимость ускорится. Оптимальный выбор зависит от задачи, но никогда не превосходит двух. Обычно рекомендуется $\omega \approx 1.9$ [95, с. 346]. К сожалению, расчет теоретически оптимального значения ω оказывается слишком сложным («лекарство хуже болезни»).

Еще одной модификацией метода Зейделя является выбор очередного уравнения в порядке убывания модуля невязки. Можно также менять количество одновременно уточняемых компонент [68, с. 87]. Считаются перспективными *двухшаговые* методы, где очередное приближение определяется через два предыдущих.

7.10. Итерационный способ обращения матриц

Если норма матрицы $B = I - A$ мала, то обратная к A матрица

$$A^{-1} = (I - B)^{-1} = I + B + B^2 + \ldots$$

в принципе может быть найдена сколь угодно точно приближенным суммированием данного степенного ряда. Правда, сходимость будет медленной.

Попытаемся построить процесс, сходящийся более быстро [15, с. 125]. Будем обозначать очередные приближения через $\{U_k\}$, $k = 0, 1, \ldots$ Поскольку должно выполняться $AU = I$, разность $D_k = I - AU_k$ можно рассматривать как матричную невязку для $\{U_k\}$. В частности, из $D_0 = I - AU_0$ следует $AU_0 = I - D_0$. Тогда $A = (I - D_0)U_0^{-1}$, а обратная ей

$$A^{-1} = U_0(I - D_0)^{-1} = U_0(I + D_0 + D_0^2 + \ldots).$$

Будем считать первым приближением к A^{-1} порядка m матрицу

$$U_1 = U_0(I + D_0 + D_0^2 + \ldots + D_0^m).$$

Тогда невязка этого приближения выразится через невязку предыдущего формулой

$$\begin{aligned} D_1 &= I - AU_1 = I - AU_0(I + D_0 + D_0^2 + \ldots + D_0^m) \\ &= I - (I - D_0)(I + D_0 + D_0^2 + \ldots + D_0^m) \\ &= I - (I - D_0^{m+1}) = D_0^{m+1}. \end{aligned}$$

Практически ограничиваются случаем $m = 1$ и полагают

$$\begin{aligned} D_k &= I - AU_k, \\ U_{k+1} &= U_k(I + D_k). \end{aligned}$$

Этот алгоритм был проверен на коррекции обратной матрицы из примера 7.1. В качестве исходной была взята

$$U_0 = \begin{bmatrix} 0.72667 & -0.92000 & -0.56667 \\ 0.04000 & 0.39333 & 0.41333 \\ -0.31333 & 1.04000 & 0.39333 \end{bmatrix}.$$

Результаты третьего шага в пяти знаках совпали с приведенными в том же примере теоретическими значениями. Область сходимости этого метода весьма мала, однако он может использоваться на заключительных шагах процессов минимизации близких к квадратичным функций многих переменных.

7.11. Сравнительная оценка точных и итерационных методов

В заключение дадим сравнительную оценку точных и итерационных методов решения систем линейных уравнений. Обычно ограничиваются подсчетом операций умножения и деления, поскольку сложения и вычитания выполняются намного быстрее. Можно сказать, что вычислительные затраты на операции умножения и деления в методе Гаусса имеют порядок $O(n^3/3)$, в методе квадратных корней $O(n^3/6)$, в методе вращений $O(4n^3/3)$, в методе прогонки — $O(5n)$ [15, с. 88].

Если итерации сходятся достаточно быстро (число итераций меньше $n/3$), то метод итераций требует меньшего объема вычислений. В то же время применение итерационных методов предполагает предварительную проверку (а если нужно, и обеспечение) условий сходимости и

подбор «хороших» начальных приближений. Для применения точных методов подобная подготовка не нужна. При увеличении размерности системы (например, в процессе приближенного решения бесконечных систем) нормы матрицы увеличиваются, так что сходимость итераций замедляется.

Итерационные методы решения систем линейных уравнений эффективны для разреженных (например, ленточных) матриц. Они:

- не требуют хранения всей матрицы;
- исключают обработку значительной части нулей;
- накапливают погрешность только в пределах итерации;
- могут использоваться для уточнения ранее полученных решений.

В процессе их реализации матрица не меняется — следовательно, сохраняется ее структура. Они безусловно предпочтительны, когда учитывают специфику задачи. Логическая структура итерационных методов значительно проще, реализующие их на ЦВМ программы короче.

Решение уравнений методом простой итерации допускает параллельный расчет новых значений всех неизвестных на каждом шаге итерации и открывает путь к сокращению времени решения задач большой размерности при использовании вычислительных систем с несколькими вычислительными устройствами (мультипроцессорных). Проблема заключается в создании средств *автоматического* распараллеливания программ как на уровне их разработки, так и в процессе счета.

Глава 8. Дополнительные разделы линейной алгебры

8.1. Дополнительные сведения о матрицах

8.1.1. Транспонирование комплексных матриц

Приведем дополнительные справочные сведения относительно векторов и матриц с комплексными элементами.

Транспонирование матрицы $a^*_{ij} = \overline{a_{ji}}$ (черта означает замену соответствующих элементов на комплексно сопряженные).

Эрмитовы матрицы отвечают условию $A^* = A$, для косоэрмитовых $K^* = -K$.

8.1.2. Ортогональные матрицы

Действительная матрица называется *ортогональной*, если ее транспонированная матрица совпадает с обратной, т. е.

$$\begin{aligned} A^T &= A^{-1}, \\ AA^T &= A^T A = I. \end{aligned} \tag{8.1.1}$$

Из этих базовых соотношений легко выводятся свойства ортогональной матрицы:

- Ее строки (столбцы) попарно ортогональны.
- Сумма квадратов элементов каждого ряда равна единице.
- Определитель равен ± 1.

- Произведение двух ортогональных матриц ортогонально:

$$(AB)^T AB = B^T A^T AB = B^T IB = I.$$

Комплексная матрица именуется *унитарной* в случае $U^*U = I$, $U^* = U^{-1}$. Столбцы, строки, собственные векторы таких матриц являются ортонормированными.

Матрица H называется правой (верхней) *матрицей Хессенберга*, если $h_{ij} = 0$ при $i > j + 1$. Она в дополнение к верхней треугольной матрице имеет дополнительную ненулевую диагональ (ниже главной). Аналогично определяется левая (нижняя) матрица Хессенберга.

8.2. Линейные векторные пространства

8.2.1. Линейная зависимость векторов

Любая совокупность n-мерных векторов, рассматриваемая с не выводящими за пределы этой совокупности операциями сложения векторов и умножения вектора на число, называется *линейным векторным пространством*.

Векторы $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ такого пространства называются *линейно зависимыми*, если существуют не все равные нулю числа $c_1, c_2, \ldots, c_n$, для которых

$$c_1 x^{(1)} + c_2 x^{(2)} + \ldots + c_n x^{(n)} = 0.$$

В этом случае один из них является линейной комбинацией остальных. Для трехмерного пространства линейная зависимость системы из двух векторов означает, что они параллельны некоторой прямой, а трех векторов — что они параллельны некоторой плоскости. Если же нулевой вектор получается только при нулевом наборе $\{c_i\}$, то векторы линейно независимы.

Пусть имеется система из m n-мерных векторов $\{x_i\}$. Тогда для определения коэффициентов $\{c_i\}$ имеем систему уравнений

$$\begin{array}{llll} c_1 x_1^{(1)} & +c_2 x_1^{(2)} & +\ldots & +c_m x_1^{(m)} = 0, \\ c_1 x_2^{(1)} & +c_2 x_2^{(2)} & +\ldots & +c_m x_2^{(m)} = 0, \\ \ldots & \ldots & \ldots & \ldots \\ c_1 x_n^{(1)} & +c_2 x_n^{(2)} & +\ldots & +c_m x_n^{(m)} = 0. \end{array}$$

Очевидно, что она имеет нулевое решение. Ранг матрицы X, составленной из компонент $\{x_i\}$, не может превосходить m. Если он равен m, то нулевое решение — единственное и векторы линейно независимы.

8.2.2. Скалярные произведения и ортогональность векторов

Введем для двух векторов в комплексном (вещественном) пространстве R *скалярное произведение* $(x,y)=\sum\limits_{i=1}^{n} x_i y_i$ (в комплексном случае $(x,y)=\sum\limits_{i=1}^{n} x_i \bar{y}_i$, где черта означает переход к комплексно сопряженному числу). Предполагается, что:

1) $(x,y)=\overline{(y,x)}$;
2) $(x,x)=0$ только при $x=0$, иначе $(x,x)>0$;
3) $(x+y,z)=(x,z)+(y,z)$;
4) $(\alpha x,y)=\alpha(x,y)$.

Скалярное произведение двух векторов удовлетворяет неравенству Коши—Буняковского

$$|(x,y)|\le |x|\cdot|y|.$$

Вещественное пространство с определенным таким образом скалярным произведением называется *евклидовым пространством* E_n. Угол между ненулевыми векторами x и y определяется из формулы

$$\cos\varphi=\frac{(x,y)}{|x|\cdot|y|}.$$

Два вектора x и y пространства E_n называются *ортогональными*, если их скалярное произведение $(x,y)=0$. Система векторов $x_1,x_2,\dots x_k$ называется ортогональной, если два любых различных вектора ее ортогональны друг другу. Если вектор ортогонален векторам $x_1,x_2,\dots x_k$, то он ортогонален и любой линейной комбинации их («натянутому на них подпространству»).

Длина вектора определяется согласно

$$|x|=\sqrt{(x,x)}=\Big(\sum_{i=1}^{n} x_i^2\Big)^{1/2}.$$

8.2.3. Базис линейного векторного пространства

Любая совокупность n линейно независимых векторов n-мерного пространства называется его *базисом*. Каждый вектор этого пространства может быть представлен в виде линейной комбинации векторов базиса, и притом единственным образом.

Базис пространства E_n называется ортогональным, если его векторы попарно ортогональны. Если они к тому же имеют единичную длину, базис считается *ортонормированным*. Простейший случай ортонормированного базиса — орты координатных осей.

8.2.4. Линейное преобразование векторов

Умножение вектора на матрицу

$$y = Ax$$

можно рассматривать как *оператор* линейного преобразования вектора. Оператор A^* называется *сопряженным* с A, если для любых векторов $x, y \in E_n$ выполняется равенство

$$(Ax, y) = (x, A^*y).$$

Напомним, что * означает транспонирование матрицы с одновременной заменой ее элементов на комплексно сопряженные. В случае симметричной вещественной матрицы обсуждаемый оператор оказывается *самосопряженным*.

8.2.5. Преобразование координат при изменении базиса

Пусть $e_1, e_2, \ldots e_n$ и $f_1, f_2, \ldots f_n$ — два базиса одного и того же линейного пространства E_n. Каждый вектор второго (нового) базиса имеет в старом координаты

$$f_j = s_{1j}e_1 + s_{2j}e_2 + \ldots + s_{nj}e_n, \quad j = \overline{1,n}, \tag{8.2.1}$$

а в новом — $\{\xi_j\}$. Неособая матрица S с элементами $\{s_{ij}\}$ называется *матрицей перехода* от нового базиса к старому. Можно показать, что вектор x в старом базисе равен матрице перехода, умноженной на вектор в новом базисе: $x = S\xi$.

Соответственно,

$$\xi = S^{-1}x.$$

8.3. Матричные разложения

Многие методы вычислительной линейной алгебры сводятся к выполнению последовательностей элементарных преобразований строк и/или столбцов матрицы решаемой задачи и (в случае линейной системы уравнений) компонент правой части. Эти преобразования можно интерпретировать как умножение исходной матрицы (или обеих частей системы линейных уравнений) на матрицы специального вида. Язык специальных матриц является удобным средством компактного описания алгоритмов линейной алгебры.

Важную роль в вычислительной алгебре играют ортогональные (унитарные) подобные преобразования, т. е. левые, правые и двусторонние умножения данной матрицы на ортогональные (унитарные). Они сохраняют спектральную и евклидову нормы исходной. Всякая нормальная матрица унитарно-подобна диагональной матрице, а эрмитова — матрице с вещественной диагональю.

8.3.1. Матрицы перестановок

Квадратная $n \times n$ матрица P называется матрицей перестановок, если для некоторой перестановки $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ символов $1, \ldots, n$

$$p_{ij} = \begin{cases} 1, & \text{если } j = \alpha_i; \\ 0 & \text{иначе.} \end{cases}$$

Умножением матрицы перестановок P на вектор $x = (x_1, x_2, \ldots, x_n)^T$ получим вектор $\tilde{x} = (x_{\alpha_1}, x_{\alpha_2}, \ldots, x_{\alpha_n})^T$. При левом умножении $n \times m$ матрицы A на матрицу P аналогичным образом переставятся строки A. Если же $m \times n$ матрицу B умножить на P справа, то столбцы полученной матрицы будут переупорядочением столбцов B. Однако это переупорядочение описывается не исходной перестановкой $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, а обратной ей $\{\beta_1, \beta_2, \ldots, \beta_n\}$. Если желаемый новый порядок столбцов все же $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, то матрицу B нужно умножить справа на матрицу $Q = P^T$. Эта матрица Q также является матрицей перестановок, но соответствует обратной перестановке

$\{\beta_1, \beta_2, \ldots, \beta_n\}$. Пример:

$$P = \begin{bmatrix} 0 & & & & 1 \\ 0 & & & 1 & \\ 0 & & \iddots & & \\ 0 & 1 & & & \\ 1 & & & & 0 \end{bmatrix}, \qquad Px = \begin{bmatrix} x_n \\ x_{n-1} \\ \cdots \\ x_2 \\ x_1 \end{bmatrix}.$$

Наиболее часто используются матрицы, описывающие транспозицию двух элементов вектора или двух строк (столбцов) матричного сомножителя. Если номера переставляемых строк суть i и j, то соответствующую матрицу перестановок будем обозначать через P_{ij}. От единичной матрицы она будет отличаться лишь в четырех элементах: $p_{ii} = p_{jj} = 0, \quad p_{ij} = p_{ji} = 1$.

Матрица перестановок является простейшим примером ортогональной матрицы.

8.3.2. Матричное представление схемы Гаусса

Матрица A системы линейных уравнений посредством метода Гаусса фактически раскладывается в произведение $A = LU$ двух треугольных матриц — левой нижней L и правой верхней U — в сочетании с той или иной схемой выбора главного элемента, описываемой массивом перестановок P. Треугольное разложение можно сформировать, не зная правую часть системы. После этого решение линейной системы $Ax = b$ сводится к последовательному решению более простых линейных систем:

- решается система $Pz = b$, что равносильно переупорядочению элементов вектора b;
- решается $Ly = z$ (применяется прямая подстановка);
- вектор x определяется из треугольной системы $Ux = y$ (обратная подстановка).

Математически имеем

$$x = U^{-1}y = U^{-1}L^{-1}z = U^{-1}L^{-1}P^{-1}b = (PLU)^{-1}b = A^{-1}b.$$

8.3.3. Разложение и метод Холецкого

Для симметричной положительно определенной матрицы A можно добиться того, чтобы сомножители треугольного разложения были

взаимно транспонированными матрицами.

Матрица A представляется в виде

$$A = S^* D S,$$

где S — правая треугольная матрица, S^* — сопряженная с ней, D — диагональная матрица с элементами главной диагонали, равными $+1$ или -1. Тогда при $i \le j$

$$a_{ij} = \bar{s}_{1i} s_{1i} d_{i1} + \ldots + \bar{s}_{ij} s_{ij} d_{ii}.$$

Аналогичные уравнения при $i > j$ опущены, так как из соображений симметрии эквивалентны вышеприведенным. Отсюда получаем формулы для компонент нового представления:

$$\begin{aligned} d_{ii} &= \operatorname{sign}\Big(a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 d_{kk}\Big), \\ s_{ii} &= \Big|a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 d_{kk}\Big|^{1/2}, \\ s_{ij} &= \frac{a_{ij} - \sum\limits_{k=1}^{i-1} \bar{s}_{ki} s_{kj} d_{kk}}{\bar{s}_{ii} d_{ii}}, \quad i < j. \end{aligned}$$

Теперь решение исходной системы сводится к последовательному решению двух систем с треугольными матрицами. Заметим, что в случае $A > 0$ все $d_{ii} = 1$ и $A = S^* S$. Объем требуемой памяти и трудоемкость разложения в сравнении со схемой Гаусса уменьшаются приблизительно вдвое.

8.4. Тактика решения линейных систем

В общем случае оптимальный выбор способа решения системы линейных уравнений зависит от такого количества обстоятельств и столь сложным образом, что ряд авторов пишет о *тактике* решения линейных систем [68, с. 50]. И. В. Бахвалов [10, с. 322] считает, что если порядок системы n небольшой, то проще всего обратиться к стандартным программам метода отражений (число арифметических действий $N \approx 2n^3/3$)

или метода вращений (число арифметических действий $N \approx 4n^3/3$, но меньше накопление вычислительной погрешности).

Если применение этих методов невозможно или нецелесообразно, следует проанализировать возможность применения простейших по своей структуре итерационных методов: простой и зейделевой итераций, сверхрелаксации.

Многоообразие свойств матриц, а также возникающих задач и их взаимосвязей определили целесообразность декомпозиции задач линейной алгебры на элементарные. Стандартные пакеты линейной алгебры включают в себя ядро, состоящее из базисных подпрограмм нижнего уровня — в частности, выполняющих LU-разложение. Для решения системы уравнений с плотной матрицей целесообразно применять метод, основанный на непосредственном разложении матрицы A в произведение нижней треугольной матрицы A и верхней треугольной матрицы U:

$$Ax = LUx = b.$$

Это разложение, если оно существует, единственно с точностью до выбора диагональных элементов упомянутых матриц: можно положить либо все $l_{ii} = 0$, либо все $u_{ii} = 0$. Рассмотрим более подробно уже упоминавшийся второй вариант, соответствующий схеме единственного деления метода Гаусса. Формулы для вычисления матричных сомножителей получим из равенства

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & \dots & u_{1n} \\ 0 & 1 & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}.$$

Элементы матриц L и U на r-м шаге определяются в следующем порядке: вначале элементы r-го столбца матрицы L

$$l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik} u_{kr}, \quad i = \overline{r, n},$$

а затем r-й строки матрицы U

$$u_{ri} = \left(a_{ri} - \sum_{k=1}^{r-1} l_{rk} u_{ki} \right) \Big/ l_{rr}, \quad i = \overline{r+1, n}.$$

Возвращаясь к записи исходной системы уравнений и полагая $Ux = y$, получаем уравнение $Ly = b$ с нижней треугольной матрицей, решение

которого находим по формулам

$$y_i = \Big(b_i - \sum_{k=1}^{l-1} l_{ik} y_k\Big) \Big/ l_{ii}, \quad i = \overline{1,n}.$$

Неизвестное x находим из системы $Ux = y$ с верхней треугольной матрицей, вычисляя

$$x_i = y_i - \sum_{k=i+1}^{n} u_{ik} x_k, \quad i = n, n-1, \ldots, 1.$$

Трудоемкость этого метода асимптотически составляет $2n^3/3$ операций умножения. В [68] описан дополняющий этот метод выбор главного элемента по столбцу.

Для систем линейных алгебраических уравнений с симметричными положительно определенными матрицами LU-разложение сводится к произведению

$$A = LL^T,$$

где

$$L = \begin{bmatrix} l_{11} & 0 & \ldots & 0 \\ l_{21} & l_{22} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ l_{n1} & l_{n2} & \ldots & l_{nn} \end{bmatrix}.$$

Реализуемый с учетом симметрии матрицы, он оказывается самым экономным из прямых методов как по числу арифметических операций, так и по загрузке памяти и легко модифицируется применительно к ленточным и разреженным матрицам. Формулы для вычисления элементов матрицы L имеют вид

$$\begin{aligned} l_{11} &= \sqrt{a_{11}}, \quad l_{i1} = a_{i1}/l_{11}, \quad i = \overline{2,n}, \\ l_{ii} &= \Big(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\Big)^{1/2}, \quad l_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}\Big) \Big/ l_{jj}, \\ i &= \overline{2,n}, \quad j = \overline{1,i-1}. \end{aligned}$$

Модификация вышеприведенных формул для решения вспомогательных треугольных систем применительно к этому случаю очевидна.

Для положительно определенной матрицы данный процесс всегда выполним, и никаких перестановок строк не требуется.

8.5. Билинейная и квадратичная формы матриц

Пусть A — действительная квадратная матрица и x, y — произвольные векторы n-мерного действительного пространства. Составим скалярное произведение

$$(Ax, y) = \sum_{j=1}^{n} (Ax)_j y_j = \sum_{j=1}^{n} \sum_{k=1}^{n} a_{jk} x_k y_j = (A^T y, x) = (x, A^T y). \quad (8.5.1)$$

Его называют *билинейной формой* матрицы A. Заметим, что матрицу можно переставлять с первого места на второе, соответственно транспонируя ее. В случае симметричной матрицы последняя оговорка становится излишней.

Запишем векторно-матричное произведение

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = ax_1^2 + 2bx_1x_2 + cx_2^2.$$

Обратите внимание на то, что средний сомножитель — симметричная матрица. Полученную *квадратичную форму* можно записать в матричном виде:

$$f = x^T A x.$$

Квадратичная форма называется *положительно определенной*, если она принимает только положительные значения, обращаясь в нуль лишь при нулевом векторе x. В этом случае уравнение $f(x_1, x_2, \ldots, x_n) = c$ при $c > 0$ представляет собой уравнение эллипсоида. О положительной (отрицательной) определенности говорят и применительно к матрице, порождающей квадратичную форму.

Необходимыми и достаточными условиями положительной определенности матрицы являются (каждое в отдельности):

- положительность всех главных миноров матрицы;
- положительность всех ведущих элементов схемы Гаусса (без перестановки строк);
- существование такой невырожденной матрицы W, что $A = W^T W$;
- положительность всех собственных значений (см. ниже).

Аналогично формулируются критерии отрицательной определенности и положительной (отрицательной) *полуопределенности*. Матрица B отрицательно определена тогда и только тогда, когда положительно определена матрица $A = -B$.

Для произвольной вещественной (m, n) матрицы A квадратная матрица $B = A^T A$ ($B = AA^T$) симметрична и положительно полуопределена. Из этих равенств следует

$$\mathrm{cond}_2 B = (\mathrm{cond}_2 A)^2.$$

Таким образом, матрица B обусловлена (по второй норме) значительно хуже, чем A.

Назовем линейную систему

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \qquad i = \overline{1, n}$$

нормальной, если:

- матрица коэффициентов $\{a_{ij}\}$ — симметричная;
- соответствующая квадратичная форма положительно определена.

Нормальные системы встречаются в способе наименьших квадратов, при нахождении направлений главных осей эллипсоида и т. д.

8.6. Собственные векторы и собственные значения

Ряд инженерных задач сводится к рассмотрению систем уравнений, которые имеют единственное решение только при определенном значении параметра, именуемом *собственным значением* системы. Так, при динамическом анализе механических систем собственные значения соответствуют собственным частотам колебаний, а собственные векторы характеризуют моды этих колебаний. При расчете конструкций собственные значения позволяют определять критические нагрузки, превышение которых приводит к потере устойчивости. Задачи на собственные значения возникают при исследовании колебаний и устойчивости иной физической и химической природы, при факторном анализе, а также как самостоятельные математические проблемы.

Пусть дано дифференциальное уравнение

$$\frac{du}{dt} = Au.$$

Полагая $u = e^{\lambda t}x$ и выполнив подстановку, имеем

$$\lambda e^{\lambda t}x = Ae^{\lambda t}x.$$

Отсюда получаем

$$Ax = \lambda x,$$
$$(A - \lambda I)x = 0 \qquad (8.6.1)$$

— основные уравнения относительно собственных значений $\{\lambda_i\}$ и собственных векторов $\{x_i\}$. Собственные векторы являются гармониками системы и при отсутствии среди $\{\lambda_i\}$ кратных независимы друг от друга. Уравнение *устойчиво*, если вещественные части всех $\{\lambda_i\}$ отрицательны. Исследование устойчивости процессов является важнейшей составной частью теории автоматического управления, где оно проверяется, например, известным критерием Рауса—Гурвица.

Пусть матрица A имеет n линейно независимых собственных векторов. Если взять эти векторы в качестве столбцов матрицы S, то $S^{-1}AS = \Lambda$ — диагональная матрица, на диагонали которой стоят собственные значения матрицы A.

Собственные значения матрицы A задаются уравнением

$$\det(A - \lambda I) = 0. \qquad (8.6.2)$$

Этот определитель называется *характеристическим* (вековым) определителем матрицы A и в развернутом виде имеет вид

$$\begin{vmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{vmatrix} = 0, \qquad (8.6.3)$$

или

$$(-\lambda)^n + \sigma_1(-\lambda)^{n-1} + \sigma_2(-\lambda)^{n-2} + \dots + \sigma_{n-1}(-\lambda) + \sigma_n = 0. \qquad (8.6.4)$$

Коэффициент σ_k характеристического полинома есть сумма всех главных миноров k-го порядка матрицы A. В частности, σ_1 равен сумме диагональных элементов матрица A, а свободный член σ_n — определителю матрицы. Корни этого уравнения образуют *спектр* собственных значений.

Квадратная матрица называется *устойчивой*, если ее спектр расположен в полуплоскости $\Re(\lambda) < 0$.

Квадратная матрица называется *сходящейся*, если ее спектр принадлежит внутренности единичного круга $|\lambda| < 1$. В последнем случае итерационный процесс решения системы линейных уравнений

$$x_{k+1} = Ax_k + f \tag{8.6.5}$$

сходится при любом начальном приближении.

Наибольший из модулей собственных значений матрицы A называется *спектральным радиусом* этой матрицы и обозначается $\rho(A)$. Для всякой квадратной матрицы A выполняется соотношение

$$\rho(A) \leq ||A||.$$

Поэтому неравенство $||A|| < 1$ является достаточным условием сходимости упомянутого процесса (8.6.5).

Собственные значения положительно определенной (полуопределенной) симметричной матрицы положительны (неотрицательны).

Пример 8.1. Найдем собственные значения матрицы

$$A = \begin{bmatrix} 2 & 5 \\ 4 & 3 \end{bmatrix}.$$

Согласно (8.6.2) имеем уравнение

$$\begin{vmatrix} 2-\lambda & 5 \\ 4 & 3-\lambda \end{vmatrix} = (2-\lambda)(3-\lambda) - 20 = \lambda^2 - 5\lambda - 14 = 0.$$

Значит, $\lambda_1 = 7$ и $\lambda_2 = -2$.

8.6.1. Основные свойства собственных значений

1. Среди собственных значений могут встречаться кратные. Эти случаи существенно меняют и усложняют картину изучаемых явлений и в данной книге не обсуждаются.

2. Если собственные значения матрицы различны, то ее собственные векторы ортогональны. Совокупность n линейно независимых векторов $\{X_i\}$ образует *базис* рассматриваемого пространства, так что любой

вектор Y этого пространства можно представить линейной комбинацией векторов базиса:

$$Y = \sum_{i=1}^{n} a_i X_i.$$

3. Если две матрицы подобны, то их собственные значения совпадают. В самом деле, так как $AX = \lambda X$, то $P^{-1}AX = \lambda P^{-1}X$. Полагая $X = PY$, имеем $P^{-1}APY = \lambda Y$, или $BY = \lambda Y$.

4. Умножая собственный вектор на скаляр, вновь получаем собственный вектор той же матрицы. Поэтому обычно все собственные векторы нормируют, разделив каждый элемент либо на наибольший модуль элемента вектора, либо на его эвклидову норму.

5. Собственные значения прямой и обратной матриц взаимообратны.

6. Сумма n собственных значений матрицы равна сумме ее диагональных элементов (следу).

7. Модуль любого собственного значения матрицы не больше любой ее нормы.

8. Собственные значения квадрата матрицы равны квадратам собственных значений исходной.

Матрицы высокой размерности кроме собственных имеют «почти собственные» значения, существенно влияющие на сходимость итераций [10, с. 270].

Зафиксируем свойства *собственных пар* {число, вектор}:

- Если для матрицы A получена собственная пара $\{\lambda, x\}$ и α — ненулевое число, то $\{\lambda, \alpha x\}$ — тоже собственная пара для A.
- Если для матрицы $A - pI$ получена собственная пара $\{\mu, x\}$, то $\{\lambda = \mu + p, x\}$ — собственная пара для A.
- Если для матрицы A получена собственная пара $\{\lambda, x\}$, то $\{1/\lambda, x\}$ есть собственная пара для A^{-1}.
- Собственными числами диагональных и треугольных матриц являются их диагональные элементы.
- Отношение Релея
$$R(x) = (Ax, x)/(x, x)$$
в случае, когда x — собственный вектор матрицы A, является ее собственным числом.

8.6.2. Собственные значения матриц специального вида

Симметричные матрицы имеют ряд свойств собственных значений, весьма полезных для вычислений:

- Все собственные значения симметричной матрицы действительны.
- Если собственные значения матрицы различны, то соответствующие им собственные векторы ортогональны и образуют базис рассматриваемого пространства. Следовательно, любой вектор в данном пространстве можно выразить через собственные векторы нашей матрицы.
- Всякую симметричную матрицу с помощью преобразования подобия можно привести к диагональному виду.
- Каждому собственному значению соответствует столько линейно независимых собственных векторов, какова кратность собственного значения.
- Матрица положительно определена тогда и только тогда, когда все собственные значения ее положительны.
- Спектральная норма матрицы равна максимальному из модулей собственных значений.
- Если матрицы A, B перестановочны, т. е. $AB = BA$, то они имеют общую систему собственных векторов.
- При том же условии собственные значения произведения равны произведению соответствующих одному и тому же собственному вектору собственных значений сомножителей.

Для эрмитовой матрицы все собственные значения вещественны, для косоэрмитовой — чисто мнимы, для унитарной находятся на единичной окружности в комплексной плоскости. Для всех этих матриц их собственные векторы ортогональны и могут быть выбраны ортонормированными.

8.6.3. О вычислении собственных значений

Непосредственное развертывание векового определителя чрезвычайно трудоемко, и для его выполнения используются специальные методы (А. Н. Крылова, А. М. Данилевского, Леверрье и др.). В [23, с. 421]

проводится сравнение трудоемкости этих методов. Второй частью задачи является вычисление корней получаемого при этом многочлена n-й степени. В связи с этим прямой подход к решению алгебраической проблемы собственных значений обычно применяют при $n = 2, 3$; уже для $n \geq 4$ на первый план выходят специальные численные методы.

Выбор наиболее эффективного метода определения собственных значений или собственных векторов зависит от типа уравнений, числа искомых собственных значений и их характера. Если матрица треугольная, то ее собственные значения совпадают с диагональными элементами. Путь к нахождению собственных значений — преобразование матрицы к треугольному или диагональному виду.

Алгоритмы задач на собственные значения делятся на две группы. Итерационные методы хорошо приспособлены для нахождения наибольшего и наименьшего собственных значений. Методы преобразований подобия сложнее, но зато позволяют определить все собственные значения и собственные векторы.

8.7. Частичная проблема собственных значений

8.7.1. Постановка задачи

В ряде случаев требуется найти лишь максимальное или минимальное по модулю собственное значение матрицы. В методе верхней сверхрелаксации (см. разд. 7.9.4) для вычисления итерационного параметра необходимо знать максимальное собственное значение итерируемой матрицы. В «колебательных» задачах практический интерес представляют несколько первых (в порядке убывания модулей) собственных значений, причем меньшие из них достаточно определить с меньшей точностью.

Еще одна задача — отыскание собственного значения матрицы, ближайшего к заданному числу λ^0, или расстояния от λ^0 до спектра матрицы (когда рассматривается явление резонанса).

В таких случаях предпочтительны менее трудоемкие частные методы. Решение всех частичных вариантов проблемы собственных значений обычно сводят к отысканию максимального по модулю собственного значения матрицы $B = g(A)$, такой, что это собственное значение соответствует разыскиваемому собственному значению матрицы A.

8.7.2. Степенной метод

Пусть собственные значения матрицы с учетом их кратности расположены в порядке

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_{n-1}| \geq |\lambda_n|.$$

Зададим произвольное начальное приближение y_0 к собственному вектору v_n и значение $\varepsilon > 0$, управляющее окончанием процесса. Пусть за базис n-мерного пространства принята система собственных векторов $\{v_j\}$, $j = \overline{1,n}$. Тогда произвольный вектор y может быть разложен по собственным векторам:

$$y^{(0)} = c_1 v_1^{(0)} + c_2 v_2^{(0)} + \ldots + c_n v_n^{(0)}.$$

Поскольку собственный вектор определяется с точностью до постоянного сомножителя, условие нормировки пока принимать во внимание не будем.

Из последовательных итераций получаем векторы

$$\begin{aligned}
y^{(1)} &= Ay^{(0)} = c_1\lambda_1 v_1^{(0)} + c_2\lambda_2 v_2^{(0)} + \ldots + c_n\lambda_n v_n^{(0)},\\
y^{(2)} &= Ay^{(1)} = c_1\lambda_1^2 v_1^{(1)} + c_2\lambda_2^2 v_2^{(1)} + \ldots + c_n\lambda_n^2 v_n^{(1)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
y^{(k)} &= Ay^{(k-1)} = c_1\lambda_1^k v_1^{(k-1)} + c_2\lambda_2^k v_2^{(k-1)} + \ldots + c_n\lambda_n^k v_n^{(k-1)}\\
&= \lambda_1^k \left[c_1 v_1^{(k-1)} + c_2 \left(\tfrac{\lambda_2}{\lambda_1}\right)^k v_2^{(k-1)} + \ldots + c_n \left(\tfrac{\lambda_n}{\lambda_1}\right)^k v_n^{(k-1)} \right].
\end{aligned}$$

Из последней строки следует, что

$$y^{(k)} = c_1\lambda_1^k \left[v_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \ldots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k v_n \right].$$

При сделанных допущениях и увеличении k в квадратных скобках будет доминировать первое слагаемое, т. е. вектор $y^{(k)}$ будет *по направлению* стремиться к собственному вектору x_1.

Построенный на этих выкладках *степенной метод* состоит в следующем:

1. Берется произвольный начальный вектор.
2. Простыми итерациями по формуле $y^{(k)} = Ay^{(k-1)}$ строится последовательность векторов $y^{(k)}$. Одновременно вычисляются отношения одноименных компонент результатов смежных итераций (отношения с очень малыми делителями игнорируются).

3. Стабилизация по номеру итерации упомянутых отношений с приемлемой точностью указывает, что найденное отношение есть наибольшее по модулю собственное значение матрицы, а последняя итерация пробного вектора есть соответствующий ему собственный вектор.

В процессе итераций получаемый вектор нормируется своей максимальной компонентой.

Скорость сходимости метода линейна (как у геометрической прогрессии) и определяется отношением $q = |\lambda_1/\lambda_2|$. Если сходимость слишком медленна, это свидетельствует о близких по модулю корнях, альтернирующий знак — о паре комплексных значений.

Убыстрение достигается ускоренным возведением матрицы в степень (уже обсуждавшимся повторным квадрированием) с нормированием пробного вектора после каждого скачка. Более популярный способ — Δ^2-процесс Эйткена. Считается, что если $\lambda^{(k-1)}, \lambda^{(k)}, \lambda^{(k+1)}$ являются последовательными приближениями к собственному числу, полученному степенным методом, то число

$$\tilde{\lambda} = \lambda^{(k-1)} - \frac{(\lambda^{(k)} - \lambda^{(k-1)})^2}{\lambda^{(k+1)} - 2\lambda^{(k)} + \lambda^{(k-1)}}$$

ближе к пределу этой последовательности, чем каждое из них. Этот факт может быть использован в реальных алгоритмах либо через несколько шагов (через два на третий), либо на завершающем этапе вычислений. Для искомого *собственного вектора* такое ускорение может производиться покоординатно.

Для отыскания *наименьшего* собственного значения применяется *обратный* степенной метод. В нем исходную систему умножают на A^{-1}:

$$A^{-1}AX = \lambda A^{-1}X.$$

Разделив обе части его на λ, получаем

$$\lambda^{-1}X = A^{-1}X.$$

Здесь собственное значение $1/\lambda$ связывается с матрицей A^{-1}, причем наименьшему λ соответствует наибольшее $1/\lambda$.

В обратном степенном методе *со сдвигом* заменяют A на $A - \alpha I$. Тогда все $\{\lambda_i\}$ сдвигаются на ту же величину α, и скорость сходимости для обратного степенного метода определяется отношением

$$r = |(\lambda_n - \alpha)/(\lambda_{n-1} - \alpha)|.$$

Если α — хорошая аппроксимация λ_n, то сходимость станет очень быстрой.

Иногда в качестве такой матрицы $g(A)$ может использоваться матрица $(A-\lambda^0 I)^{-1}$. При этом $(A-\lambda^0 I)^{-1}$ не выписывается явном виде, а необходимые по ходу вычислений векторы $(A-\lambda^0 I)^{-1}y$ находят, решая систему уравнений $(A-\lambda^0 I)x = y$.

В симметричном случае при отсутствии λ_1 пользуются отношением Релея

$$\alpha_k = (u_k^T A u_k)/(u_k^T u_k).$$

8.7.3. Улучшение сходимости простых итераций

Знание наибольшего собственного значения матрицы позволяет ускорить сходимость простых итераций. Л. А. Люстерник предложил считать

$$x \approx x^{(m)} + \frac{\lambda_1}{1-\lambda_1}(x^{(m)} - x^{(m-1)}).$$

Здесь λ_1 — наибольшее по модулю собственное значение матрицы A (при условии, что оно единственное). Приближенной оценкой последнего служит

$$\lambda_1 \approx \frac{(x^{(m)} - x^{(m-1)})^2}{(x^{(m-1)} - x^{(m-2)})^2}$$

(в числителе и знаменателе — квадраты длин последовательных векторов-поправок).

8.8. Полная проблема собственных значений

Методы решения полной проблемы собственных значений существенно используют понятие *подобия матриц*. Две матрицы A и B, описывающие в различных базисах одно и то же линейное преобразование, называются подобными: $A \propto B$. Подобие всегда взаимно.

Пусть одно и то же линейное преобразование записано в двух различных базисах:

$$y = Ax,$$

$$\eta = B\xi.$$

Обозначим через S невырожденную матрицу перехода от второй системы к первой, т. е.

$$x = S\xi, \quad y = S\eta.$$

Тогда первое преобразование можно записать как

$$S\eta = AS\xi.$$

Умножив эту запись слева на S^{-1}, получаем

$$\eta = S^{-1}AS\xi.$$

Сравнивая результат со вторым вариантом преобразования, убеждаемся, что

$$B = S^{-1}AS.$$

Соответственно,

$$A = SBS^{-1}. \tag{8.8.1}$$

Можно доказать, что подобные матрицы имеют одинаковые характеристические полиномы и, следовательно, одинаковые наборы собственных значений. Это позволяет находить собственные числа матрицы, заменив последнюю на подобную ей, но такого вида, при котором эта задача решается легко. На этой идее основан уже упоминавшийся метод Данилевского. В частности, всякую действительную симметричную матрицу при помощи преобразования подобия можно привести к диагональному виду. Напомним, что собственными значениями диагональной матрицы являются элементы диагонали.

Теорема 8.1. Собственные векторы действительной симметричной матрицы, соответствующие различным собственным значениям, ортогональны между собой.

Доказательство. Пусть дана действительная симметричная матрица A и два собственных вектора ее $x^{(1)}$ и $x^{(2)}$, соответствующих собственным числам λ_1 и λ_2 ($\lambda_1 \neq \lambda_2$). По определению собственных векторов

$$\begin{aligned} Ax^{(1)} &= \lambda_1 x^{(1)}, \\ Ax^{(2)} &= \lambda_2 x^{(2)}. \end{aligned} \tag{8.8.2}$$

Запишем скалярное произведение

$$(Ax^{(1)}, x^{(2)}) = (x^{(1)}, Ax^{(2)})$$

(напомним, что матричный множитель в скалярном произведении можно переставлять, заменяя его транспонированным). В силу равенства (8.8.2) получаем

$$(\lambda_1 x^{(1)}, x^{(2)}) = (x^{(1)}, \lambda_2 x^{(2)})$$

или

$$\lambda_1 (x^{(1)}, x^{(2)}) = \lambda_2 (x^{(1)}, x^{(2)}).$$

Таким образом, $(\lambda_1 - \lambda_2)(x^{(1)}, x^{(2)}) = 0$. С учетом условия теоремы это равенство возможно лишь при ортогональности $x^{(1)}$ и $x^{(2)}$. ▲

Существует ряд отработанных алгоритмов решения полной проблемы собственных значений. В [110, с. 68–69] приведена сводная таблица с рекомендациями по выбору способов расчета собственных значений. Наиболее совершенные из них основаны на различных модификациях QR-алгоритма [10]. Здесь, начиная с $A_0 = A$, строят для $k = 1, 2, \ldots$ последовательность матриц $\{A_k\}$ по формулам

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k.$$

Первая из них означает разложение матрицы A_k в произведение ортогональной Q_k и правой треугольной R_k (такое разложение существует для любой квадратной матрицы), а вторая — перемножение полученных в результате факторизации A_k матриц Q_k и R_k в обратном порядке.

При определенных ограничениях, одно из которых — отсутствие у матрицы A равных по модулю собственных значений, генерируемая последовательность матриц $\{A_k\}$ сходится к матрице правой треугольной формы с диагональю из собственных чисел.

Для вычисления всех собственных значений и принадлежащих им собственных векторов симметричной матрицы целесообразно использовать метод Якоби. Напомним, что всякая симметричная матрица A n-го порядка может быть представлена в виде

$$A = Q^T \Lambda Q, \tag{8.8.3}$$

где Q — ортогональная матрица, Λ — диагональная с элементами $\lambda_1, \lambda_2, \ldots, \lambda_n$ (собственными значениями матрицы A). Из вышеизложенного следует, что

$$AQ^T = Q^T \Lambda, \quad QAQ^T = \Lambda.$$

Таким образом, i-я строка матрицы Q является собственным вектором, соответствующим собственному числу λ_i, и преобразование подобия переводит матрицу A в диагональную Λ.

В методе Якоби используется цепочка преобразований вращения.

8.9. Сингулярные числа и сингулярное разложение

Арифметические квадратные корни из положительных собственных значений матрицы AA^T (A^TA) называют *сингулярными числами* матрицы A. Спектральная норма $||A||_2$ равна s_1 — наибольшему из ее сингулярных чисел. Число обусловленности по второй норме

$$\mathrm{cond}_2 A = s_1(A)/s_n(A).$$

Наименьшее сингулярное число s_n равно расстоянию в той же норме от матрицы A до ближайшей вырожденной матрицы [37, с. 261]. Его оценка считается наиболее надежным (но весьма трудоемким) методом проверки невырожденности матрицы. Сингулярное разложение матриц является весьма эффективным методом сжатия изображений [37, с. 265–267].

8.10. Матричные ряды

Пользуясь понятием предела матрицы, можно ввести в рассмотрение матричные степенные ряды

$$\sum_{k=1}^{\infty} A_k X^k = \lim_{N\to\infty} \sum_{k=1}^{N} A_k X^k, \qquad (8.10.1)$$

где X — квадратная матрица порядка (n, n) и $\{A_k\}$ — матрицы одного и того же типа с n столбцами. Мы рассматриваем правый степенной ряд, но аналогично можно определить и левый.

Если предел (8.10.1) существует, то матричный ряд называется сходящимся, а полученная в пределе матрица считается суммой этого ряда. Для сходимости достаточно потребовать, чтобы для упомянутых матриц какая-либо из канонических норм была меньше единицы.

Наибольший интерес представляет матрично-геометрическая прогрессия, сумма которой при $||X|| < 1$ равна

$$\sum_{k=0}^{\infty} AX^k = A(I - X)^{-1}. \qquad (8.10.2)$$

(см. [23, с. 250]). Она легко выводится по аналогии со скалярным случаем. Эта формула широко используется в теории дискретных цепей Маркова и теории очередей.

8.11. Матричные уравнения специального вида

8.11.1. Типы уравнений

В приложениях (теории автоматического управления) часто встречаются уравнения Сильвестра: непрерывное $AX+XB=C$ и дискретное $AXG-X=C$. Важным частным случаем является непрерывное уравнение Ляпунова: $AX+XA^T=C$. Теорема 8.2 (Ляпунова) утверждает: если A — вещественная квадратная матрица, C — также вещественная, симметричная и положительно определенная, и уравнение

$$A^TX+XA=-C \tag{8.11.1}$$

имеет положительно определенное решение X, то матрица A *устойчива*, т. е. все ее собственные значения $\{\lambda_i\}$ удовлетворяют условию $\Re(\lambda)<0$.

Уравнения указанных типов не приводятся к стандартным линейным системам вида (7.7.1). Тем более это относится к квадратичным уравнениям

$$AX+XB+XFX+C=0, \tag{8.11.2}$$

включающим в себя непрерывное уравнение Риккати, и полиномиальным уравнениям

$$A_0X^n+A_1X^{n-1}+\ldots+A_{n-1}X+A_n=0. \tag{8.11.3}$$

Возможные подходы к их решению обсуждаются в [31]. Общей чертой решения матричных уравнений перечисленных видов и родственных им является составление и частичное решение вспомогательной проблемы собственных значений — обычной или обобщенной. Из полученной спектральной информации конструируется решение исходного уравнения.

Методы решения матричного уравнения Ляпунова основаны на приведении матрицы A к так называемой *форме Шура* — блочно-треугольной с диагональными блоками первого и второго порядка, из которой определяются собственные значения A.

Дискретное уравнение Ляпунова $AXA^T - X = C$ сводится к системам из mn линейных уравнений, которые в матричной форме записываются через прямые (кронекеровы) произведения матриц. Методы состоят из четырех этапов: приведение матриц к специальным формам (ортогонально-треугольным); преобразование правой части; решение преобразованного уравнения; обратная замена переменных.

8.11.2. Пример на квадратное уравнение

Практически единственным эффективным методом расчета многоканальных систем обслуживания с отличными от экспоненциального временны́ми распределениями является метод фиктивных фаз. На диаграмме переходов для $M/E_3/2$ (рис. 8.1) состояние системы идентифицируется полным количеством заявок в ней (номер яруса) и распределением проходящих обслуживание заявок по его фазам («ключ» соответствующего микросостояния). Здесь сумма значений позиций ключа равна числу задействованных каналов обслуживания.

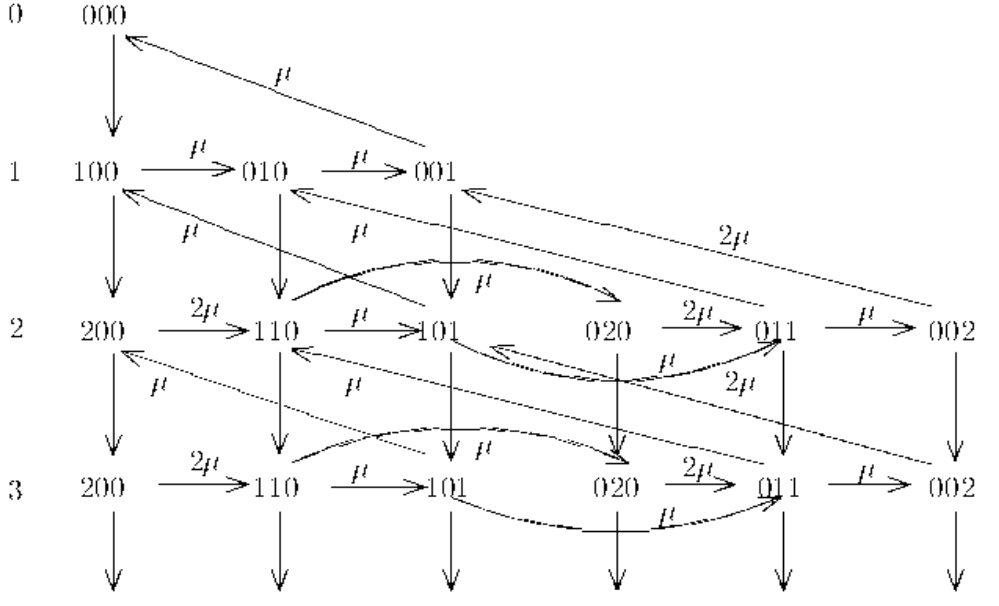

Рис. 8.1. Переходы в системе $M/E_3/2$

Для расчета подобных фиктивных фаз можно использовать метод матрично-геометрической прогрессии. Здесь векторы вероятностей микросостояний полностью занятой системы представляются соотношением типа

$$\gamma_j = \gamma_n R^{j-n}, \qquad j = n, n+1, \ldots, \tag{8.11.4}$$

где R — матричный знаменатель прогрессии. Можно показать [84], что искомый знаменатель прогрессии должен удовлетворять матричному квадратному уравнению

$$R^2 B - R(D - C) + A = 0, \tag{8.11.5}$$

где A, B, D, C — известные матрицы интенсивностей переходов между микросостояниями смежных ярусов.

8.11.3. Простое решение

Ниже описан поиск более простого пути к решению уравнения вида (8.11.5). Прежде всего рассматривались две итерационных формы его

$$R = A(D - C)^{-1} + R^2 B(D - C)^{-1} \tag{8.11.6}$$

и

$$R = A(D - C - RB)^{-1}. \tag{8.11.7}$$

В качестве начального приближения использовалось

$$R_0 = xI, \tag{8.11.8}$$

где x — определяемое по известной формуле в зависимости от исходных данных предельное при $j \to \infty$ отношение суммарных вероятностей смежных ярусов. Число итераций для достижения нормы невязки 10^{-6} доходило до 343, причем скорость сходимости по мере приближения к решению замедлялась.

Наилучшим решением вопроса оказалось вычисление поправок к знаменателю прогрессии в линейном приближении. Для упрощения обозначений без потери общности перепишем (8.11.5), полагая $C = 0$, и определим матричную поправку Δ из условия

$$(R + \Delta)(R + \Delta)B - (R + \Delta)D + A = 0.$$

Пренебрегая членом, содержащим квадрат поправки, имеем

$$R^2 B + \Delta RB + R\Delta B - RD - \Delta D + A \approx 0,$$

или

$$\Delta(RB-D)+R\Delta B\approx RD-R^2B-A.$$

Последнее уравнение приводится к виду

$$\Delta F+R\Delta B=G, \tag{8.11.9}$$

является «ближайшим родственником» сильвестровых и *линейно* относительно поправочной матрицы. Оно легко решается после покомпонентного расписывания системы из n^2 уравнений относительно элементов искомой матрицы. Каждое такое уравнение имеет вид

$$\sum_{m=1}^{n} d_{im}f_{mj}-\sum_{m=1}^{n} r_{im}\sum_{l=1}^{n} d_{lm}a_{mj}=g_{ij}$$

и может быть переписано в виде

$$\sum_{m=1}^{n} d_{im}f_{mj}-\sum_{l=1}^{n} d_{lm}\sum_{m=1}^{n} r_{im}a_{mj}=g_{ij}.$$

Ниже приводятся процедура решения обсуждаемого матричного уравнения и результаты ее применения. Работа процедуры начинается с обнуления расширенной матрицы коэффициентов, расположенной по строкам. Затем в нее по строкам заносятся F-коэффициенты, образуемые элементами матрицы F, причем запись каждый раз переадресуется на N элементов. Та же идея используется при учете коэффициентов, образуемых произведением RA. Здесь сумма не копится, а ее слагаемые последовательно вычитаются из ранее сформированного значения. Формирование столбца правых частей пояснений не требует. Полученная система уравнений решается процедурой GAUSSC, рассчитанной на работу с комплексными коэффициентами. Наконец, результат засылается в ответную матрицу *по столбцам* (это особенность представления матриц в Фортране).

```
subroutine msqreq(n,a,f,g,r,d)
!****************************************
!*       Решение матричного уравнения      *
!*                d*f-r*d*a=g              *
!*          относительно матрицы d         *
!*                для метода               *
!*  матрично-геометрической прогрессии     *
!*         Разработчик Ю. И. Рыжиков       *
```

```
!*                    (c)                    *
!*             Версия 20.12.1998             *
!*****************************************
   integer           n
   double complex    a,f,g,r,d
   dimension         a(n,n),f(n,n),r(n,n),g(n,n),d(n,n)
$include: 'gaussc.ins'

   integer           i,j,k,l,l1,m,nn,nn1
   double complex, allocatable :: u(:,:), dd(:)

   nn=n*n; nn1=nn+1
   allocate (u(nn,nn1),dd(nn))
   ! предварительное обнуление
   do i=1,nn
      do j=1,nn1; u(i,j)=0; end do
   end do

   ! f-коэффициенты
   k=1
   do i=1,n
      do j=1,n
         l=i
         do m=1,n; u(k,l)=f(m,j); l=l+n; end do
         k=k+1
      end do
   end do

   ! ra-коэффициенты
   k=1
   do i=1,n
      do j=1,n
         l1=0
         do l=1,n
            do m=1,n
               u(k,l1+m)=u(k,l1+m)-a(l,j)*r(i,m)
            end do
         l1=l1+n
      end do
```

```
      k=k+1
      end do
   end do

   ! свободные члены
   k=1
   do i=1,n
      do j=1,n; u(k,nn1)=g(i,j); k=k+1; end do
   end do
   ! решение системы
   call gaussc(nn,1,1,u,dd)

   ! упаковка результатов в матрицу
   m=1
   do j=1,n
      do i=1,n; d(i,j)=dd(m); m=m+1; end do
   end do

   deallocate(u,dd)
end subroutine msqreq
```

Для системы $H_2/H_2/3$ с коэффициентом загрузки $\rho = 0.9$ и коэффициентами вариации исходных распределений $v_A = 0.4$ и $v_B = 3$ (данные, близкие к наихудшим в смысле сходимости) и начальной норме невязки 19.280 в трех последовательных итерациях были получены невязки 3.350, $2.30 \cdot 10^{-3}$ и $7.23 \cdot 10^{-10}$. Заметим, что исходный вариант метода для снижения нормы невязки до 10^{-6} потребовал 266 шагов, из которых 104 пришлось на последний порядок. Ни в одном из рассмотренных случаев не потребовалось более четырех шагов итераций.

Этот метод после очевидных переобозначений может быть применен и для остальных «сильвестровых» уравнений.

Глава 9.

Решение систем нелинейных уравнений

9.1. Основные предположения и вспомогательный аппарат

Пусть нужно найти n вещественных чисел $x_1, x_2, \ldots, x_n$, удовлетворяющих нелинейным (точнее, не предполагаемым линейными) уравнениям

$$\begin{array}{rcl} f_1(x_1, x_2, \ldots x_n) & = & 0, \\ f_2(x_1, x_2, \ldots x_n) & = & 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots & & \\ f_n(x_1, x_2, \ldots x_n) & = & 0, \end{array} \qquad (9.1.1)$$

где $\{f_i(x_1, x_2, \ldots x_n)\}$ будем считать функциями n переменных, определенными на всем n-мерном пространстве. При $n = 2$ решение системы (9.1.1) эквивалентно отысканию точек пересечения линий $f_1(x_1, x_2) = 0$ и $f_2(x_1, x_2) = 0$, причем таких точек может быть несколько (см. рис. 9.1).

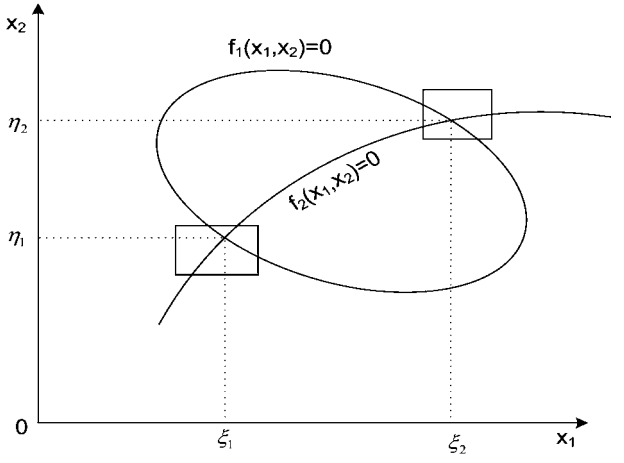

Рис. 9.1. Решения системы из двух нелинейных уравнений

В матричной записи система (9.1.1) имеет вид

$$f(x) = 0, \tag{9.1.2}$$

где f — векторная функция векторного аргумента x. В дальнейшем будем описывать методы решения применительно к виду (9.1.2). Предполагается, что корни уравнения (9.1.2) *отделены*, т. е. указаны многомерные области D (при $n = 2$ — прямоугольники $a \le x_1 \le b$, $c \le x_2 \le d$), в каждом из которых находится один и только один векторный корень (9.1.2).

Задание хороших начальных приближений требует знания специфики задачи и умения строить аппроксимации. Для получения комплексных решений следует задавать комплексные начальные приближения. Чтобы найти новое решение, нужно задать новую стартовую точку. Трудность нахождения начальных приближений быстро возрастает по числу переменных; поэтому желательно предварительно исключить максимальное число неизвестных.

С увеличением числа переменных как скорость, так и область сходимости уменьшаются, и решение удается получить лишь при очень хороших начальных приближениях. При их выборе «ничто не может заменить здравого смысла инженера» [110].

Ниже будут описаны некоторые методы последовательного приближения к корню $\hat{x}$. Специфика методов состоит в правилах перехода от результатов k-го шага $x^{(k)}$ к очередному результату $x^{(k+1)}$.

Для последующих рассуждений нам понадобится производная от вектор-функции многих переменных. Рассматривая аргумент как *вектор* x, приходим к векторной производной скалярной функции многих переменных:

$$\frac{df}{dx} = \left\{ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right\},$$

т. е. к вектору-строке.

Приближенное значение приращения скалярной функции при малых приращениях аргументов находится по формуле

$$\Delta f \approx \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \Delta x_i = \left(\frac{df}{dx}, \Delta x \right), \tag{9.1.3}$$

где круглые скобки означают скалярное произведение векторов. Применив полученные результаты к каждой строке вектора-столбца $f(x)$, получаем матрицу, которая является производной от вектор-функции $f(x)$ по векторному аргументу x:

$$J(x) = \frac{d}{dx} f(x) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix}. \tag{9.1.4}$$

Эта матрица называется *матрицей Якоби* системы функций $f_1, f_2, \ldots f_n$. Приближенное значение вектор-столбца приращений

$$\Delta f = \begin{bmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_n \end{bmatrix} \approx \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1}\Delta x_1 & + \dfrac{\partial f_1}{\partial x_2}\Delta x_2 & + \ldots & + \dfrac{\partial f_1}{\partial x_n}\Delta x_n \\ \dfrac{\partial f_2}{\partial x_1}\Delta x_1 & + \dfrac{\partial f_2}{\partial x_2}\Delta x_2 & + \ldots & + \dfrac{\partial f_2}{\partial x_n}\Delta x_n \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial f_n}{\partial x_1}\Delta x_1 & + \dfrac{\partial f_n}{\partial x_2}\Delta x_2 & + \ldots & + \dfrac{\partial f_n}{\partial x_n}\Delta x_n \end{bmatrix} = J(x)\Delta x \tag{9.1.5}$$

оказывается равным произведению матрицы $J(x)$ на столбец Δx.

В дальнейшем предполагается, что $f(x)$ и все ее производные непрерывны в области D локализации корня $\hat{x}$.

9.2. Метод Ньютона

При практическом применении любого из описанных ниже вариантов метода вначале следует задать небольшое число шагов, чтобы убедиться в сходимости процесса. При его расходимости нужно прежде всего попытаться сменить начальное приближение.

9.2.1. Основной вариант

Обозначим через x_k результат k-го шага метода. Вектор $f_k \equiv f(x_k)$ может рассматриваться как вектор *невязок* — отклонений левых частей системы (9.1.1) от нуля. Введем вектор $\Delta_k \equiv \Delta x_k$ поправок к k-му приближению. Тогда должно быть выполнено

$$f(x_k + \Delta_k) = 0. \tag{9.2.1}$$

Левая часть выражения (9.2.1) может быть представлена в виде $f(x_k) + \Delta f(x_k)$. При x_k, достаточно близком к корню $\hat{x}$, приращение вектора невязок может быть приближенно выражено через вектор поправок с помощью (9.1.5). Обозначая $J_k \equiv J(x_k)$, имеем

$$f(x_k) + \Delta f(x_k) \approx f_k + J_k \Delta_k.$$

Приравняв результат нулю, получим

$$J_k \Delta_k = -f_k, \tag{9.2.2}$$

откуда искомая поправка

$$\Delta_k = -J_k^{-1} f_k, \tag{9.2.3}$$

а очередное приближение

$$x_{k+1} = x_k - J_k^{-1} f_k, \qquad k = 0, 1, \ldots \tag{9.2.4}$$

Если последовательность векторов $\{x_k\}$ не выходит из области D и имеет предел при $k \to \infty$, то она сходится к решению системы (9.1.1): $\hat{x} = \lim_{k\to\infty} x_k$.

В обсуждавшемся выше основном варианте метода Ньютона вычисление обратной матрицы и ее обращение на каждом шаге не являются необходимыми: предпочтительнее решать линейную систему (9.2.2) относительно поправки Δ_k.

9.2.2. Модифицированный вариант

Меньшую трудоемкость шага имеет *модифицированный* метод Ньютона, счет в котором идет по формуле

$$x_{k+1} = x_k - J_0^{-1} f_k, \qquad k = 0, 1, \ldots \tag{9.2.5}$$

Здесь вычисление матрицы Якоби и ее обращение выполняются однократно.

В модифицированном методе Ньютона на любом шаге вычислений сохраняется непосредственная связь поправки с начальным приближением. Это усугубляет зависимость сходимости метода от начального приближения. Чтобы ослабить данную зависимость, но сохранить умеренную трудоемкость, можно обновлять матрицу Якоби через *несколько* шагов. В частности, имеет смысл пересчитывать матрицу Якоби, когда при использовании старой перестал наблюдаться существенный прогресс. В простейшем случае матрица Якоби пересчитывается (и обращается) через шаг. Доказано [15, с. 288], что такой процесс может порождать *кубически* сходящуюся последовательность.

Можно построить вариант модифицированного метода Ньютона с точным получением обратной матрицы Якоби на первом шаге и ее приближенным уточнением на последующих согласно разд. 7.10.

Пример 9.1. Найдем положительные решения системы

$$\begin{array}{rrrrl} x_1^2 & +x_2^2 & +x_3^2 & -1 & = 0, \\ 2x_1^2 & +x_2^2 & -x_3^2 & & = 0, \\ x_1^2 & -e^{x_2} & & +1 & = 0. \end{array} \tag{9.2.6}$$

Выберем начальное приближение $x_0 = \{0.5, 0.5, 0.5\}$. Соответствующая системе (9.2.6) матрица Якоби имеет вид

$$J(x) = \begin{bmatrix} 2x_1 & 2x_2 & 2x_3 \\ 4x_1 & 2x_2 & -2x_3 \\ 2x_1 & -e^{x_2} & 0 \end{bmatrix}.$$

Для исходной точки $x^{(0)} = (0.5, 0.5, 0.5)$ вычисляем

$$f_0 = f(x_0) = \begin{bmatrix} -0.2500 \\ 0.5000 \\ -0.3987 \end{bmatrix}, \quad J_0 = \begin{bmatrix} 1.0000 & 1.0000 & 1.0000 \\ 2.0000 & 1.0000 & -1.0000 \\ 1.0000 & -1.6487 & 0.0000 \end{bmatrix}.$$

Обратная матрица Якоби

$$J_0^{-1} = J^{-1}(x_0) = \begin{bmatrix} 0.2374 & 0.2374 & 0.2879 \\ 0.1440 & 0.1440 & -0.4319 \\ 0.6187 & -0.3813 & 0.1440 \end{bmatrix}.$$

Теперь вектор поправок

$$J_0^{-1} f_0 = \begin{bmatrix} -0.0055 \\ 0.2082 \\ -0.4027 \end{bmatrix} \quad \text{и} \quad x_1 = \begin{bmatrix} 0.5555 \\ 0.2918 \\ -0.9027 \end{bmatrix}.$$

Аналогично вычисляются результаты последующих итераций. В табл. 9.1 приведены пошаговые поправки для основного и модифицированного вариантов метода Ньютона, а также для альтернирующего варианта — с обновлением обратной матрицы Якоби через шаг. Базовый вариант уже после третьего шага входит в режим квадратичной сходимости, и предельная точность для 16-разрядной десятичной сетки достигается на шестом шаге. Решение системы мы приведем с четырьмя знаками после точки: $x_1 = 0.5386$, $x_2 = 0.2547$, $x_3 = 0.8031$.

В модифицированном варианте сходимость близка к линейной; по x_3 она имеет колебательный характер, и четыре верных знака не получены даже после 16 шагов.

В альтернирующем варианте также достигается квадратичная сходимость (только после шагов с перевычисляемой матрицей Якоби). Этот вариант доставляет решение практически за то же число шагов, что и основной метод, но имеет заметно меньшую трудоемкость.

9.2.3. Преобразованная система

В систему (9.2.6) переменные x_1 и x_3 входят только через свои квадраты. Следовательно, ей удовлетворяют и $x_1 = -0.5386$, и $x_3 = -0.8031$. Учитывая комбинации положительных и отрицательных корней, можно утверждать, что система (9.2.6) имеет не менее четырех решений. Для отыскания всех решений удобнее ввести новые переменные $u_1 = x_1^2$ и $u_3 = x_3^2$ — тогда система будет «ближе» к линейной. В частности, матрица Якоби примет вид

$$J(u) = \begin{bmatrix} 1 & 2u_2 & 1 \\ 2 & 2u_2 & -1 \\ 1 & -e^{u_2} & 0 \end{bmatrix},$$

Таблица 9.1. Решение исходной системы

k	Ньютон			Альтернирующий			Модифицированный		
	Δx_1	Δx_2	Δx_3	Δx_1	Δx_2	Δx_3	Δx_1	Δx_2	Δx_3
1	-5.55e-2	2.08e-1	-4.03e-1	-5.55e-2	2.08e-1	-4.03e-1	-5.55e-2	2.08e-1	-4.03e-1
2	1.62e-2	3.60e-2	9.40e-2	1.40e-2	2.69e-2	1.68e-1	1.40e-2	2.69e-2	1.68e-1
3	7.11e-4	1.06e-3	5.61e-3	2.78e-3	1.01e-2	-7.13e-2	1.76e-3	7.24e-3	-1.05e-1
4	8.76e-7	8.99e-7	1.98e-5	8.30e-7	1.33e-6	-3.00e-4	7.68e-4	2.09e-3	6.11e-2
5	1.01e-12	6.51e-13	2.43e-10	1.29e-12	1.43e-12	5.61e-8	2.08e-4	6.14e-4	-3.77e-2
6	2.16e-17	2.30e-18	4.71e-17	2.16e-17	2.28e-18	-2.09e-11	6.33e-5	1.81e-4	2.25e-2
7				2.16e-17	2.30e-18	4.71e-17	1.86e-5	5.37e-5	-1.38e-2
8							5.52e-6	1.59e-5	8.31e-3
9							1.63e-6	4.71e-6	-5.05e-3
10							4.84e-7	1.40e-6	3.06e-3
11							1.43e-7	4.13e-7	-1.86e-3
12							4.25e-8	1.22e-7	1.12e-3
13							1.26e-8	3.63e-8	-6.82e-4
14							3.73e-9	1.07e-8	4.13e-4
15							1.10e-9	3.18e-9	-2.51e-4
16							3.27e-10	9.41e-10	1.52e-4

так что пересчитываться в ней должен только второй столбец. В табл. 9.2 приводятся пошаговые векторы поправок для новых переменных. Заметим, что при счете в удвоенной разрядной сетке $\varepsilon_{\text{маш}} \approx 2 \cdot 10^{-16}$.

Таблица 9.2. Решение преобразованной системы

k	Ньютон			Альтернирующий			Модифицированный		
	Δx_1	Δx_2	Δx_3	Δx_1	Δx_2	Δx_3	Δx_1	Δx_2	Δx_3
1	-5.51e-2	2.08e-1	-4.03e-1	-5.51e-2	-5.51e-2	-4.03e-1	-5.51e-2	-5.51e-2	-4.03e-1
2	1.49e-2	3.60e-2	7.43e-3	1.10e-2	2.69e-2	5.48e-3	1.10e-2	2.69e-2	5.48e-3
3	5.06e-4	1.06e-3	2.53e-4	4.37e-3	1.01e-2	2.19e-3	3.12e-3	7.24e-3	1.56e-3
4	4.39e-7	8.99e-7	2.20e-7	3.95e-5	8.14e-5	1.98e-5	9.11e-4	2.09e-3	4.55e-4
5	3.18e-13	6.51e-13	1.59e-13	6.49e-7	1.33e-6	3.24e-7	2.69e-4	6.14e-4	1.34e-4
6	-1.66e-17	2.29e-18	1.95e-17	7.00e-13	1.43e-12	3.50e-13	7.94e-5	1.82e-4	3.97e-5
7				-1.65e-17	2.29e-18	1.95e-17	2.35e-5	5.37e-5	1.18e-5
8							6.96e-6	1.59e-5	3.48e-6
9							2.06e-6	4.71e-6	1.03e-6
10							6.10e-7	1.40e-6	3.05e-7
11							1.81e-7	4.13e-7	9.04e-8
12							2.03e-7	2.02e-7	-1.48e-7
13							5.35e-8	1.22e-7	2.68e-8
14							1.58e-8	3.62e-8	7.93e-9
15							4.70e-9	1.07e-8	2.35e-9
16							1.39e-9	3.19e-9	6.96e-10

Из нее усматривается заметное улучшение сходимости модифицированного метода по самой «неудобной» переменной x_3.

9.2.4. Квазиньютоновы методы

Метод Ньютона на каждом шаге требует трудоемких вычислений матрицы Якоби и решения системы линейных алгебраических уравнений. Были разработаны модификации метода Ньютона, в которых в ходе итерационного процесса либо вместо самой матрицы Якоби, либо вместо обратной ей строятся их аппроксимации. Такие методы получили название *квазиньютоновых*.

В первом случае начальную аппроксимацию B_0 матрицы Якоби получают, либо полагая ее единичной, либо аппроксимируя J_0 по конечно-разностным формулам. Затем для $k = 0, 1, \ldots$ вычисляют

$$\begin{aligned} x_{k+1} &= x_k - B_k^{-1} f_k, \\ \Delta_k &= x_{k+1} - x_k, \\ \Delta f_k &= f_{k+1} - f_k, \\ B_{k+1} &= B_k + \frac{\Delta_k - B_k \Delta_k c_k^T}{(\Delta_k c_k)}, \end{aligned} \tag{9.2.7}$$

где c_k — n-мерный вектор, определяющий специфику конкретного метода рассматриваемого класса: $c_k = \Delta_k$ соответствует первому методу Бройдена, $c_k = \Delta f_k$ — методу Пирсона, $c_k = \Delta f_k - B_k \Delta_k$ — симметричному методу первого ранга.

Во втором случае матрица J_k^{-1} аппроксимируется матрицей H_k, которая для начала выбирается единичной или обратной к конечно-разностной аппроксимации J_0. Далее вычисляют

$$\begin{aligned} x_{k+1} &= x_k - H_k f_k, \\ H_{k+1} &= H_k + \frac{(\Delta_k - H_k \Delta f_k) d_k^T}{(\Delta f_k^T d_k)}. \end{aligned} \tag{9.2.8}$$

Здесь $d_k = \Delta f_k$ дает второй метод Бройдена, а $d_k = \Delta^{(k)}$ — метод Мак-Кормика.

Для симметричных матриц Якоби существуют специальные методы из рассматриваемых классов.

9.3. Метод итераций

Пусть каким-либо способом система (9.1.1) преобразована к равносильному виду

$$\begin{array}{rcl} x_1 & = & \varphi_1(x_1, x_2, \ldots, x_n), \\ x_2 & = & \varphi_2(x_1, x_2, \ldots, x_n), \\ \ldots & \ldots & \ldots \\ x_n & = & \varphi_n(x_1, x_2, \ldots, x_n) \end{array} \tag{9.3.1}$$

или — в векторной форме — к виду

$$x = \varphi(x). \tag{9.3.2}$$

Тогда итерационный процесс нахождения корней осуществляется по формуле

$$x_k = \varphi(x_{k-1}), \qquad k = 1, 2, \ldots \tag{9.3.3}$$

Если при $k \to \infty$ все компоненты вектора стремятся к конечным пределам $\{\hat{x}_i\}$, то эти пределы являются компонентами решения (9.3.1). Имеет место следующая

Теорема 9.1. Пусть имеется замкнутая выпуклая область D, такая, что:

1. $\varphi(x)$ определена и непрерывно дифференцируема в области D.
2. Начальное приближение x_0 и все последующие, получаемые согласно (9.3.3), принадлежат области D.
3. В области D одна из норм матрицы Якоби $G_\varphi(x)$ для $\varphi(x)$ меньше единицы.

Тогда в области D существует, и притом единственное, решение $\hat{x}$ системы (9.3.2), и процесс итераций, определяемый (9.3.3), сходится к решению $\hat{x}$ системы (9.3.2).

Доказательство этой теоремы основано на неравенстве

$$\|x_k - \hat{x}\| \le \|G_\varphi(x)\| \cdot \|x_{k-1} - \hat{x}\|,$$

обеспечивающем (если решение существует) в условиях теоремы убывание нормы вектора погрешности и, следовательно, сходимость к решению. Из данной теоремы как частный случай ($\varphi_1, \varphi_2, \ldots, \varphi_n$ — линейные функции своих переменных и, значит, $J_\varphi(x)$ постоянна) следуют условия сходимости метода простой итерации для линейных систем. В этом случае область D расширяется на все n-мерное пространство.

Для оценки близости результатов очередной итерации к решению можно воспользоваться условиями разд. 7.9 с заменой $\|A\|$ на $q \geq \max\limits_{x \in D} \|G(x)\|$, $0 < q < 1$.

Для систем нелинейных уравнений, как и в линейном случае, возможен вариант расчетной схемы типа Зейделя, когда значения только что полученных переменных используются для уточнения последующих в пределах одной итерации.

Пример 9.2. Рассмотрим систему

$$\begin{aligned} x_1 &= 0.2(\sin x_1 + \cos x_2 + e^{-x_3}), \\ x_2 &= 0.3(\cos x_1 + e^{-x_2} + \sin x_3), \\ x_3 &= 0.25(e^{-x_1} + \sin x_2 + \cos x_3). \end{aligned} \tag{9.3.4}$$

Покажем, что в области $D: \ 0 \leq x_i \leq 1, \quad i = \overline{1,3}$ существует единственное решение системы (ξ_1, ξ_2, ξ_3), и процесс итерации сходится к этому решению при любом выборе векторного начального приближения $x^{(0)}$. Действительно, для произвольной точки $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) \in D$ справедливы неравенства $0 \leq \sin x_i^{(0)} \leq 1$, $0 \leq \cos x_i^{(0)} \leq 1$, $0 \leq \exp(-x_i^{(0)}) \leq 1$ для всех i, и поэтому очевидно, что $0 \leq x_1^{(1)} \leq 0.6$, $0 \leq x_2^{(1)} \leq 0.9$, $0 \leq x_3^{(1)}) \leq 0.75$, т. е. точка $x^{(1)}$ также лежит в области D. Отсюда следует, что если $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) \in D$, то для всех k точка $(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}) \in D$, т. е. выполнено условие 2 теоремы 9.1. Здесь очевидно, что первая норма матрицы G_φ не превосходит величины

$$0.3(\max|-\sin x_1| + \max|e^{-x_2}| + \max|\cos x_3|) = 0.3 \cdot (1+1+1) = 0.9 < 1,$$

и тем самым выполнено условие 3 той же теоремы. Условие 1, очевидно, выполнено, так как правые части системы определены и непрерывно дифференцируемы во всем пространстве, а не только в области D.

В приведенных ниже результатах итерации (простой и Зейделя) в качестве начального приближения была выбрана точка $x^{(0)} = (0,0,0)$. Результаты расчета приведены в табл. 9.3.

Таблица 9.3. Решение системы (9.3.4) итерационными методами

Номер шага	Простые итерации			Зейделевы		
	x_1	x_2	x_3	x_1'	x_2'	x_3'
1	0.4000	0.6000	0.5000	0.4000	0.5763	0.5536
2	0.3643	0.5848	0.5281	0.3605	0.6071	0.5296
3	0.3560	0.5987	0.5276	0.3526	0.5966	0.5319
4	0.3529	0.5971	0.5320	0.3520	0.5990	0.5322
5	0.3520	0.5988	0.5317	0.3516	0.5987	0.5322
6	0.3517	0.5985	0.5322	0.3516	0.5987	0.5322
7	0.3516	0.5988	0.5322	-	-	-
8	0.3516	0.5987	0.5322	-	-	-

В методе простой итерации четыре верных знака после запятой были получены на восьмом шаге, а в методе Зейделя — на пятом.

Для преобразования системы нелинейных уравнений к *итерационному* виду, обеспечивающему сходимость итераций, нужно вычислить модули частных производных левых частей каждого уравнения в окрестности предполагаемого решения и попытаться разрешить каждое уравнение относительно переменной с наибольшим модулем производной.

Если система уравнений задана в общем виде, т. е. как векторное равенство

$$f(x) = 0,$$

то итерирующую вектор-функцию — см. формулу (9.3.2) — можно искать в виде

$$\varphi(x) = x - G^{-1}(x^{(0)})f(x).$$

Предполагается, что матрица $G(x^{(0)})$ неособая. В этом случае счет идет по схеме, эквивалентной модифицированному методу Ньютона.

9.4. Одна специальная система

Наряду с общими методами решения нелинейных систем могут применяться частные, учитывающие специфику конкретной задачи. Весьма поучительно применение описанного в книге [104] метода к расчету параметров гиперэкспоненциального распределения времени обслуживания.

Запись дополнительной функции распределения в виде

$$\bar{F}(t) = \sum_{i=1}^{n} y_i e^{-\mu_i t} \qquad (9.4.1)$$

позволяет представить исходный процесс проходящим одну из n альтернативных фаз. Здесь $\{y_i\}, \quad 0 \le y_i \le 1, \quad \sum_{i=1}^{n} y_i = 1$ интерпретируются как вероятности выбора i-й фазы, а $\{\mu_i\}$ — как параметры показательного распределения времени пребывания в ней. Диаграмма гиперэкспоненциального распределения приведена на рис. 9.2.

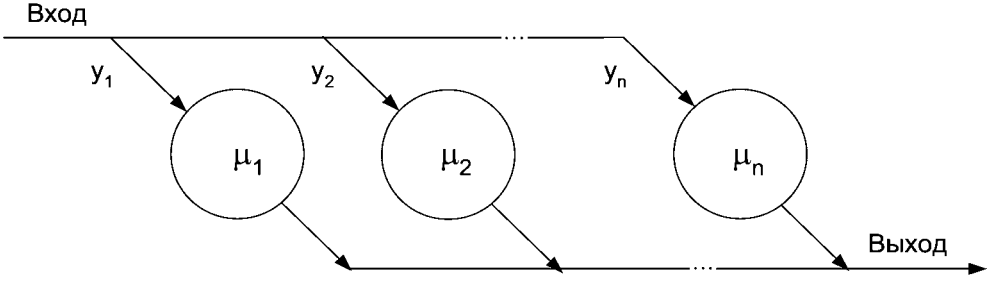

Рис. 9.2. Гиперэкспоненциальное распределение

Среди названных величин могут быть комплексные (попарно сопряженные), что подчеркивает фиктивный характер расщепления процесса на фазы. Допустимость таких параметров была впервые отмечена Д. Коксом в 1955 г. и в дальнейшем многократно подтверждалась осмысленностью конечных результатов расчетов. Указанная аппроксимация придает процессу обслуживания сильно упрощающее расчеты *марковское свойство* независимости распределения оставшейся длительности обслуживания в фазе от уже истекшей.

Будем искать параметры аппроксимации (9.4.1), приравнивая ее моменты, выраженные через упомянутые параметры, заданным значениям $\{\tilde{f}_i\}$, $i = \overline{1, 2n-1}$. Число уравнений равно количеству *независимых* параметров (9.4.1) (на них наложено естественное ограничение $\sum_{i=1}^{n} y_i = 1$). Применение формулы для моментов показательного распределения

$$b_k = k!/\mu^k, \qquad k = 1, 2, \ldots$$

приводит к системе уравнений относительно параметров аппроксимации

$$\begin{array}{lllll} y_1 & + \; y_2 & +\ldots+ & y_n & = 1, \\ y_1/\mu_1 & + \; y_2/\mu_2 & +\ldots+ & y_n/\mu_n & = \tilde{f}_1, \\ y_1/\mu_1^2 & + \; y_2/\mu_2^2 & +\ldots+ & y_n/\mu_n^2 & = \tilde{f}_2/2\,!, \\ \ldots\ldots & \ldots\ldots & \ldots & \ldots\ldots & \ldots\ldots \\ y_1/\mu_1^i & + \; y_2/\mu_2^i & +\ldots+ & y_n/\mu_n^i & = \tilde{f}_i/i\,!, \qquad i = \overline{0, 2n-1}. \end{array} \tag{9.4.2}$$

Здесь $\tilde{f}_i$ — i-й момент исходного распределения. Введем обозначения $x_i = \mu_i^{-1}$, $f_i = \tilde{f}_i/i!$. Тогда система (9.4.2) переходит в

$$\sum_{j=1}^{n} y_j x_j^i = f_i, \qquad i = \overline{0, 2n-1}. \tag{9.4.3}$$

Для решения этой системы применим метод, описанный в книге Р. В. Хэмминга [104, с. 139–141]. Введем многочлен

$$\pi(x) = \prod_{i=1}^{n} (x - x_i) = \sum_{k=0}^{n} C_k x^k. \tag{9.4.4}$$

Очевидно, для всех i имеет место $\pi(x_i) = 0$, а $C_n = 1$. Умножим первое из уравнений системы (9.4.3) на C_0, второе на C_1, ..., n-е — на C_n и сложим результаты:

$$\sum_{i=0}^{n} C_i \sum_{j=1}^{n} y_j x_j^i = \sum_{i=0}^{n} C_i f_i = \sum_{j=1}^{n} y_j \sum_{i=0}^{n} C_i x_j^i = \sum_{j=1}^{n} y_j \pi(x_j) = 0.$$

Сдвинем теперь множители $\{C_i\}$ на одну строку вниз:

$$\sum_{i=0}^{n} C_i \sum_{j=1}^{n} y_j x_j^{i+1} = \sum_{i=0}^{n} C_i f_{i+1} = \sum_{j=1}^{n} y_j \sum_{i=0}^{n} C_i x_j^{i+1} = \sum_{j=1}^{n} y_j x_j \sum_{i=0}^{n} C_i x_j^i = 0.$$

Продолжая эти преобразования до сдвига на n строк, приходим к системе уравнений

$$\begin{array}{lll} \sum\limits_{j=0}^{n} C_j f_{i+j} & = & 0, \qquad i = \overline{0, n-1}, \\ C_n & = & 1, \end{array} \tag{9.4.5}$$

решаемой одним из стандартных численных методов. Теперь согласно (9.4.4) имеем уравнение

$$\sum_{i=0}^{n} C_i x^i = 0$$

с известными коэффициентами. Решая его, находим корни $\{x_j\}$. Подстановка корней в первые n уравнений из (9.4.3) дает коэффициенты $\{y_j\}$. Наконец, вычисляем все $\mu_j = 1/x_j$.

Проверим это решение для H_2-аппроксимации плотности гамма-распределения

$$f(t) = \frac{\lambda(\lambda t)^{r-1}}{\Gamma(r)} e^{-\lambda t}$$

с моментами $f_i = r(r+1)\dots(r+i-1)/(i!\,\lambda^i)$. Система (9.4.5) теперь сводится к

$$\begin{array}{rcl} C_0 + C_1 f_1 & = & -f_2, \\ C_0 f_1 + C_1 f_2 & = & -f_3. \end{array}$$

Ее определитель

$$A = \begin{vmatrix} 1 & f_1 \\ f_1 & f_2 \end{vmatrix} = f_2 - f_1^2$$

обращается в нуль при $r = 1$, что исключает стандартное применение метода для показательного распределения — точнее, для распределений с единичным коэффициентом вариации. Далее, при успешном прохождении этого этапа имеем

$$C_0 = \frac{f_1 f_3 - f_2^2}{f_2 - f_1^2}, \quad C_1 = \frac{f_1 f_2 - f_3}{f_2 - f_1^2}.$$

Соответственно, корни квадратного уравнения $x^2 + C_1 x + C_0 = 0$ подсчитываются согласно

$$x_{1,2} = -\frac{f_1 f_2 - f_3}{2(f_2 - f_1^2)} \pm \sqrt{D}.$$

Дискриминант уравнения

$$D = \left[\frac{f_1 f_2 - f_3}{2(f_2 - f_1^2)}\right]^2 - \frac{f_1 f_3 - f_2^2}{f_2 - f_1^2}$$

равен нулю при $r = 2$, т. е. для распределения Эрланга второго порядка. В этом случае корни уравнения равны, и на заключительном этапе алгоритма определитель системы

$$\begin{array}{rcl} y_1 + y_2 & = & 1, \\ x_1 y_1 + x_2 y_2 & = & f_1 \end{array}$$

обращается в нуль. Таким образом, упомянутые типы распределений являются особыми случаями алгоритма и должны обрабатываться отдельно.

Глава 10.
Приближение функций

10.1. Постановка задачи

В практике вычислений часто требуется приближенно заменить на интервале $[a, b]$ некоторую функцию $f_1(x)$ другой функцией $f_2(x)$, более удобной для расчетов. Одна из таких задач рассматривалась в разд. 9.4. Подобная надобность регулярно возникает при расчете подробных таблиц, а также интегрировании или дифференцировании функций, заданных алгоритмически, сложными аналитическими выражениями либо таблицами (т. е. в конечном числе точек).

В роли аппроксимирующих функций обычно применяют многочлены: их легко вычислять, перемножать, дифференцировать и интегрировать. Теоретической основой для применения многочленов в качестве приближений к непрерывным функциям является

Теорема 10.1 (Вейерштрасса): если функция $f(x)$ непрерывна на отрезке $[a, b]$, то для любого $\varepsilon > 0$ можно указать многочлен $P_n(x)$ степени $n(\varepsilon)$, отвечающий условию

$$|f(x) - P_n(x)| < \varepsilon$$

при всех $\in [a, b]$.

При решении задач данного вида существенна гладкость приближаемой функции. Говорят, что функция $f \in C^n[a, b]$, если она имеет на промежутке $[a, b]$ производные по крайней мере до n-го порядка включительно.

10.2. Оценка качества приближения

Переход от исходной функции к аппроксимирующей ставит вопрос о *качестве* полученного приближения, т. е. о степени близости на интервале $[a,b]$ функций $f_1(x)$ и $f_2(x)$. Для функции $f_1(x)$, определенной на всем интервале $[a,b]$, оценку качества аппроксимации можно получить с помощью понятия *нормы функции*.

Пусть на отрезке $a \le x \le b$ рассматривается такое множество функций, что вместе с двумя произвольными функциями $f_1(x)$ и $f_2(x)$ этого множества ему принадлежит их линейная комбинация $c_1 f_1(x) + c_2 f_2(x)$, где c_1, c_2 — произвольные числа. Говорят, что на рассматриваемом множестве функций определена *норма*, если каждой функции $f(x)$ этого множества поставлено в соответствие число (норма) $\|f(x)\|$, обладающее свойствами:

1) $\|f(x)\| \ge 0$, причем $\|f(x)\| = 0$ только для $f(x) \equiv 0$;

2) $\|cf(x)\| = |c| \cdot \|f(x)\|$, где c — произвольное число;

3) $\|f_1 + f_2\| \le \|f_1\| + \|f_2\|$ — неравенство треугольника.

Приведем два примера норм:

- равномерная норма $\|f(x)\|_p = \max\limits_{a \le x \le b} |f(x)|$;
- квадратичная норма $\|f(x)\|_q = \sqrt{\int\limits_a^b [f(x)]^2\, dx}$.

Справедливость свойств 1) и 2) для обеих введенных норм непосредственно очевидна. Можно доказать, что для них справедливо и неравенство треугольника. В качестве меры «близости» двух функций $f_1(x)$ и $f_2(x)$ обычно принимают норму разности этих функций, т. е. величину $\|f_1 - f_2\|$.

В заключение обсудим соотношение между двумя рассмотренными нормами. Очевидно, что для произвольной функции $f(x)$

$$\|f\|_q \le \sqrt{b-a} \cdot \|f\|_p ;$$

поэтому из малости величины $\|f\|_p$ следует малость $\|f\|_q$. Если равномерная норма функций последовательности $f_1(x), f_2(x), \ldots, f_n(x)$ при $n \to \infty$ стремится к нулю, то стремится к нулю и квадратичная норма

этих же функций. Однако обратное утверждение неверно (см. пример в [92, с. 78]).

Оценка погрешности аппроксимации функции, заданной *таблично*, может основываться только на сравнении значений $f_2(x)$ со значениями $f_1(x)$, попавшими в таблицу. Пусть таких точек n. Тогда по аналогии с нормами погрешности непрерывной аппроксимации имеет смысл применять оценки

$$\Delta_p = \max_{1\le i\le n} |f_1(x_i) - f_2(x_i)| \qquad (10.2.1)$$

или

$$\Delta_q = \Big(\frac{1}{n}\sum_{i=1}^{n}[f_1(x_i) - f_2(x_i)]^2\Big)^{1/2} \qquad (10.2.2)$$

(для упрощения терминологии будем называть их *дискретными нормами*).

Из нескольких возможных приближений к функции $f(x)$ на отрезке $[a,b]$ (или на конечном множестве значений аргумента) лучшим считается то, для которого выбранная норма погрешности меньше.

10.3. Сортировка и поиск

Обсуждаемые ниже алгоритмы интерполяции и их использование опираются на *упорядоченные* данные.

10.3.1. Поиск в массиве

Существенной частью многих алгоритмов является поиск элемента в списке (массиве) по заданному критерию. Предположим, что все элементы массива `x(1:n)` различны и требуется найти номер элемента, равного `z`, если таковой имеется. Для поиска в среднем потребуется `n/2` проверок, а в наихудшем случае — `n`.

Если массив `x` упорядочен (для определенности по возрастанию), для поиска можно применить метод половинного деления: проверить элемент в середине диапазона и в зависимости от результатов проверки в дальнейшем повторить процедуру для левой или правой половины его. Здесь среднее число проверок составит $[\log_2 n]+1$. Приведем фрагмент программы дихотомического поиска:

```
l=1; r=n
do while(l<=n)
   i=(l+r)/2
   if (x(i)=z) then
      print *, i
      stop
   else
      if (l=r) then
         exit
      else
         if (x(i)<z) then
            l=i+1
         else
            r=i
         end if
      end if
   end if
end do
print *, 'элемент не найден'
```

Этот алгоритм поясняет происхождение приведенной выше логарифмической оценки трудоемкости поиска. С его помощью можно, например, вычислить $y(z)$ линейной интерполяцией между табличными (неравноотстоящими, но упорядоченными) значениями аргумента. Ключ к этой задаче — найти первое значение аргумента, превосходящее z.

10.3.2. Понятие о сортировках

Сортировка — это процесс расстановки элементов вектора в некотором порядке. Она имеет целью эффективное выполнение вычислений, предполагающих упорядоченность данных (поиск, построение функций распределения случайных величин и интерполяционных многочленов и т. п.), и часто облегчает интерпретацию результатов. Простейшие примеры упорядоченности — по алфавиту, по возрастанию числовых значений (например, моментов наступления предстоящих событий при имитационном моделировании). Модификация алгоритмов применительно к другим критериям упорядоченности сводится к замене проверяемых условий.

Целесообразность сортировки определяется сопоставлением преимуществ работы с упорядоченным массивом и затрат на упорядочение. Сортировка не обязательно предполагает физическое упорядочение записей: можно ограничиться упорядочением ссылок на них (как в библиотеке хранят каталожные карточки).

Признак, по которому производится упорядочение, называется *ключом*. Различные сортировки файла могут использовать разные ключи. Например, один и тот же список курсантов военного училища может быть отсортирован по их росту (для формирования парадного расчета), по весу (для подбора пар при обучении единоборствам), по среднему баллу (для справедливого распределения по местам дальнейшей службы). При сортировке записей в целом возможны два варианта действий: записи могут сопровождать ключи при всех перестановках или быть выстроены в результирующем массиве согласно финальной расстановке ключей.

Известно множество алгоритмов сортировки. Линейные алгоритмы предполагают отсутствие структуры в упорядочиваемом объекте: его элементы просматриваются подряд. Их средняя трудоемкость имеет порядок n^2, где n — длина массива. Нелинейные методы ориентированы на структурированные объекты (обычно — двоичные деревья). Все *эффективные* сортировки, т. е. требующие теоретически минимального среднего числа сравнений порядка $n \log_2 n$, являются нелинейными.

10.3.3. Линейные сортировки

При *линейном выборе с обменом* в начале первого просмотра предполагается, что первый элемент списка имеет наименьший ключ. Этот ключ вместе с его адресом пересылается в рабочую память и затем сравнивается со всеми последующими элементами, пока не встретится меньший. Тогда последний заменяет собой базу сравнения, и просмотр списка продолжается.

По окончании первого просмотра базовая запись переставляется с записью из вершины списка. Второй просмотр начинается со второй записи списка, и в его итоге следующий по величине элемент занимает вторую позицию, и т. д. Работа заканчивается, когда свое место занимает $(n-1)$-я запись:

```
subroutine chanlin(n,x)
   integer, intent(in)     :: n
   real,intent(inout)      :: x(:)
```

```
      integer i,j,k
      real p
      do i=1,n-1
         p=x(i); k=i
         do j=i+1,n
            if (x(j)<p) then
               p=x(j); k=j
            end if
         end do
         x(k)=x(i); x(i)=p
      end do
   end subroutine chanlin
```

При *пузырьковой сортировке* производится попарное сравнение соседних элементов и при необходимости — их перестановка. Число сравнений каждый раз уменьшается на единицу. При завершении очередного просмотра с нулевым числом инверсий сортировка заканчивается досрочно. В ходе этой сортировки малые элементы постепенно «всплывают» к началу массива, что и определяет название метода. Большие элементы «оседают на дно» — в первом прогоне самый большой, затем следующий по величине и т. д. Это обстоятельство можно использовать, когда достаточна частичная упорядоченность (например, нужно найти 5 наибольших чисел).

Челночная сортировка работает аналогично стандартному обмену до тех пор, пока не надо выполнять перестановку. Сравниваемая величина с меньшим ключом поднимается вверх по списку, насколько это возможно, сопоставляясь со всеми ее предшественниками по направлению к вершине (началу) списка. Если ее ключ меньше, чем у предшественника, то выполняется обмен и начинается очередное сравнение. Здесь в любой момент список выше последнего первичного сравнения упорядочен. Поэтому, когда передвигаемый элемент встречает элемент с меньшим ключом, этот процесс прекращается и нисходящее сравнение возобновляется с той позиции, с которой выполнялся первый обмен. Сортировка заканчивается, когда нисходящее сравнение захватит n-й элемент. Отмеченное выше свойство частичной упорядоченности позволяет применить для поиска места вставки метод половинного деления.

10.3.4. Быстрая сортировка

Предложенная Ч. Хоаром в 1960 г. нелинейная сортировка каждый шаг своей реализации сводит к сортировке двух массивов меньшей длины. Она основана на определении места начального элемента массива в упорядоченном множестве. Далее тот же метод можно применить к отрезкам текущей последовательности слева и справа от найденной позиции. Итак, задача может быть решена посредством *рекурсивной* процедуры.

Примеры программ сортировок (в том числе нелинейных) обсуждаются в [85].

10.4. Интерполирование

Под *интерполированием* понимается построение функции, проходящей через конечное (табличное) множество точек. Первые приложения таблиц связаны с навигацией по звездам, где они использовались для определения широты и долготы. Создание необходимой совокупности таблиц во Франции после Великой революции рассматривалось как фундаментальная задача и было связано с именем классика интерполяции — Лагранжа. Проблемы работы с таблицами актуальны и в наши дни. Для примера сошлемся на использование данных переписи населения, проводимой лишь один раз в 10 лет. О разнообразии и практической актуальности задач этого рода свидетельствуют приведенные в [75] таблицы зависимостей:

- температуры кипения разных жидкостей — от давления;
- плотности жидкостей и давления насыщенного водяного пара — от температуры;
- ускорения свободного падения, плотности атмосферы, скорости распространения звука — от высоты над земной поверхностью;
- силы света — от мощности электрической лампы;
- показателей древостоя — от возраста;
- технико-экономических показателей угледобывающей промышленности — от года.

Есть мнение [75], что в вопросах прогнозирования интерполяция и в особенности экстраполяция могут служить гидом в научном предвидении. Однако нам ближе другая точка зрения: решение задачи об *экстраполяции* (прежде всего по времени) в каждом конкретном случае требует глубокого понимания изучаемого явления. «Очень трудно что-нибудь предвидеть, особенно на будущее» [29, с. 54]. Добавим: и особенно в России!

Теорию интерполяции можно назвать наукой чтения между строк математической таблицы [37, с. 104]. Именно проблема исправления ошибок в таблицах подвигла Ч. Бэббиджа на конструирование вычислительной машины. Некоторые способы обнаружения ошибок были достаточно изощренны (например, по сопоставлению разностей и их разностей). Интересный опыт составления таблиц обсуждается в [37, с. 109–110].

Прежде чем рассматривать отдельные способы интерполяции, сформулируем некоторые общие положения:

1. Интерполяция полезна лишь при условии, что данные не содержат ошибок.

2. Данные сами по себе не могут определить интерполянт. Для фиксированного набора данных существует бесчисленное множество интерполянтов.

3. Если изучаемая функция разрывна, то интерполяцию можно проводить только на интервалах, не содержащих разрывов. Например, зависимость внутренней энергии единицы массы воды от температуры имеет разрывы при температуре 0° (точка плавления) и 100° (точка кипения) [29, с. 40]. Аналогично обстоит дело при разрыве производных.

10.4.1. Интерполяционный многочлен Лагранжа

Пусть на плоскости (xOy) задан набор точек $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, причем никакие две точки не находятся на одной вертикали. Многочлен, совпадающий с y_k при $x = x_k, \quad k = \overline{0, n}$, будем называть *интерполяционным* для этого множества точек. Известно, что существует единственный многочлен $P_n(x)$ степени не выше n, принимающий в точках $x_0, x_1, \ldots, x_n$ заданные значения $y_0, y_1, \ldots, y_n$. Этот

многочлен может быть записан, например, в виде

$$\begin{aligned} P(x) &= y_0 \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} \\ &+ y_1 \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)} \\ &\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ &+ y_n \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})}. \end{aligned}$$

Покажем, что такой многочлен единственный. В самом деле, пусть имеется другой интерполяционный многочлен $\tilde{P}(x)$ степени не выше n. Тогда многочлен $R(x) = P_n(x) - \tilde{P}(x)$ степени не выше n должен обращаться в нуль в $(n+1)$ точках $x_0, x_1, \dots, x_n$, т. е. иметь не менее $(n+1)$ корней. Поскольку это невозможно, остается предположить $R(x) \equiv 0$. Значит, $P_n(x)$ — единственный.

Заметим, что если этот многочлен сложить с многочленом вида $(x-x_0)(x-x_1)\dots(x-x_n)Q(x)$, где $Q(x)$ — произвольный многочлен, то сумма будет многочленом степени не ниже $(n+1)$, принимающим при $x = x_k$ значение y_k. Таким образом, существует бесчисленное множество многочленов, принимающих при $x = x_k$ значения y_k, $\quad k = \overline{0,n}$, но только один из них имеет степень не выше n.

Пусть на отрезке $[a,b]$ оси Ox определена функция $f(x)$, значения которой известны лишь в $(n+1)$ различных точках $x_0, x_1, \dots, x_n$. Рассмотрим теперь многочлен

$$\begin{aligned} L_f(x) &= f(x_0) \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} \\ &+ f(x_1) \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)} \\ &\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ &+ f(x_n) \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})}, \end{aligned} \qquad (10.4.1)$$

получающийся из $P_n(x)$ заменой y_k на $f(x_k)$. Этот многочлен и называется *многочленом Лагранжа* для функции $f(x)$ и точек $x_0, x_1, \dots, x_n$. Он имеет степень не выше n и совпадает с $f(x)$ во всех узлах интерполирования $\{x_k\}$. Вопрос о погрешности интерполяции в точках, *отличных от узлов*, может быть поставлен, если функция $f(x)$ определена в этих точках. Ниже рассматривается его решение для случая $f(x) \in C^{n+1}[a,b]$.

Введем многочлен $\omega(x) = (x - x_0)(x - x_1)\dots(x - x_n)$, имеющий степень $n+1$ и старший коэффициент 1.

Теорема 10.2. Пусть $f(x) \in C^{n+1}[a,b]$. Тогда для каждого $x \in [a,b]$ найдется по меньшей мере одна точка $\xi_x \in (a,b)$, для которой справедливо равенство

$$f(x) - L_f(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\omega(x). \qquad (10.4.2)$$

Действительно, для $x = x_k, \quad k = \overline{0,n}$ равенство (10.4.2) имеет место, так как обе его части обращаются в нуль. Пусть теперь $x \neq x_k, \quad k = \overline{0,n}$. Тогда $\omega(x) \neq 0$, и из равенства

$$f(x) - L_f(x) = A\omega(x) \qquad (10.4.3)$$

число A определяется единственным образом. При таком выборе числа образуем вспомогательную функцию

$$\psi(t) = f(t) - L_f(t) - A\omega(t).$$

Очевидно, что, как и $f(x)$, функция $\psi(t) \in C^{n+1}[a,b]$. В силу равенства (10.4.3) имеем $\psi(x) = 0$. Кроме того, $\psi(x_k) = 0, \quad k = \overline{0,n}$. Таким образом, функция $\psi(t)$ обращается в нуль по меньшей мере в $(n+2)$ различных точках промежутка $[a,b]$: $x_0, x_1, \dots, x_n$. По теореме Ролля, $\psi'(t)$ обращается в нуль по меньшей мере в одной точке каждого промежутка, на концах которого функция $\psi(t)$ обращается в нуль. Следовательно, $\psi'(t)$ обращается в нуль по меньшей мере в $(n+1)$ различных точках промежутка (a,b). По аналогичным соображениям $\psi''(t)$ обращается в нуль по меньшей мере в n различных точках промежутка (a,b). Продолжая эти рассуждения, приходим к выходу, что $\psi^{(n+1)}(t)$ обращается в нуль по меньшей мере в одной точке $\xi_x \in (a,b)$: $\psi^{(n+1)}(\xi_x) = 0$.

Поскольку $L_f(t)$ является многочленом степени не выше n, то $L_f^{(n+1)}(t) \equiv 0$. Так как $\omega(t)$ является многочленом степени $(n+1)$ со старшим коэффициентом 1, то $\omega^{(n+1)}(t) = (n+1)!$. Поэтому

$$\psi^{(n+1)}(t) = f^{(n+1)}(t) - 0 - (n+1)!A$$

и, следовательно, при $t = \xi_x$

$$0 = f^{(n+1)}(\xi_x) - (n+1)!A,$$

откуда введенный нами множитель

$$A = f^{(n+1)}(\xi_x)/(n+1)!.$$

Подставив найденное выражение для A в формулу (10.4.3), получим (10.4.2), и теорема 10.2 доказана. ▲

Обозначим

$$M_{n+1} \stackrel{\text{def}}{=} \max_{a\le x\le b} |f^{n+1}(x)|.$$

Тогда из (10.4.2) следует неравенство

$$|f(x) - L_f(x)| \le \frac{M_{n+1}}{(n+1)!}|\omega(x)|, \tag{10.4.4}$$

позволяющее оценивать погрешность интерполяции в каждой отдельной точке. Для оценки максимальной погрешности из (10.4.4) получаем неравенство

$$\max_{a\le x\le b} |f(x) - L_f(x)| \le \frac{M_{n+1}}{(n+1)!} \max_{a\le x\le b} |\omega(x)|, \tag{10.4.5}$$

или в терминах равномерных норм —

$$\|f(x) - L_f(x)\|_p \le \frac{M_{n+1}}{(n+1)!}\|\omega(x)\|_p. \tag{10.4.6}$$

Ниже приводятся примеры применения этих результатов. Оценим

$$\|\omega(x)\|_p = \max_{a\le x\le b} |\omega(x)|$$

в случае, когда $x_0 = a$, $x_k = x_0 + kh, \quad k = \overline{1,n}$, где $h = (b-a)/n$.

Поскольку узлами интерполяции выбраны концы отрезка $[a,b]$ и точки, делящие его на n равных частей,

$$\max_{0\le x\le b} |\omega(x)| \le h^{n+1}\frac{n!}{4} = \frac{(b-a)^{n+1}}{n^{n+1}}\frac{n!}{4}. \tag{10.4.7}$$

Итак, при указанном выше выборе узлов на отрезке $[a,b]$ оценка погрешности интерполяционного многочлена $L_f(x)$

$$\max_{0\le x\le b} |f(x) - L_f(x)| \le \frac{M_{n+1}}{(n+1)!}\frac{(b-a)^{n+1}n!}{4n^{n+1}} = \frac{M_{n+1}}{4(n+1)}\frac{(b-a)^{n+1}}{n^{n+1}}. \tag{10.4.8}$$

Пример 10.1. Известно, что

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots$$

Если для приближенного вычисления e^x на интервале $[0,1]$ пользоваться n-й частной суммой приведенного выше ряда Маклорена, то ошибка Δ_M будет не меньше $x^{n+1}/(n+1)!$, и максимальная (для $x=1$) ошибка будет не меньше $1/(n+1)!$. Итак,

$$\Delta_M > \frac{1}{(n+1)!}.$$

С другой стороны, образуем интерполяционный многочлен Лагранжа для e^x по узлам $\{0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}, 1\}$, делящим отрезок [0,1] на n равных частей. Ясно, что

$$M_{n+1} = \max_{0\le x\le 1} |(e^x)^{(n+1)}| = \max_{0\le x\le 1} |e^x| = e < 3.$$

Обозначив через Δ_L максимум модуля погрешности на отрезке [0,1], на основании формулы (10.3.8) получим

$$\Delta_L = \max_{0\le x\le 1} |e^x - L(x)| \le \frac{3}{4(n+1)} \frac{1^{n+1}}{n^{n+1}}.$$

Отсюда для отношения максимальных ошибок Δ_L и Δ_M будем иметь

$$\frac{\Delta_L}{\Delta_M} \le \frac{3/[4(n+1)]\cdot 1/n^{n+1}}{1/(n+1)!} = \frac{3}{4}\frac{n!}{n^{n+1}} = \frac{3}{4}\frac{(n-1)!}{n^n}.$$

Ниже приведены значения правой части последнего неравенства для $n = \overline{2,5}$:

$$\begin{array}{lll} n=2: & (3\cdot 1)/(4\cdot 2^2) & = 3/16; \\ n=3: & (3\cdot 2)/(4\cdot 3^3) & = 1/18; \\ n=4: & (3\cdot 6)/(4\cdot 4^4) & \approx 1/57; \\ n=5: & (3\cdot 24)/(4\cdot 5^5) & \approx 1/173. \end{array}$$

Таким образом, выигрыш в точности при одной и той же степени аппроксимирующего многочлена становится ощутимым уже при небольших степенях многочлена. Поэтому выгоднее запрограммировать вычисление интерполяционного многочлена (в особенности для обсуждаемых ниже чебышевских узлов), чем вычисление отрезка ряда Маклорена с той же степенью — см. рис. 10.1.

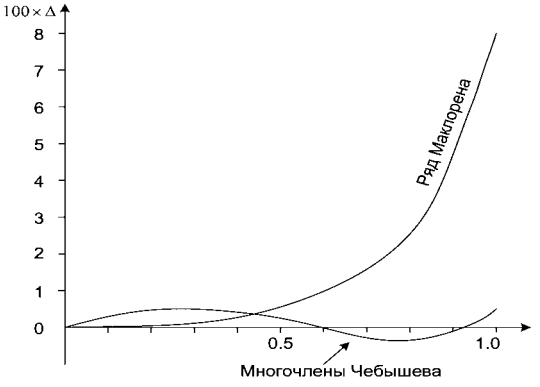

Рис. 10.1. Погрешности интерполяции и отрезка ряда Маклорена

Существуют несколько обобщений многочлена Лагранжа. Например, для интерполяционных многочленов Эрмита в узлах задаются не только значения функции, но и ее производной.

Процедура построения интерполяционного полинома при большом количестве узлов является неустойчивой ([68, с. 231]): чем выше степень полинома, тем больше размах осцилляций. Поэтому интерполирование следует применять *локально*, выбирая небольшое число наиболее подходящих узлов. Это объясняет тенденцию использовать многочлены невысокой степени (линейные, квадратичные; кубические *сплайны*).

Для интерполирования периодических функций пользуются тригонометрическими многочленами [52, с. 26]. Для функции с периодом 2π можно построить формулу, аналогичную лагранжевой (взяв синусы от каждой скобки).

10.4.2. Интерполяционные многочлены Ньютона

Мало того что при добавлении к $L_f(x)$ нового узла надо присоединять к нему еще одно слагаемое; приходится изменять и все полученные ранее. Рассмотрим способ «наращивания» интерполяционного многочлена, позволяющий при добавлении нового узла лишь формировать новое слагаемое, *не затрагивая* уже имеющиеся.

Пусть $L_n(x)$ есть интерполяционный многочлен для следующих данных:

$$\begin{array}{cccc} x_0, & x_1, & \ldots, & x_n; \\ y_0, & y_1, & \ldots, & y_n. \end{array}$$

Положим $\omega_{n+1}(x) = (x - x_0)(x - x_1)\dots(x - x_n)$. Далее, пусть $L_{n+1}(x)$ обозначает интерполяционный многочлен для данных

$$\begin{array}{cccccc} x_0, & x_1, & \dots, & x_n, & x_{n+1}; \\ y_0, & y_1, & \dots, & y_n, & x_{n+1}. \end{array}$$

Нетрудно видеть, что при любом выборе постоянной C многочлен $L_n(x) + C\omega_{n+1}(x)$ имеет степень не выше $(n+1)$ и в старых узлах $x_0, x_1, \dots x_n$ совпадает с $L_n(x)$. Для обращения этого многочлена в y_{n+1} при $x = x_{n+1}$ достаточно положить $C = (y_{n+1} - L_n(x_{n+1}))/\omega_{n+1}(x_{n+1})$. Таким образом, имеем

$$L_{n+1}(x) = L_n(x) + \frac{y_{n+1} - L_n(x_{n+1})}{\omega_{n+1}(x_{n+1})}\omega_{n+1}(x).$$

Покажем применение этого приема. Очевидно, что при $n = 0$ [данные (x_0, y_0)] получим $\omega_1(x) = x - x_0$ и $L_0(x) \equiv y_0$. При добавлении новых данных (x_1, y_1) имеем

$$L_1(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0),$$

и теперь $\omega_2(x) = (x - x_0)(x - x_1)$. Если добавить еще и данные (x_2, y_2), то

$$L_2(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + \frac{y_2 - L_1(x)}{(x_2 - x_0)(x_2 - x_1)}(x - x_0)(x - x_1).$$

Итак, интерполяционный многочлен может быть получен постепенно для *произвольного* числа узлов. Этот прием наращивания интерполяционного многочлена мы применим в одном частном случае, когда узлы интерполяции равноотстоящие и нумеруются в порядке возрастания: $x_0, x_1, \dots, x_n$, где $x_k = x_0 + kh$, $h > 0$, $k = \overline{1, n}$. Получаемые при этом интерполяционные многочлены называются *многочленами Ньютона*; их будем обозначать через $N_1(x), N_2(x)$ и т. д. Сразу заметим, что $\omega_{n+1}(x_{n+1}) = (x_{n+1} - x_0)(x_{n+1} - x_1)\dots(x_{n+1} - x_n) = [(n+1)h](nh)\dots(h) = h^{n+1}(n+1)!$.

Чтобы придать удобную форму выражению $y_{n+1} = L_n(x_{n+1})$, введем *разности* различных порядков для функции $f(x)$. Будем обозначать $y_k = f(x_k)$. Разность Δy_{k-1} назовем первой разностью в точке x_{k-1}, разность $\Delta^2 y_{k-1} = \Delta y_k - \Delta y_{k-1}$ — второй разностью в той же точке и т. д. Эти разности запишем в табл. 10.1.

Таблица 10.1. Конечные разности функции $y(x)$

x	y	Δy	$\Delta^2 y$	$\ldots$	$\Delta^{k-1} y$	$\Delta^k y$
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\ldots$	$\Delta^{k-1} y_0$	$\Delta^k y_0$
x_1	y_1	Δy_1	$\Delta^2 y_1$	$\ldots$	$\Delta^{k-1} y_1$	
x_2	y_2	Δy_2	$\Delta^2 y_2$	$\ldots$		
x_3	y_3	Δy_3	$\Delta^2 y_3$	$\ldots$		
$\ldots$	$\ldots$	$\ldots$				
x_{k-1}	y_{k-1}	Δy_{k-1}				
x_k	y_k					

Тогда

$$\begin{aligned} N_1(x) = L_1(x) &= y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) = y_0 + \frac{\Delta y_0}{1!h}(x - x_0); \\ N_2(x) = L_2(x) &= \left[y_0 + \frac{\Delta y_0}{1!h}(x - x_0) \right] + \frac{y_2 - N_1(x_2)}{2!h^2}(x - x_0)(x - x_1). \end{aligned}$$

Но

$$\begin{aligned} y_2 - N_1(x_2) &= y_2 - \left(y_0 + \frac{\Delta y_0}{h}(x_2 - x_0) \right) = y_2 - (y_0 + 2\Delta y_0) \\ &= [y_2 - (y_0 + \Delta y_0)] - \Delta y_0 = (y_2 - y_1) - \Delta y_0 \\ &= \Delta y_1 - \Delta y_0 = \Delta^2 y_0 \end{aligned}$$

и, таким образом,

$$N_2(x) = y_0 + \frac{\Delta y_0}{1!h}(x - x_0) + \frac{\Delta^2 y_0}{2!h^2}(x - x_0)(x - x_1).$$

Продолжение аналогичных рассуждений приводит к формуле

$$\begin{aligned} N_k(x) &= y_0 + \frac{\Delta y_0}{1!h}(x - x_0) + \frac{\Delta^2 y_0}{2!h^2}(x - x_0)(x - x_1) + \ldots \\ &+ \frac{\Delta^k y_0}{k!h^k}(x - x_0)(x - x_1)\ldots(x - x_{k-1}). \end{aligned} \tag{10.4.9}$$

Многочлен вида (10.4.9) и называется *интерполяционным многочленом Ньютона* порядка k.

Полагая $(x - x_0)/h = t$, формулу (10.4.9) можно переписать в стандартной форме

$$N_k(t) = y_0 + \frac{\Delta y_0}{1!}t + \frac{\Delta^2 y_0}{2!}t(t-1) + \ldots + \frac{\Delta^k y_0}{k!}t(t-1)\ldots(t-k+1). \tag{10.4.10}$$

Очевидно, что t есть «число шагов» длины h, необходимое для достижения точки x из x_0.

Интерполяционные многочлены Ньютона находят широкое применение в практике расчетов с таблицами, составленными для равноотстоящих значений аргумента.

Пример 10.2. Вычислим четыре знака $\operatorname{ctg} 0.00205$ с помощью четырехзначной табл. 10.2 функции $y = \operatorname{ctg} x$.

Таблица 10.2. Значения функции $y = \operatorname{ctg} x$ и ее конечные разности

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
0.0020	500.0	-23.8	2.1	-0.1
0.0021	476.2	-21.7	2.0	
0.0022	454.5	-19.7		
0.0023	434.8			

В нашем случае $t = (x - x_0)/h = 0.5$, откуда коэффициенты при последующих разностях

$$\begin{aligned} \frac{t(t-1)}{2!} &= \frac{0.5(-0.5)}{2} = -0.125, \\ \frac{t(t-1)(t-2)}{3!} &= -0.125 \cdot \frac{-1.5}{3} = -0.0625. \end{aligned}$$

Согласно формуле (10.4.10) имеем

$$\begin{aligned} N_0(0.00205) &= y_0 = 500.0; \\ N_1(0.00205) &= N_0(0.00205) - 23.8 \cdot 0.5 = 500.0 - 11.9 = 488.1; \\ N_2(0.00205) &= N_1(0.00205) - 2.1 \cdot 0.125 = 488.1 - 0.3 = 487.8. \end{aligned}$$

Учет третьей и последующих разностей не изменяет последнего результата (с девятью верными цифрами $\operatorname{ctg} 0.00205 = 487.804190$).

Для интерполирования *в конце таблицы* применяется другой вариант формулы Ньютона. Запишем искомый интерполяционный многочлен в форме

$$\begin{aligned} \tilde{N}_k(x) &= a_0 + a_1(x - x_k) + a_1(x - x_k)(x - x_{k-1}) + \dots \\ &+ a_k(x - x_k)(x - x_{k-1}) \dots (x - x_1) \end{aligned}$$

и определим коэффициенты разложения из условия

$$\tilde{N}_k(x_i) = y_i = f(x_i).$$

Полагая $x = x_k$, убеждаемся, что $a_0 = y_k$. При $x = x_{k-1}$ имеем

$$y_{k-1} = y_k + a_1(x_{k-1} - x_k).$$

Но $x_{k-1} - x_k = -h$. Значит,

$$a_1 = \frac{y_k - y_{k-1}}{h} = \frac{\Delta y_{k-1}}{h}.$$

Далее,

$$\begin{aligned} y_{k-2} &= y_k + \frac{\Delta y_{k-1}}{h}(x_{k-2} - x_k) + a_2(x_{k-2} - x_k)(x_{k-2} - x_{k-1}) \\ &= y_k + \frac{y_k - y_{k-1}}{h} \cdot (-2h) + a_2(-2h)(-h) \\ &= y_k - 2y_k + 2y_{k-1} + 2a_2h^2, \end{aligned}$$

откуда

$$a_2 = \frac{1}{2h^2}(y_k - 2y_{k-1} + y_{k-2}) = \frac{1}{2h^2}[(y_k - y_{k-1}) - (y_{k-1} - y_{k-2})] = \frac{\Delta^2 y_{k-2}}{2!h^2}.$$

Продолжая последовательные подстановки, убеждаемся в справедливости разложения

$$\begin{aligned} \tilde{N}_k(x) &= y_k + \frac{\Delta y_{k-1}}{1!h}(x - x_k) + \frac{\Delta^2 y_{k-2}}{2!h^2}(x - x_k)(x - x_{k-1}) + \dots \\ &+ \frac{\Delta^k y_0}{k!h^k}(x - x_k)(x - x_{k-1}) \dots (x - x_1). \end{aligned}$$

Обозначив $(x - x_k)/h = t$, перепишем $\tilde{N}_k(x_k + th) = \tilde{N}_k(t)$ в виде

$$\tilde{N}_k(t) = y_k + t\frac{\Delta y_{k-1}}{1!} + t(t+1)\frac{\Delta^2 y_{k-2}}{2!} + \dots + t(t+1)\dots(t+k-1)\frac{\Delta^k y_0}{k!}. \tag{10.4.11}$$

Оценим погрешность интерполяционного многочлена Ньютона. Разные способы построения интерполяционных многочленов Лагранжа и Ньютона для заданной таблицы дают тождественные интерполянты, что следует из единственности интерполяционного многочлена степени n. Значит, формула (10.4.7) для погрешности остается в силе. Если предполагать, что $x_0 < x < x_n$, то на основании (10.4.8) получим

$$|f(x) - N_{f,n}(x)| \le \frac{\max|f^{(n+1)}(x)|}{(n+1)!} \cdot \frac{h^{n+1} n!}{4} = \frac{\max|f^{(n+1)}(x)|}{4(n+1)} h^{n+1}. \tag{10.4.12}$$

При затруднениях в аналитическом нахождении производных принимают

$$f^{(n+1)}(x) \approx \frac{\Delta^{n+1} f(x)}{h^{n+1}}.$$

Пример 10.3. Пусть необходимо выбрать шаг h для аргумента таблиц некоторой функции с тем, чтобы линейная интерполяция ($n = 1$) давала ошибку не более $10^{-5}/2$. Это приводит к требованию

$$\frac{\max|f''(x)|h^2}{8} \le \frac{1}{2}10^{-5},$$

или

$$h \le 2\Big(\frac{10^{-5}}{\max|f''(x)|}\Big)^{1/2} = 0.002\Big(\frac{10}{\max|f''(x)|}\Big)^{1/2}.$$

Например, при составлении таблиц функции $\sin x$ достаточно выбрать $h \le 0.006$.

Центральные формулы (Стирлинга и Бесселя) могут быть получены преобразованием формул Ньютона.

10.4.3. Обратная интерполяция

Пусть имеется таблица монотонной функции $y = f(x)$. Задача *обратного* интерполирования состоит в определении аргумента x по заданному значению функции y. Эта задача может быть решена обсуждавшимися выше средствами прямой интерполяции, если в исходной таблице поменять местами x и y. Разумеется, потребуется заново отсортировать данные. Среди чисел $\{y_i\}$ не должно быть одинаковых!

Обсудим теперь способы, не требующие упомянутой инверсии. Предположим, что функция $f(x)$ монотонна и указанное значение y лежит между $y_0 = f(x_0)$ и $y_1 = f(x_1)$. Заменив функцию y первым интерполяционным многочленом Ньютона, получим

$$y = y_0 + \frac{\Delta y_0}{1!}t + \frac{\Delta^2 y_0}{2!}t(t-1) + \ldots + \frac{\Delta^k y_0}{k!}t(t-1)\ldots(t-k+1).$$

Отсюда находим итерационную формулу вида

$$t = \frac{y - y_0}{\Delta y_0} - \frac{\Delta^2 y_0}{2!}t(t-1) - \ldots - \frac{\Delta^k y_0}{k!}t(t-1)\ldots(t-k+1). \quad (10.4.13)$$

За начальное приближение к t можно принять $t_0 = (y - y_0)/\Delta y_0$. Вычислив t с достаточной точностью, полагают $x = x_0 + th$.

Обычно при обратной интерполяции ограничиваются многочленом 3-го порядка.

Пример 10.4. В табл 10.3 приведены значения интеграла вероятностей $y = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\, dt$. При каком x значение $y = 0.5$?

Таблица 10.3. Значения интеграла вероятностей

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
0.47	0.493745256	0,009004673	-86017	-942
0.48	0.502749929	8918556	-86959	
0.49	0.511668585	8831697		
0.50	0.520500281			

Начальное значение

$$t_0 = \frac{0.5 - 0.493745256}{0.009004673} = 0.694610930.$$

Подставляя числовые коэффициенты из табл. 10.3 в (10.4.13), получаем формулу

$$t = a_0 - a_1 t(t-1) - a_2 t(t-1)(t-2),$$

в которой $a_0 = t_0 = 0,694610930$; $a_1 = -0.477624984 \cdot 10^{-2}$; $a_2 = -0.1743694113 \cdot 10^{-4}$. В результате трех итераций находим $x = 0.476936007$.

Обратная интерполяция широко применяется для нахождения корней уравнений по таблице значений левой части. Выбор ее методов ограничивается неравномерностью распределения значений аргумента.

10.4.4. Сплайны

Обычная полиномиальная интерполяция является глобальной в том смысле, что через все заданные точки должен проходить единственный (общий) интерполянт. При добавлении данных приходится увеличивать степень полинома.

Иначе обстоит дело с кусочно-полиномиальными функциями (*сплайнами*), которые позволяют провести гладкую кривую через сдвигаемые наборы из фиксированного количества точек. Сплайн есть математический образ гибкой линейки, деформация которой позволяет

провести кривую через заданные точки. Если закрепить такую линейку в двух соседних узлах с заданными углами наклона α и β, то между точками закрепления она примет форму $S(x)$, минимизирующую ее потенциальную энергию. Из теории упругости известно, что четвертая производная от $S(x)$ должна быть равна нулю. Отсюда следует, что $S(x)$ есть многочлен третьей степени (кубический).

Эрмитовым кубическим интерполянтом является кусочно-кубический интерполянт с непрерывной производной, кубическим сплайном — с двумя непрерывными производными.

Программы кусочно-кубической интерполяции используют пару подпрограмм. Одна из них вычисляет необходимые параметры аппроксимации и для заданного набора данных используется однократно. Другая выполняется при каждом запросе на интерполяцию. Она относит аргумент к одному из интервалов и вычисляет значение интерполянта.

Чтобы построить кубический сплайн, надо задать коэффициенты всех образующих его кубических многочленов

$$q_i(x) = k_{i1} + k_{i2}x + k_{i3}x^2 + k_{i4}x^3.$$

Пусть m — количество интервалов аппроксимации. Первые $2m$ условий суть условия непрерывности:

$$\begin{aligned} q_i(x_i) &= y_i, \qquad i = \overline{1,m}, \\ q_{i+1}(x_i) &= y_i, \qquad i = \overline{0,m-1}. \end{aligned}$$

Следующие $2m-2$ условий определяют равенство первых и вторых производных функции на стыках участков:

$$\begin{aligned} q'_{i+1}(x_i) &= q'_i(x_i), \qquad i = \overline{1,m-1}, \\ q''_{i+1}(x_i) &= q''_i(x_i), \qquad i = \overline{1,m-1}. \end{aligned}$$

Для замыкания задачи нужны еще два уравнения. На свободных концах кривизну линии естественно приравнять нулю:

$$q''_1(x_0) = 0, \qquad q''_m(x_m) = 0.$$

Полученный таким образом сплайн называют «естественным кубическим».

Специальный выбор вида кубических многочленов позволяет упростить задачу построения сплайна. Если отдельные кубические уравнения имеют вид

$$q_i(x) = ty_i + \bar{t}y_{i-1} + \Delta x_i[(k_{i-1} - d_i)t\bar{t}^2 - (k_i - d_i)t^2\bar{t}], \quad i = \overline{1,m},$$

где

$$\Delta x_i = x_i - x_{i-1}, \qquad t = \frac{x - x_{i-1}}{\Delta x_i}, \qquad \bar{t} = 1 - t,$$
$$\Delta y_i = y_i - y_{i-1}, \qquad d_i = \frac{\Delta y_i}{\Delta x_i},$$

то каждое из уравнений содержит только два неизвестных коэффициента. После того как первое уравнение записано, с очередным уравнением добавляется только один новый неизвестный коэффициент. При этом для $x = x_{i-1}$ имеем $t = 0, \quad \bar{t} = 1$, а при $x = x_i$ будет $\bar{t} = 0, \quad t = 1$. При таком выборе кубических многочленов автоматически удовлетворяются все условия, кроме налагаемых на вторые производные. Для внутренних точек недостающие условия имеют вид

$$k_{i-1}\Delta x_{i+1} + 2k_i(\Delta x_i + \Delta x_{i+1}) + k_{i+1}\Delta x_i = 3(d_i\Delta x_{i+1} + d_{i+1}\Delta x_i),$$

а для двух внешних —

$$2k_0 + k_1 = 3d_1, \qquad k_{m-1} + 2k_m = 3d_m.$$

В итоге получается система линейных алгебраических уравнений с трехдиагональной матрицей

$$\begin{bmatrix} 2 & 1 & 0 & 0 & \dots & & \\ \Delta x_2 & 2(\Delta x_1 + \Delta x_2) & \Delta x_1 & 0 & \dots & & \\ 0 & \Delta x_3 & 2(\Delta x_2 + \Delta x_3) & \Delta x_2 & \dots & & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & \dots & \Delta x_m & 2(\Delta x_{m-1} + \Delta x_m) & \Delta x_{m-1} \\ & & & \dots & 0 & 1 & 2 \end{bmatrix},$$

столбцом неизвестных

$$[k_0, k_1, \dots, k_m]^T$$

и столбцом свободных членов

$$\begin{bmatrix} d_1 \\ d_1\Delta x_2 + d_2\Delta x_1 \\ d_2\Delta x_3 + d_3\Delta x_2 \\ \dots \\ d_{m-1}\Delta x_m + d_m\Delta x_{m-1} \\ d_m \end{bmatrix}.$$

Для решения трехдиагональной системы уравнений относительно коэффициентов сплайна $\{c_i\}$ можно использовать метод прогонки (см. разд. 7.8).

Описанные выше сплайны неудобны из-за своей нелокальности: значение сплайна в точке x зависит от значений $f(x_i)$ *во всех узлах*. Даже если требуется исправить одно значение, всю систему уравнений придется решать заново.

Существуют и другие сплайны, получаемые при иных граничных условиях или при использовании многочленов иных степеней. Некоторые программы сплайн-интерполяции дают дополнительные возможности (например, сделать интерполянт периодической функцией).

10.5. Чебышевские приближения

10.5.1. Проблема выбора узлов интерполяции

Теорема Вейерштрасса является типично «экзистенциальной»: она не дает никакого алгоритма построения приемлемого интерполирующего полинома. Если на фиксированном интервале интерполировать известную функцию $g(x)$ все в большем числе точек, то факториал и произведение разностей с увеличением n будут уменьшать оценку ошибки, но порядок производной будет расти. Для большинства функций величины производных увеличиваются быстрее, чем $n!$. Практически полиномиальный интерполянт будет очень плохо вести себя в точках, отличных от узлов. Поэтому обычно степень интерполянта не превосходит 4–5.

Опасности, связанные с полиномиальной интерполяцией, впервые обнаружил К. Рунге в 1901 г. Он пытался интерполировать полиномами на интервале $[-1, 1]$ функцию $y(x) = 1/(1 + 25x^2)$ при *равномерном* распределении абсцисс. Выяснилось, что при увеличении порядка n интерполяционного полинома последовательность $\{P_n(x)\}$ расходится в интервале $0.726 \leq |x| < 1$. Следовательно, равномерное распределение узлов заведомо не является наилучшим.

Оптимальный выбор узлов связан с понятием о *многочленах Чебышева*.

10.5.2. Многочлены Чебышева

Пусть n — целое неотрицательное число. Убедимся, что функция $\cos(n \arccos x)$ является многочленом степени n относительно x.

Действительно, полагая $\arccos x = \theta$, имеем

$$\begin{aligned}\tilde{T}_0(x) &= \cos 0 = 1;\\ \tilde{T}_1(x) &= \cos(\arccos x) = x;\\ \tilde{T}_2(x) &= \cos(2\arccos x) = \cos^2\theta - \sin^2\theta = x^2 - (1-x^2) = 2x^2 - 1.\end{aligned}$$

Рассмотрим тождество

$$\cos[(k+1)\theta] + \cos[(k-1)\theta] = 2\cos k\theta\cos\theta,$$

получаемое как частный случай известной формулы

$$\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}.$$

Вновь полагая $\theta = \arccos x$, получим

$$\cos[(k+1)\arccos x] + \cos[(k-1)\arccos x] = 2\cos(k\arccos x)\cdot\cos(\arccos x),$$

или

$$\tilde{T}_{k+1}(x) = 2x\tilde{T}_k(x) - \tilde{T}_{k-1}(x), \qquad k = 1, 2, \ldots$$

Прямая подстановка $k = 1$ дает уже полученный результат

$$\tilde{T}_2(x) = 2x\cdot x - 1 = 2x^2 - 1.$$

Последовательными подстановками можно получить

$$\begin{aligned}\tilde{T}_3(x) &= 4x^3 - 3x;\\ \tilde{T}_4(x) &= 2x(4x^3 - 3x) - (2x^2 - 1) = 8x^4 - 8x^2 + 1;\\ \tilde{T}_5(x) &= 2x(8x^4 - 8x^2 + 1) - (4x^3 - 3x) = 16x^5 - 20x^3 + 5x\end{aligned}$$

и т. д. Отметим свойства многочленов $\{\tilde{T}_n(x)\}$:

1. $\tilde{T}_n(x)$ есть многочлен n-й степени. Он содержит только четные степени x, если n четно, и нечетные — если n нечетно.

2. Старший коэффициент многочлена $\tilde{T}_n(x)$ при $n \geq 1$ равен 2^{n-1}.

3. Корни многочлена вещественны, различны и содержатся в открытом промежутке $-1 < x < 1$.

Докажем это последнее утверждение. Равенство $\tilde{T}_n(x) = 0$ эквивалентно $\cos(n\arccos x) = 0$. Отсюда следует, что $n\arccos x_k =$

$(2k+1)\pi/2$, $k = \overline{0, n-1}$, или $\arccos x_k = (2k+1)\pi/(2n)$ для всех $k = \overline{0, n-1}$. Окончательно получим

$$x_k = \cos\frac{2k+1}{2n}\pi, \quad k = \overline{0, n-1}.$$

Из неравенства $|\cos n\theta| \le 1$ следует, что $|\tilde{T}_n(x)| \le 1$ для $-1 \le x \le 1$. Для значений x, удовлетворяющих условию $n\theta = n\arccos x = k\pi$, $k = \overline{0, n}$, многочлен $\tilde{T}_n(x)$ принимает поочередно значения ± 1. Отсюда следует, что

$$\max_{-1\le x\le 1} |\tilde{T}_n(x)| = 1.$$

Введем *нормированные* многочлены Чебышева

$$T_n(x) = \begin{cases} \tilde{T}_0(x) = 1 & \text{при } n = 0; \\ \tilde{T}_n(x)/2^{n-1} & \text{при } n \ge 1. \end{cases}$$

Их свойства очевидным образом следуют из свойств многочленов $\{\tilde{T}_n(x)\}$. Укажем лишь некоторые из них:

а) их корни совпадают с корнями $\tilde{T}_n(x)$;

б) старший коэффициент $T_n(x)$ равен единице;

в) для $n \ge 1$ в $(n+1)$ различных точках промежутка [-1,1] $T_n(x)$ поочередно принимает значения $\pm 1/2^{n-1}$, и

$$\max_{-1\le x\le 1} |T_n(x)| = 1/2^{n-1}.$$

Отметим еще одно важное свойство многочленов Чебышева $\{T_n(x)\}$. Пусть $P_n(x)$ — любой многочлен степени n со старшим коэффициентом 1. Наибольшее значение модуля $P_n(x)$ на отрезке [-1,1] обозначим через $\max|P_n(x)| = \|P_n\|_p$ (равномерная норма). Имеет место

Теорема 10.3: из всех многочленов степени n с единичным старшим коэффициентом многочлен Чебышева порядка n имеет наименьший максимум модуля на отрезке [-1,1].

Докажем это. Предположим противное: имеется многочлен $P_n(x)$, такой, что

$$\max_{-1\le x\le 1} |P_n(x)| < 1/2^{n-1}.$$

Тогда графики $P_n(x)$ и $T_n(x)$, показанные на рис. 10.2, должны пересечься не менее чем в n различных точках.

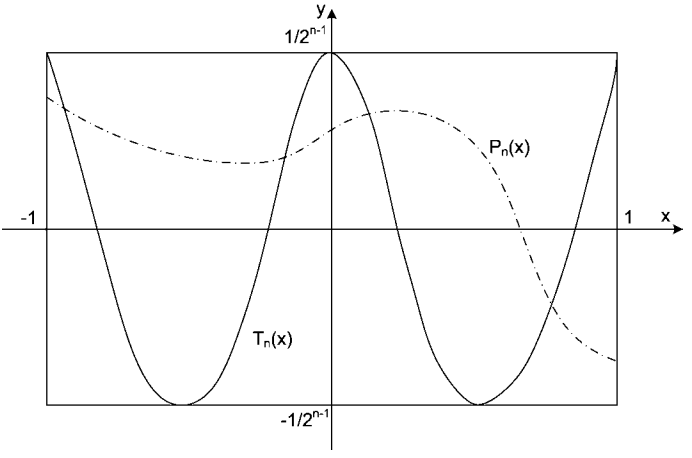

Рис. 10.2. К минимальности модуля многочлена Чебышева

Многочлен $R_n(x) = P_n(x) - T_n(x)$ имеет не менее n различных корней. Но R_n имеет степень не выше $n-1$, поскольку коэффициенты при x^n в P_n и T_n равны. Значит, наше предположение неверно, и максимальный модуль $P_n(x)$ на отрезке [-1, 1] не может быть меньше максимального модуля $T_n(x)$. ▲

10.5.3. Интерполяция по чебышевским узлам

Будем обозначать $L_f^{\omega}(x)$ интерполяционный многочлен, построенный по узлам в корнях многочлена $\omega(x) = (x-x_0)(x-x_1)\dots(x-x_n)$. Пусть ставится задача построения интерполяционного многочлена $L_f(x)$ функции $f(x)$, заданной в промежутке [-1,1]. Если выбрать узлы в корнях многочлена $T_{n+1}(x)$, т. е. принять $x_k = \cos\frac{2k+1}{2(n+1)}\pi, \quad k = \overline{0,n}$, то множитель $\omega(x)$ в формуле (10.4.5) совпадет с $T_{n+1}(x)$, и поэтому будем иметь

$$\max_{-1\le x\le 1} |f(x) - L_f^T(x)| \le \frac{\max|f^{(n+1)}(x)|}{(n+1)!}\frac{1}{2^n}. \qquad (10.5.1)$$

Иначе на основании теоремы 10.3 $\max|\omega(x)| \ge \max|T_{n+1}(x)|$, и оценка погрешности для $L_f^{\omega}(x)$ хуже (больше), чем для $L_f^T(x)$. Например, если отрезок [-1,1] разбить на n равных частей и в качестве узлов выбрать его концы и точки деления, то $|\omega(1-1/n)|$ будет иметь оценку

$$\left|\left[\left(1-\frac{1}{n}\right)-(-1)\right]\left[\left(1-\frac{1}{n}\right)-\left(-1+\frac{2}{n}\right)\right]\left[\left(1-\frac{1}{n}\right)-\left(-1+\frac{4}{n}\right)\right]\right.$$

$$\begin{aligned} \dots \quad & \left[\left(1-\frac{1}{n}\right)-\left(-1+\frac{2}{n}\right)\right]\left[\left(1-\frac{1}{n}\right)-1\right]\Bigg| \\ = \quad & \frac{2n-1}{n}\cdot\frac{2n-3}{n}\cdots\frac{1}{n}=\frac{(2n-1)!!}{n^{n+1}}, \end{aligned}$$

т. е.

$$\max|\omega(x)| \geq \frac{(2n-1)!!}{n^{n+1}}.$$

С другой стороны, полагая в формуле (10.3.8) $b-a=1-(-1)=2$, получим

$$\max_{-1\leq x\leq 1}|\omega(x)| \leq \frac{2^{n+1}}{n^{n+1}}\frac{n!}{4}=\frac{2^{n-1}}{n^{n+1}}n!$$

и, следовательно,

$$\frac{2^n(2n-1)!!}{n^{n+1}} \leq \frac{\max|\omega(x)|}{\max|T_{n+1}(x)|} \leq \frac{2^{n-1}n!/n^{n+1}}{2^{-n}}=\frac{2^{2n-1}(n-1)!}{n^n}.$$

В табл. 10.4 приведены значения крайних частей последнего неравенства для нескольких первых значений n.

Таблица 10.4. Сопоставление оценок погрешности

n	$2^n(2n-1)!!/n^{n+1}$	$2^{2n-1}(n-1)!/n^n$
2	1.500	2.000
3	1.481	2.370
4	1.641	3.000
5	1.935	3.932
6	2.377	5.267
7	3.000	7.162
8	3.866	9.844
9	5.060	13.641
10	6.704	19.025

Графическая иллюстрация приближений на примере интерполяции функции $y=(2+x^2)/(2+x)$ для n=3 приведена на рис. 10.3 (сплошная линия — по равноотстоящим узлам, штриховая — по узлам многочлена Чебышева).

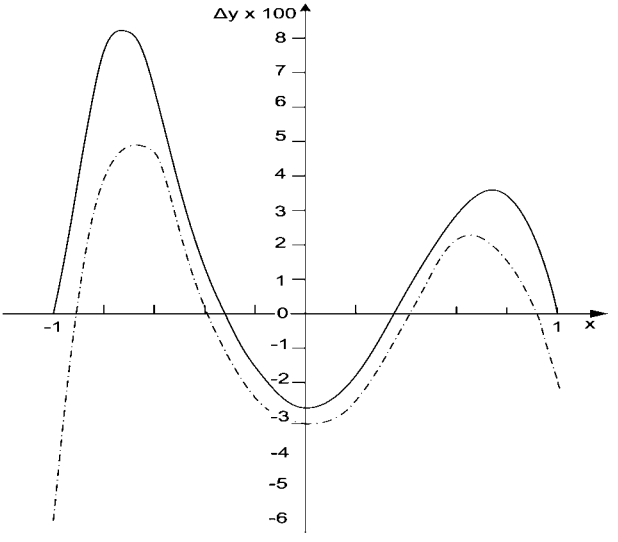

Рис. 10.3. Эффект выбора чебышевских узлов

При составлении стандартных подпрограмм некоторых элементарных функций часто применяют интерполяционный многочлен $L_f^T(x)$ вместо $L_f^\omega(x)$ или вместо отрезка ряда Тейлора. Пусть функция $f(x)$ задана на отрезке [-1,1]. Положим $x = \cos\theta$. Сложная функция $\varphi(\theta) = f(\cos\theta)$ определена на всей числовой оси, 2π-периодична и четна. Ее ряд Фурье содержит только косинусы, т. е.

$$\varphi(\theta) = f(\cos\theta) = \frac{a'_0}{2} + \sum_{n=1}^{\infty} a'_n \cos n\theta.$$

Считая $0 \le \theta \le \pi$ и полагая $\theta = \arccos x$, т. е. возвращаясь к старой переменной x, получим

$$f(x) = \frac{a'_0}{2}\tilde{T}_0 + \sum_{n=1}^{\infty} a'_n \tilde{T}_n(x) = \frac{a'_0}{2}T_0 + \sum_{n=1}^{\infty} 2^{n-1} a'_n T_n(x),$$

или

$$f(x) = \sum_{n=0}^{\infty} a_n T_n(x),$$

где

$$a_0 = a'_0/2; \quad a_n = 2^{n-1}a'_n, \quad n \ge 1.$$

Частная сумма $S_n(x)$ этого ряда, как правило, дает меньшую максимальную погрешность по сравнению с погрешностью отрезка той

же длины ряда Маклорена для функции $f(x)$. В табл. 10.5 приведены максимальные погрешности аппроксимации функции $f(x) = \sin \pi x/2$ рядом Маклорена и разложением по многочленам Чебышева.

Таблица 10.5. Максимальные погрешности аппроксимации $f(x) = \sin \pi x/2$

Аппроксимация	Наивысшая степень				
	1	3	5	7	9
Многочлены Чебышева	0.14e0	0.45e-2	0.68e-4	0.59e-6	0.34e-8
Ряд Маклорена	0.57e0	0.75e-1	0.45e-2	0.16e-3	0.35e-5

Если функция $f(x)$ определена на отрезке $[a, b]$, то, вводя новый аргумент t из условия $x = (b - a)t/2 + (b + a)/2$, придем к изучению функции, определенной на отрезке [-1,1]. Тем самым выводы о сравнении различных многочленных приближений переносятся на произвольный промежуток $[a, b]$.

10.5.4. Экономизация степенных рядов

В математическом анализе хорошо изучено и часто применяется разложение функций в степенные ряды — в частности, в ряды Тейлора. Расчет основных элементарных функций (тригонометрических, экспоненты, логарифма и т. п.) на микрокалькуляторах и компьютерах базируется на формулах приведения произвольного аргумента к стандартному интервалу. Частичные суммы таких рядов — многочлены — используются в качестве локальных аппроксимаций для исходных функций. Степени этих многочленов зависят от требуемой точности аппроксимации и положения базовой точки. В некоторых случаях такой подход мало приемлем из-за медленной сходимости рядов и существенной разницы в требуемом числе членов при разных значениях аргумента. Здесь необходима замена функции многочленом наилучшего равномерного приближения с гарантированно ограниченной погрешностью.

Процедура преобразования степенного ряда в разложение по системе многочленов Чебышева называется *экономизацией степенного ряда*.

Выразим через многочлены Чебышева последовательные степени x:

$$\begin{array}{rcl} 1 & = & T_0; \\ x & = & T_1; \\ x^2 & = & (T_0 + T_2)/2; \\ x^3 & = & (3T_1 + T_3)/4; \\ x^4 & = & (3T_0 + 4T_2 + T_4)/8; \\ x^5 & = & (10T_1 + 5T_3 + T_5)/16 \end{array} \qquad (10.5.2)$$

и т. д. (аргумент x в правых частях этой системы для краткости опущен). Пусть некоторая функция $y = f(x)$ на некотором промежутке $[a, b] \in [-1, 1]$ представлена степенным рядом. Подставляя вместо степеней x их выражения (15.1) через многочлены Чебышева и приведя подобные члены, получим разложение вида

$$f(x) = b_0 + b_1 T_1 + b_2 T_2 + \ldots + b_n T_n + \ldots$$

Можно ожидать, что результат будет близок к многочлену наилучшего равномерного приближения.

10.6. Метод наименьших квадратов

Точное проведение интерполянта через заданные точки имеет смысл только в случае, если последние не содержат ошибок. Характерной особенностью задач обработки наблюдений и экспериментов является заведомо приближенный характер результатов вследствие погрешности измерительных приборов, невозможности точно повторить условия эксперимента, случайных ошибок разного рода и т. п. Естественно стремление исследователей к накоплению как можно большего количества экспериментальных данных с тем, чтобы, некоторым образом усредняя их, получить требуемые значения малого числа искомых параметров.

В подобных случаях переходят к построению наилучшего в некотором смысле *приближения в среднем* (например, с минимальной суммой квадратов отклонений). К методу наименьших квадратов могут привести и иные соображения — например, получение корреляционных и функциональных зависимостей при анализе случайных процессов.

Пусть нам известны (скажем, из эксперимента) приближенные значения функции $f(x)$ в n точках $x_1, x_2, \ldots, x_n$. Так как эти значения

$f(x_1), f(x_2), \dots f(x_n)$ заведомо не являются точными, то нет смысла строить по ним интерполяционный многочлен Лагранжа степени $n-1$. Разумнее попытаться построить многочлен более низкой степени, *в среднем* отражающий характер функции $f(x)$.

Возьмем в качестве базисных функций для обобщенного многочлена степенные функции

$$\varphi_0 = 1, \quad \varphi_1 = x, \quad \varphi_2 = x^2, \dots, \varphi_m = x^m.$$

Тогда обобщенный многочлен

$$P_m(x) = c_0 + c_1 x + c_2 x^2 + \dots c_m x^m.$$

Будем искать такой набор коэффициентов $\{c_i\}$, при котором достигается минимум величины

$$\int_0^1 [P_m(x) - f(x)]^2 \, dx.$$

Можно показать, что эта задача сводится к решению системы

$$\begin{array}{lllll} c_0 & + & \frac{1}{2}c_1 & + \dots & \frac{1}{m+1}c_m = \int\limits_0^1 f(x)\,dx, \\ \frac{1}{2}c_0 & + & \frac{1}{3}c_1 & + \dots & \frac{1}{m+2}c_m = \int\limits_0^1 x f(x)\,dx, \\ \dots & \dots & \dots & \dots & \dots \\ \frac{1}{m+1}c_0 & + & \frac{1}{m+2}c_1 & + \dots & \frac{1}{2m+1}c_m = \int\limits_0^1 x^m f(x)\,dx. \end{array} \qquad (10.6.1)$$

Коэффициенты этой системы образуют матрицу Гильберта с элементами $\{h_{ij} = 1/(i+j+1)\}$, широко известную в математике как классический пример плохо обусловленной матрицы: (cond $H_2 \approx 20$; cond $H_6 \approx 10^7$; cond$H_{10} \approx 10^{13}$). Таким образом, базис из степенных функций непригоден для построения наилучших среднеквадратических приближений сколько-нибудь высоких степеней.

Выберем целое число m, $0 < m \le n-1$. Будем искать такой многочлен

$$P(x) = a_0 + a_1 x + \dots + a_m x^m \qquad (10.6.2)$$

степени m, чтобы сумма квадратов разностей между $f(x_k)$ и $P(x_k)$, $k = \overline{1,n}$, была наименьшей, т. е. выберем $a_0, a_1, \dots, a_m$ из условия

$$S = \sum_{k=1}^{n} [f(x_k) - (a_0 + a_1 x_k + \dots + a_m x_k^m)]^2 = \min. \qquad (10.6.3)$$

Этот подход эквивалентен минимизации дискретной квадратичной нормы погрешности аппроксимации, введенной в начале данной главы.

Величина S является функцией величин $\{a_l\}$. Для нахождения их значений, обращающих S в минимум, воспользуемся равенствами $\partial S/\partial a_k = 0, \quad k = \overline{0,m}$:

$$\begin{aligned}
-\frac{\partial S}{\partial a_0} &= 2\sum_{k=1}^{n}[f(x_k) - (a_0 + a_1x_k + \ldots + a_mx_k^m)] = 0;\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
-\frac{\partial S}{\partial a_l} &= 2\sum_{k=1}^{n}[f(x_k) - (a_0 + a_1x_k + \ldots + a_mx_k^m)]x_k^l = 0;\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
-\frac{\partial S}{\partial a_m} &= 2\sum_{k=1}^{n}[f(x_k) - (a_0 + a_1x_k + \ldots + a_mx_k^m)]x_k^m = 0.
\end{aligned}$$

После упрощений эта система принимает вид

$$\sum_{k=1}^{n}(a_0 + a_1x_k + \ldots + a_mx_k^m)x_k^l = \sum_{k=1}^{n} f(x_k)x_k^l, \qquad l = \overline{0,m}. \tag{10.6.4}$$

Система (10.6.4) есть система $(m+1)$ линейных уравнений с $m+1$ неизвестными. Более подробно она запишется в форме

$$\begin{array}{llllll}
a_0 n & +a_1\sum\limits_{k=1}^{n} x_k & +a_2\sum\limits_{k=1}^{n} x_k^2 & +\ldots & +a_m\sum\limits_{k=1}^{n} x_k^m & = \sum\limits_{k=1}^{n} f(x_k);\\
a_0\sum\limits_{k=1}^{n} x_k & +a_1\sum\limits_{k=1}^{n} x_k^2 & +a_2\sum\limits_{k=1}^{n} x_k^3 & +\ldots & +a_m\sum\limits_{k=1}^{n} x_k^{m+1} & = \sum\limits_{k=1}^{n} x_k f(x_k);\\
\cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots & \cdots\cdots\\
a_0\sum\limits_{k=1}^{n} x_k^m & +a_1\sum\limits_{k=1}^{n} x_k^{m+1} & +a_2\sum\limits_{k=1}^{n} x_k^{m+2} & +\ldots & +a_m\sum\limits_{k=1}^{n} x_k^{2m} & = \sum\limits_{k=1}^{n} x_k^m f(x_k).
\end{array} \tag{10.6.5}$$

Определитель этой системы при $x_i \neq x_j$ отличен от нуля, так что система имеет единственное решение, дающее минимум S.

Пример 10.5. Пусть даны точки на плоскости с координатами, указанными в табл. 10.6.

Таблица 10.6. К методу наименьших квадратов

k	1	2	3	4
x_k	1	3	4	6
$f(x_k)$	1	4	7	10
$\tilde{y}(x_k)$	0.884	4.576	6.422	10.114

Их расположение показано на рис. 10.4.

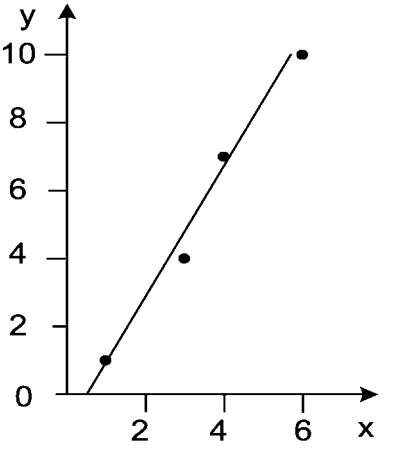

Рис. 10.4. К подбору параметров прямой по методу наименьших квадратов

Так как точки разместились почти по прямой, то естественно искать приближенное выражение $f(x)$ в виде многочлена первой степени, т. е. в виде $P(x) = a_0 + a_1 x$. В нашем случае $n = 4$, $m = 1$ и система принимает вид

$$\begin{aligned} 4a_0 + 14a_1 &= 1 + 4 + 7 + 10, \\ 14a_0 + (1^2 + 3^2 + 4^2 + 6^2)a_1 &= 1 \cdot 1 + 4 \cdot 3 + 7 \cdot 4 + 10 \cdot 6, \end{aligned}$$

или

$$\begin{aligned} 4a_0 + 14a_1 &= 22, \\ 14a_0 + 62a_1 &= 101. \end{aligned} \qquad (10.6.6)$$

Следовательно,

$$a_0 = \frac{22 \cdot 62 - 101 \cdot 14}{4 \cdot 62 - 14 \cdot 14} = \frac{-50}{52} = -0.962;$$

$$a_1 = \frac{4 \cdot 101 - 14 \cdot 22}{4 \cdot 62 - 14 \cdot 14} = \frac{404 - 308}{52} = \frac{96}{52} = 1.846.$$

Таким образом, метод наименьших квадратов приводит к линейной функции $\tilde{y}(x) = 1.846x - 0.962$. В последней строке табл. 10.6 приведены значения $\tilde{y}(x)$ для табличных значений аргумента.

Во многих приложениях не все учитываемые точки важны одинаково. В таких случаях минимизируют *взвешенную* сумму квадратов рассогласования. Веса разумно назначить обратно пропорциональными погрешности измерений.

При выравнивании статистических данных (подборе параметров распределений) метод наименьших квадратов дает оценки, близкие к *максимально правдоподобным*, т. е. доставляющим максимальную вероятность наблюдения именно исходных значений.

Аппроксимирующая функция не обязательно должна быть полиномом. Необходимо лишь, чтобы она была линейной комбинацией заданной системы линейно независимых функций:

$$P(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + \ldots a_m\varphi_m(x).$$

Ранее мы рассматривали систему $1, x, x^2, \ldots, x^m$. Системы

$$\begin{array}{cccccc} 1, & \cos x, & \cos 2x, & \ldots, & \cos mx, \\ & \sin x, & \sin 2x, & \ldots, & \sin mx \end{array}$$

обычно применяются для аппроксимации периодических функций. Расчеты значительно упрощаются, если $\{\varphi_i(x)\}$ являются полиномами, ортогональными на участке аппроксимации.

10.7. Подбор эмпирических формул

Указанная задача не является полностью алгоритмизуемой. Построение эмпирической формулы по экспериментальным данным делается в два этапа: подбор общего вида (из физических или иных теоретических соображений) и отыскание наилучших параметров. Успех определяется опытом исследователя.

При подборе зависимостей следует ввести новые переменные так, чтобы в этих переменных интересующая нас зависимость оказалась возможно ближе к линейной. Для степенной зависимости $y = ax^n$ имеем

$\log y = \log a + n \log x$; здесь желателен логарифмический масштаб по обеим осям. Для показательной $y = y_0 e^{-\alpha t}$ получаем $\ln y = \ln y_0 - \alpha t$. Масштаб по оси времени должен быть линейным.

Далее в новых координатах методом наименьших квадратов определяются параметры аппроксимации, после чего осуществляется возврат к первоначальным координатам.

Глава 11.
Методы оптимизации

Под оптимизацией в инженерном деле понимают выбор варианта, наилучшего по значению целевой функции: прочности конструкции, ее массе, мощности, надежности, производительности, стоимости и т. п.

Многомерная нелинейная оптимизация соответствующей математической модели — это одна из наиболее часто встречающихся и в то же время наиболее трудных прикладных задач. Целевых функций может быть несколько, что переводит задачу в область теории решений [105]. Здесь мы ограничимся случаем единственной целевой функции. Число проектных параметров характеризует размерность задачи оптимизации.

Нахождение оптимального решения методом прямого перебора возможно только для конечного множества вариантов и сопровождается экспоненциальным ростом трудоемкости по числу параметров. Подобные задачи решаются методами *направленного* перебора — линейного и динамического программирования, которые подробно обсуждаются в литературе по оптимизации [19, 60, 65, 105, 110].

Пожалуй, наиболее общей рекомендацией явится необходимость полного учета специфики задачи и жесткий отбор варьируемых факторов.

11.1. Базовые понятия

11.1.1. Введение

Мы будем рассматривать задачи оптимизации в конечномерных пространствах. Здесь мы ограничимся самыми общими

соображениями об их математической сути и принципиальных особенностях.

Задача оптимизации заключается в определении максимума или минимума (и соответствующих ему аргументов) вещественной функции $F(x_1, x_2, \ldots x_n)$ от n вещественных аргументов на множестве D n-мерного пространства. При этом F играет роль целевой функции, а D — ограничений. *Практически* «приходится довольствоваться улучшением известных решений, а не доведением их до совершенства. Поэтому под оптимизацией понимают скорее стремление к совершенству, которое, возможно, не будет достигнуто» [110, с. 135]. Задачи нахождения экстремума и его координат могут решаться с разной точностью. Как правило, экстремумы весьма пологи, что смягчает требования к точности определения их координат.

Описываемые ниже методы излагаются применительно к *минимизации*. Задача максимизации сводится к минимизации простой сменой знака целевой функции. Для осознания базовых принципов чрезвычайно полезна визуализация поиска оптимума по двум переменным — наглядная демонстрация поверхности отклика и изолиний уровня целевой функции (см. рис. 11.1–11.2).

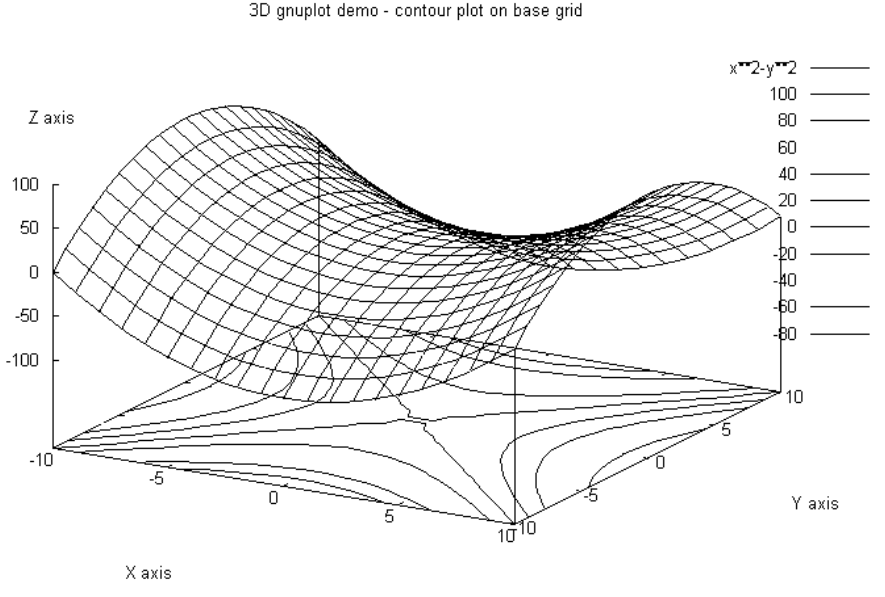

Рис. 11.1. Поверхность функции $z = x^2 - y^2$ и изолинии уровней

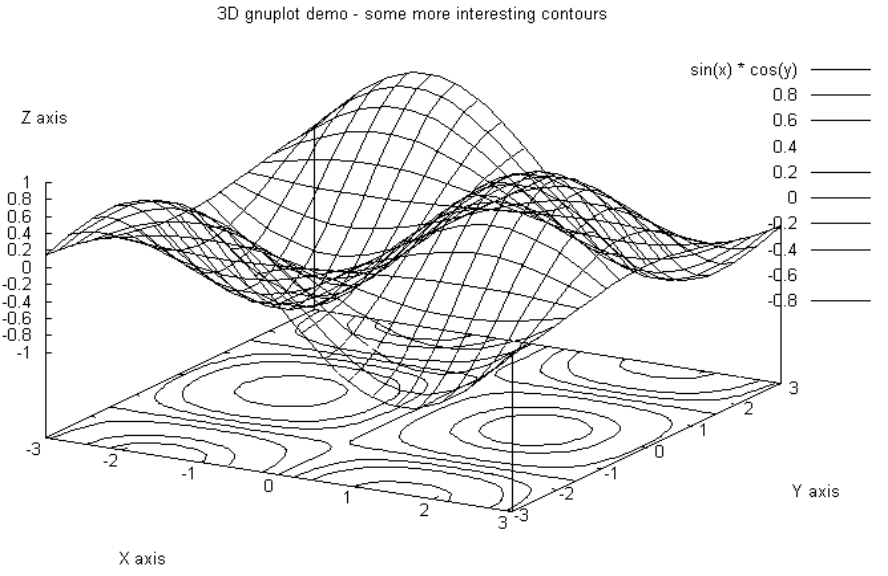

Рис. 11.2. Поверхность функции $z = \sin x * \cos y$ и изолинии уровней

Упомянутые проблемы решаются средствами универсальных математических пакетов (глава 15), а также системы Gnuplot (см. [86]). Часть рисунков данной главы заимствована из demo-файлов последней.

11.1.2. Градиент и гессиан

В процессе оптимизации нелинейной функции активно используются ее градиент и гессиан.

Градиент целевой функции определяется как вектор

$$\nabla F(x) \equiv g(x) = \left\{ \frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \ldots, \frac{\partial F}{\partial x_n} \right\}^T .$$

В основе большинства методов оптимизации лежит спуск в направлении антиградиента или близком к нему.

Гессианом (матрицей Гессе) называется составленная из вторых производных целевой функции матрица

$$G(x) = \begin{bmatrix} \dfrac{\partial^2 F}{\partial x_1^2} & \dfrac{\partial^2 F}{\partial x_1 x_2} & \cdots & \dfrac{\partial^2 F}{\partial x_1 x_n} \\ \dfrac{\partial^2 F}{\partial x_2 \partial x_1} & \dfrac{\partial^2 F}{\partial x_2^2} & \cdots & \dfrac{\partial^2 F}{\partial x_2 \partial x_n} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ \dfrac{\partial^2 F}{\partial x_n \partial x_1} & \dfrac{\partial^2 F}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 F}{\partial x_n^2} \end{bmatrix}. \qquad (11.1.1)$$

Известные свойства смешанных частных производных гарантируют симметричность матрицы $G(x)$.

С гессианом и обратной ему матрицей работают методы «второго порядка» — ньютоновы и квазиньютоновы.

11.1.3. Выпуклость и вогнутость

Сложность задачи нелинейного программирования определяется зависимостью решения от характера функции, формы и положения области допустимых решений (ОДР).

Область называется выпуклой, если отрезок прямой, соединяющий любые две точки области, целиком принадлежит этой области (см. рис. 11.3).

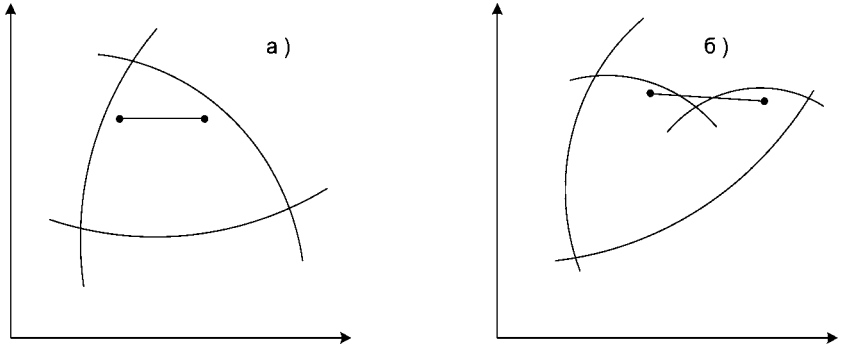

Рис. 11.3. Выпуклые и невыпуклые области

Функция называется выпуклой на выпуклой области D, если для любых ее точек $x_1, x_2 \in X$ выполнено

$$F(\theta x_1 + (1-\theta)x_2) \le \theta F(x_1) + (1-\theta)F(x_2), \qquad 0 \le \theta \le 1.$$

В двумерном случае график функции между двумя любыми точками проходит ниже хорды, соединяющей эти точки.

Выпуклые функции лежат выше любой касательной (плоскости, гиперплоскости — в зависимости от мерности пространства) к данной функции. Функция выпукла, если ее гессиан положительно определен.

Строго выпуклая функция имеет только одну точку минимума. Для минимизации такой функции необходимые условия теоремы Куна—Таккера (см. [65]) одновременно являются достаточными.

Практически приходится рассматривать следующие комбинации свойств ОДР и целевой функции:

1) ограничения несущественны; в общем случае имеется несколько локальных минимумов, среди которых надлежит выбрать глобальный;

2) активно одно из ограничений (в точке минимума $\nabla F(x) = \lambda \nabla g_i(x)$, поскольку направление $\nabla F(x)$ перпендикулярно линии уровня и границе области ограничений в данной точке); в этом случае минимум достигается на границе области;

3) имеется несколько активных ограничений и такое же число условных минимумов, среди которых надлежит выбрать наименьший.

Задача упрощается, если выпуклы как ОДР, так и минимизируемая функция.

11.1.4. «Овражные» целевые функции

Означенная проблема для придания ей наглядности ниже иллюстрируется на функции двух переменных. Поиск минимума заметно усложняется, если поверхность целевой функции имеет узкие и длинные ямы — «овраги». Есть точка зрения [10, с. 347], что подобные функции возникают при неудачном выборе математической модели явления. По [105], источниками «овражности» являются:

- неучтенные связи между параметрами;
- зависимость целевой функции от $агрегатов$ параметров;
- «жесткость» целевой функции, почти обязательная при введении штрафов за нарушение ограничений — см. раздел о жестких дифференциальных уравнениях.

Классическим примером обсуждаемой ситуации является знаменитая функция Розенброка

$$F(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \qquad (11.1.2)$$

На рис. 11.4 показан общий вид поверхности функции Розенброка.

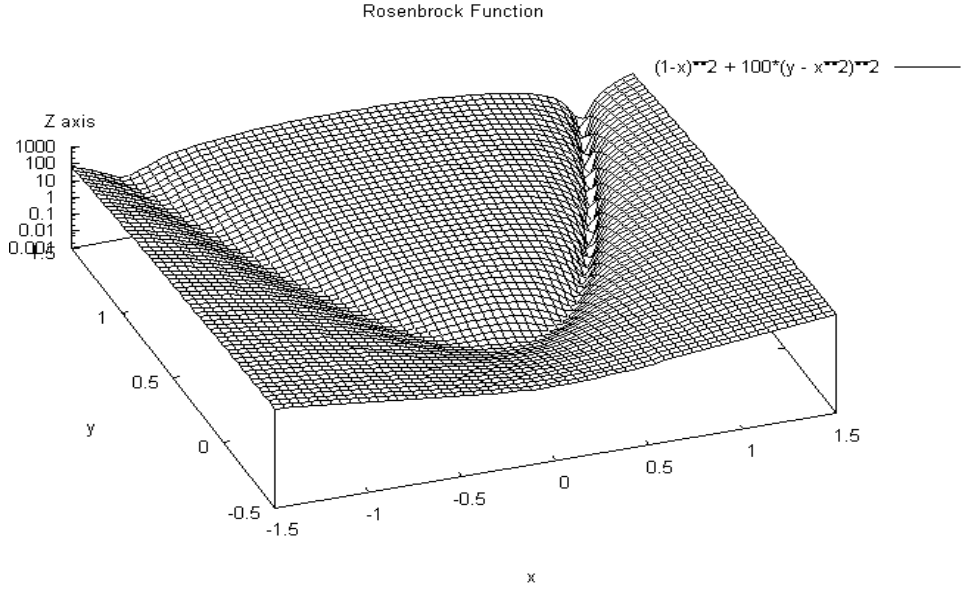

Рис. 11.4. Поверхность функции Розенброка

Рис. 11.5 с другой точки зрения (Gnuplot позволяет ее менять) демонстрирует часть оврага в районе минимума.

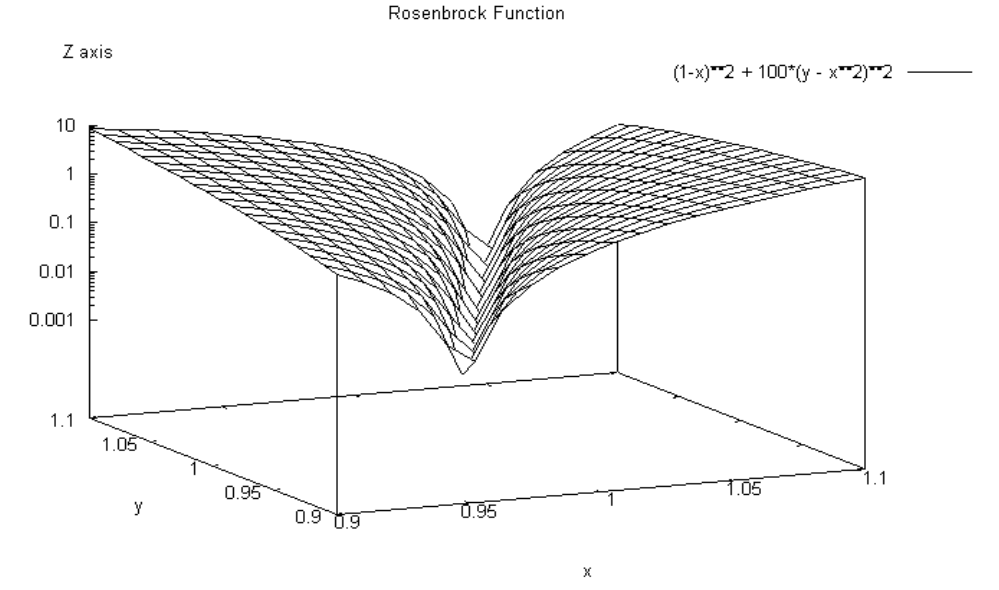

Рис. 11.5. «Овраг» функции Розенброка

Наконец, на рис. 11.6 видны изолинии функции Розенброка.

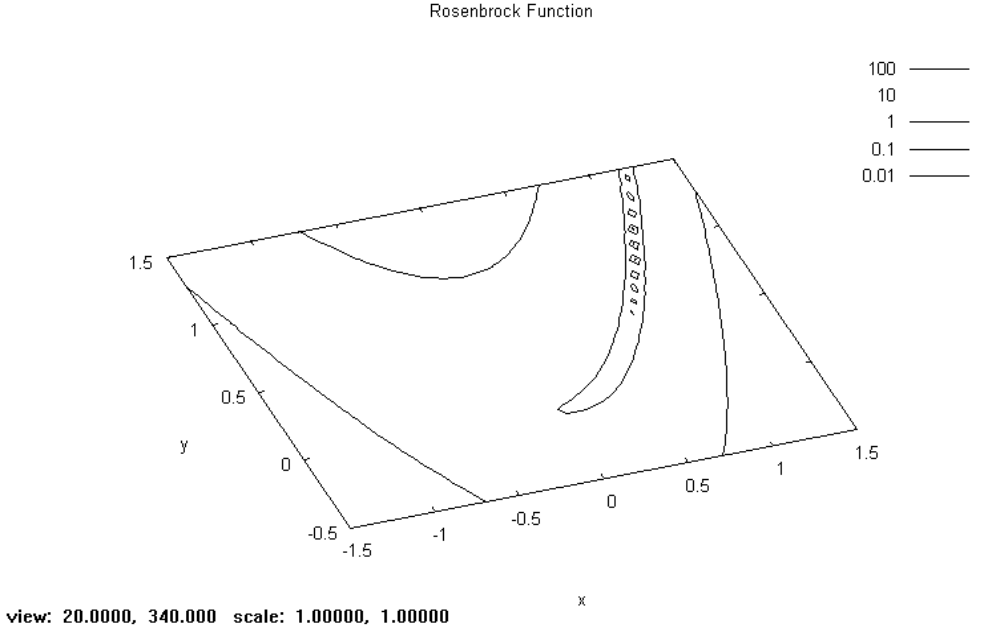

Рис. 11.6. Изолинии функции Розенброка

Эти изображения при вдумчивом анализе целевой функции (11.1.2) вполне прогнозируемы: очевидное условие минимума $x_2 = x_1^2$ (вышеупомянутая связь между оптимальными значениями параметров) порождает овраг с параболическим дном, а $x_1 = 1$ определяет точку минимума на дне оврага.

Из обсуждаемых рисунков вытекают реальное *вырождение* пространства поиска минимума (уменьшение его размерности на число дополнительных связей) и общая идея оптимизации «овражных» функций в менее очевидных ситуациях: продвигаться к минимуму *по дну оврага*. Линию последнего можно построить по конечным точкам нескольких спусков.

11.1.5. Классификация методов минимизации

Многомерные методы минимизации в зависимости от порядка используемых производных делятся на нулевые (прямые) — типа Хука—Дживса, Розенброка; первого порядка (градиентные); второго (ньютоновы и родственные им).

Прямые методы минимизации основаны на непосредственном сравнении целевой функции в различных точках пространства поиска и другую аналитическую информацию не используют. То же можно сказать о последовательном *покоординатном спуске* — аналоге метода Зейделя решения системы линейных уравнений. Покоординатный спуск совершенно неприемлем, если оси изолиний уровня наклонены к осям координат.

Градиентные методы на каждом шаге выбирают очередное направление движения к минимуму и длину шага. Первоочередным кандидатом в направление спуска служит антиградиент. При сильно вытянутых изолиниях уровня, что порождается заметным различием проекций градиента на координатные оси, продвижение к минимуму состоит из осцилляций в направлении локальных антиградиентов, даже приблизительно не указывающих направление к центру эллипсоида. В этом случае применяются *сопряженные градиенты* — направление очередного шага строится как взвешенная сумма антиградиента и направления предыдущего шага.

Градиентный спуск часто используется в комбинации с методом Ньютона для получения необходимых последнему хороших начальных приближений.

Методы *второго порядка* базируются на многомерном обобщении разложения функции многих переменных в ряд Тейлора с учетом произ-

водных до второго порядка включительно, т. е. гессиана функции. Они особенно эффективны при работе с выпуклой функцией.

Наконец, *косвенные* методы основаны на решении систем уравнений, получающихся приравниванием нулю частных производных целевой функции по переменным задачи.

Многообразие методов оптимизации порождается сложностью и необходимостью учета специфики решаемых задач. Методы характеризуются *быстротой сходимости* к минимуму и *областью сходимости*. Быстрота сходимости есть характеристика, обратная трудоемкости. Последняя при прочих равных условиях определяется числом шагов к минимуму и трудоемкостью отдельного шага. Трудоемкость иногда оценивают суммарным числом обращений к целевой функции. Более тщательный (и потому более затратный в расчете на шаг) метод определения направления и величины шага уменьшает количество шагов.

Быстро сходящиеся методы, как правило, имеют малую область сходимости и потому требуют хороших начальных приближений, обычно получаемых градиентными методами. Разработан ряд методов решения экстремальных задач, которые сочетают в себе низкую требовательность к выбору начальной точки и высокую скорость сходимости.

В случае оптимизации *с ограничениями* на выбор шага по направлению спуска накладывается дополнительное условие невыхода из допустимой области. Если очередная точка лежит на границе области, то движение происходит по проекции антиградиента на упомянутую границу (методы *возможных направлений*).

Ни универсальных, ни глобально предпочтительных методов оптимизации не существует. В разд. 10.6 [105] приводится набор из семи задач для тестирования алгоритмов оптимизации с указанием, какой метод на какой из задач «спотыкается».

В процессе решения многомерных задач оптимизации полезен режим диалога с ЭВМ.

Значения целевой функции могут быть «зашумлены» случайными ошибками (например, при ее определении в натурном эксперименте или прогоном имитационной модели). В таких случаях при определении направления шага следует учитывать *статистическую значимость* улучшения целевой функции. В связи с высокой стоимостью натурных опытов для экономного определения коэффициентов влияния факторов и их комбинаций создана *теория планирования экспериментов* [38].

11.1.6. Проблема «глобализации»

Большинство методов безусловной минимизации функции многих переменных дают возможность находить *локальный* минимум, и лишь априорная информация о свойствах функции позволяет считать этот минимум глобальным. В общем случае *глобальный* минимум можно найти:

- методом Монте-Карло — в частности, перебором на неравномерной (постепенно сгущаемой) сетке;
- комбинацией случайного выбора начальной точки с одним из алгоритмов спуска;
- построением нижней огибающей целевой функции;
- последовательным отсечением «неперспективных» подмножеств области допустимых решений (метод *ветвей и границ*).

11.2. Методы косвенной оптимизации

Стационарная точка целевой функции удовлетворяет условиям

$$\frac{\partial F}{\partial x_1} = 0, \quad \frac{\partial F}{\partial x_2} = 0, \quad \ldots, \quad \frac{\partial F}{\partial x_n} = 0.$$

Сначала находят все n первых производных F и приравнивают их нулю. Из полученных решений (в общем случае их несколько) отбирают соответствующие положительно определенной матрице Гессе.

Этот метод непрактичен, поскольку:

1) оптимальное решение часто соответствует границе области, а не стационарной точке;
2) решение систем нелинейных уравнений весьма трудоемко;
3) «поиск» гарантирует сходимость к локальному минимуму, тогда как решение нелинейных уравнений может не найти такой точки или вывести в стационарную точку другого типа;
4) при «поиске» гораздо легче учесть ограничения.

11.3. Прямая минимизация

11.3.1. Поиск минимума функции одной переменной

Интерес к одномерным задачам объясняется следующими обстоятельствами:

1. Одномерные задачи часто встречаются на практике.
2. Для одномерного случая разработаны особенно эффективные и надежные методы.
3. Многие идеи, используемые в многомерных задачах, можно проиллюстрировать на одномерном случае.
4. Каждый шаг решения многомерной задачи включает в себя приближенное решение одномерной (расчет длины шага).

Мы ограничимся функциями с единственным минимальным значением, поскольку любое сечение выпуклой поверхности этим свойством обладает. Добавим к этому, что в контексте многомерной оптимизации точное определение минимума по направлению не имеет смысла — достаточно выполнить 1–2 шага такой минимизации.

Метод проб. Пусть точки a и b определяют интервал, на котором достигается единственный минимум целевой функции. Требуется локализовать корень с заданной точностью, вычислив наименьшее число значений функции. Решение этой задачи дается правилом «золотого сечения»: точка деления текущего интервала неопределенности находится из пропорции

$$\frac{1}{x} = \frac{x}{1-x}.$$

Данное условие приводится к квадратному уравнению $x^2 + x - 1 = 0$ с положительным решением

$$x = \frac{-1+\sqrt{1+4}}{2} = (\sqrt{5}-1)/2 \approx 0.618.$$

После этого вычисляется $F(x)$, и найденная точка заменяет границу интервала с наибольшим значением F. На каждом шаге кроме первого F вычисляется однократно.

Квадратичная аппроксимация. Одномерная задача поиска минимума может быть решена после замены интересующей нас функции F

квадратичной согласно разд. 6.5. Первый шаг требует вычисления $F(x)$ в трех точках, каждый последующий — еще в одной.

11.3.2. Метод конфигураций (Хука—Дживса)

Здесь сначала выбираются исходная базовая точка пространства и величины шагов, которые будут использованы при исследовании функции. Затем производится исследование с заданными приращениями по всем координатным осям. Там, где получено наилучшее значение функции, помещают новую временную базовую точку. «Сдвиг схемы» происходит вдоль прямой, соединяющей две базовых точки, с коэффициентом усиления $\alpha > 1$ (т. е. за новую точку). Далее α увеличивается, и процесс повторяется. Это придает ему способность *ускоряться*. При обнаружении отсутствия улучшений в окрестности очередной базовой точки возвращаются в предыдущую и возобновляют исследование с уменьшенным шагом.

Поиск прекращается при уменьшении шага до установленного малого значения и отсутствии значимых улучшений целевой функции. Метод особенно эффективен на гиперповерхностях, содержащих глубокие узкие впадины, — когда градиентные методы неэффективны.

11.3.3. Метод Нелдера—Мида

Этот метод работает с наборами из $(n+1)$-й точки — *симплексами* (в двумерном пространстве — треугольниками, в трехмерном — тетраэдрами). Для всех вершин симплекса вычисляются значения целевой функции, после чего вершины упорядочиваются по убыванию этих значений. Таким образом, худшей оказывается x_1. Далее вычисляется «центр тяжести» всех остальных точек

$$x_0 = \frac{1}{n}\sum_{i=2}^{n+1} x_i.$$

Поиск новой точки ведется вдоль прямой (x_1, x_0). Прежде всего проверяется находящееся вне симплекса «отражение» точки x_1 относительно x_0 — см. рис. 11.7.

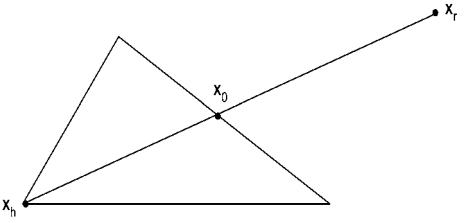

Рис. 11.7. Выбор шага в методе Нелдера—Мида

В зависимости от результата этот поиск либо завершается, либо продолжается внутри или вне комплекса. Выбранная новая точка заменяет собой x_1.

11.3.4. Покоординатный спуск

Покоординатный спуск последовательно отыскивает минимумы по текущей координате. Спуск не обязательно упорядочен по номерам переменных. Можно чаще спускаться по направлениям, которые показали свою перспективность ранее. Возможен и случайный выбор переменной.

При осях изолиний уровня, наклоненнных к осям исходной системы координат, покоординатный спуск совершенно неприемлем — см. рис. 11.8.

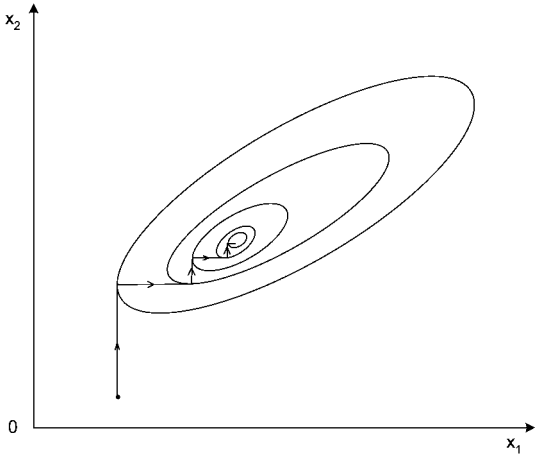

Рис. 11.8. Покоординатный спуск

Приходится эту систему поворачивать. При минимизации квадратичной функции методом покоординатного спуска оси наиболее рациональ-

ной системы координат совпадают с собственными векторами матрицы вторых производных ([105, с. 214]).

11.3.5. Метод Розенброка

Специально на задачи с «овражной» целевой функцией ориентирован *метод Розенброка* — один из немногих, позволяющих минимизировать функцию (11.1.2). При его реализации *в общем случае* предлагается поворачивать систему координат так, чтобы одна из осей была направлена вдоль линии дна оврага, положение которой определяется предварительным исследованием. Остальные оси образуют с ней ортогональную систему координат. После поворота «ведущей» переменной x_i сообщают приращение h_i и вычисляют целевую функцию. Если новое значение меньше предыдущего, то сдвиг считается удачным; движение в том же направлении продолжается с умножением шага на $\alpha > 1$. При неудачном сдвиге шаг делается в противоположном направлении с умножением его величины на $\beta < 1$.

Осуществив сдвиги по всем направлениям, проверяют условие сходимости. При его невыполнении выясняется, было ли сделано хотя бы по одному успешному и безуспешному сдвигу в каждом направлении. Если это не так, то процедура повторяется, начиная с первой переменной. При положительном ответе оси поворачивают так, чтобы исходное направление поиска совпало с наиболее перспективным из рассмотренных направлений.

11.4. Градиентные методы

11.4.1. Общие соображения

Поиск минимума нелинейной функции обычно ведется в направлении убывания функции, т. е. в направлении *антиградиента* или близком к нему. В отличие от обсуждаемого ниже метода Ньютона, градиентные методы при разумном выборе длин шагов гарантированно сходятся к минимуму (возможно, к локальному). Необходимо лишь, чтобы начальное приближение лежало в области его притяжения, т. е. не было расположено на «обратном склоне» невыпуклой поверхности $F(x)$.

Главный их недостаток — медленная (линейная) сходимость. Поэтому есть резон в применении гибридных алгоритмов: поиск начинается каким-либо из градиентных методов и продолжается по Ньютону.

Направление и модуль градиента в общем случае *меняются в каждой точке пространства параметров*. Длину шага для уменьшения риска проскочить минимум можно выбирать по одному из следующих правил:

- достаточно малой и постоянной;
- пропорциональной модулю градиента;
- убывающей функцией номера шага.

Продвижение малыми шагами потребует вычисления направления спуска в очень большом количестве точек и увеличит трудоемкость минимизации. С другой стороны, длинные шаги могут слишком далеко уводить в направлениях, «по большому счету» весьма отличных от оптимального, и порождают *колебательность* поиска. Относительно последнего правила есть мнение (Л. А. Растригин), что коэффициенты уменьшения первоначального шага должны образовывать расходящийся числовой ряд, например гармонический: 1, 1/2, 1/3, 1/4, ...

Для типичных случаев разумным компромиссом представляется продвижение по антиградиенту до достижения минимума целевой функции на этом направлении (метод *скорейшего спуска*) — рис. 11.9.

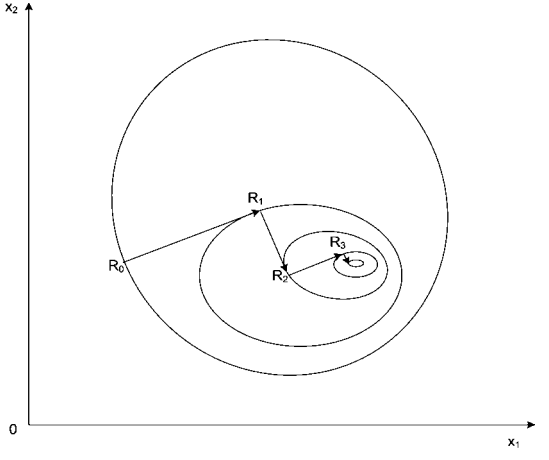

Рис. 11.9. Идея скорейшего спуска

Известно, что градиент (и антиградиент) ортогональны поверхности постоянства значений целевой функции. В двумерном случае то же можно сказать о линии уровня. Если поверхность уровня имеет излом, то направление нормали определяется неоднозначно, и поиск может пойти зигзагами поперек линии излома. При невыпуклой области поиска в этой ситуации до оптимума можно не добраться.

В случае оптимизации *с ограничениями* на выбор шага по направлению спуска накладывается дополнительное условие невыхода из допустимой области. Если очередная точка лежит на границе области, то движение происходит по проекции антиградиента на упомянутую границу.

11.4.2. Алгоритм градиентного спуска

Алгоритм спуска состоит из следующих шагов:

1. Положить $k = 0$. Задать начальное приближение x_0.
2. Вычислить значения функции $F_k = F(x_k)$ и градиента $\nabla_k = \nabla F(x_k)$.
3. Если $||\nabla_k|| < \varepsilon_1$, перейти к шагу 8.
4. Найти такое направление спуска d_k, что $F(x_k + \delta d_k) < F_k$ для малых δ.
5. Найти такое $\tau_k > 0$, что $F(x_k + \tau_k d_k)$ минимальна (линейный поиск по направлению).
6. Если $\tau_k < \varepsilon_2$, перейти к шагу 8.
7. Положить $x_{k+1} = x_k + \tau_k d_k$, $k = k + 1$. Перейти к шагу 2.
8. Конец алгоритма.

Для квадратичной функции n переменных поиск завершается за n шагов. При неквадратичной функции каждое n-е направление должно быть направлением антиградиента.

11.4.3. Скорейший спуск

Если спуск из каждой точки делается по антиградиенту, а длина шага выбирается из условия минимума целевой функции на

выбранном направлении, то спуск называется *скорейшим*[1]. Для минимизации по направлению один шаг метода парабол считается достаточным. Предполагается, что скорейший спуск уменьшает число трудоемких вычислений градиента.

Заметное различие проекций градиента на координатные оси порождает сильно вытянутые изолинии уровней и делает спуск осциллирующим — см. рис. 11.10.

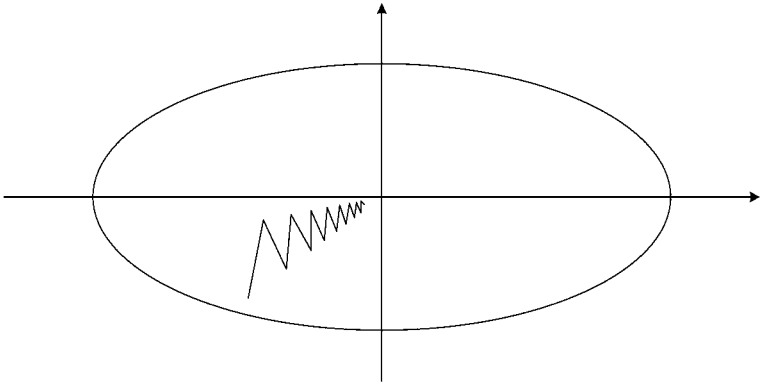

Рис. 11.10. Колебательность скорейшего спуска

В таких случаях надо попытаться модифицировать целевую функцию, уравняв в зоне поиска коэффициенты влияния исследуемых факторов.

11.4.4. Скорейший спуск для систем уравнений

Пусть необходимо решить систему

$$\begin{array}{rcl} f_1(x_1, x_2, \ldots, x_n) & = & 0, \\ f_2(x_1, x_2, \ldots, x_n) & = & 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots & & \\ f_n(x_1, x_2, \ldots, x_n) & = & 0. \end{array} \qquad (11.4.1)$$

Запишем ее в сокращенной форме:

$$f(x) = 0 \qquad (11.4.2)$$

(вектор-функция векторного аргумента). Рассмотрим функцию

$$U(x) = \sum_{i=1}^{n} f_i^2(x) = (f(x), f(x)) \qquad (11.4.3)$$

[1] Зачастую он таковым не является.

(скалярное произведение функции самой на себя). Пусть x — вектор-корень и x_0 — его нулевое приближение. Тогда $U(x) = 0$ — минимальное значение U. Эта точка может быть достигнута скорейшим спуском из x_0. Обозначим через

$$\nabla = \left[\frac{\partial U}{\partial x_1}, \frac{\partial U}{\partial x_2}, \ldots, \frac{\partial U}{\partial x_n}\right]^T \tag{11.4.4}$$

градиент функции $U(x)$ (вектор-столбец). Перейдем в точку

$$x_{k+1} = x_k - \tau_k \nabla_k, \qquad k = 0, 1, \ldots$$

Остается определить множители $\{\tau_k\}$. Введем скалярную функцию

$$\Phi(\tau) = U(x_k - \tau \nabla_k), \tag{11.4.5}$$

которая дает изменение функции U по нормали к поверхности уровня в точке x_k. Множитель $\tau = \tau_k$ надо выбрать из условия обращения $\Phi(\tau)$ в минимум.

Запишем $\Phi(\tau)$ в виде скалярного произведения векторов

$$\Phi(\tau) = (f[x_k - \tau \nabla_k], f[x_k - \tau \nabla_k])$$

и разложим сомножители в ряд по степеням приращения аргумента, ограничившись линейным приближением:

$$f(x - \tau \nabla) \approx f(x) - \frac{df}{dx} \tau \nabla.$$

Тогда

$$\Phi(\tau) \approx (f(x), f(x)) - 2\tau \left(f(x), \frac{df}{dx}\nabla\right) + \tau^2 \left(\frac{df}{dx}\nabla, \frac{df}{dx}\nabla\right),$$

и производная от правой части последнего равенства

$$\Phi'(\tau) = -2 \left(f(x), \frac{df}{dx}\nabla\right) + 2\tau \left(\frac{df}{dx}\nabla, \frac{df}{dx}\nabla\right).$$

Приравняв производную нулю, находим

$$\tau = \left(f(x), \frac{df}{dx}\nabla\right) \Big/ \left(\frac{df}{dx}\nabla, \frac{df}{dx}\nabla\right). \tag{11.4.6}$$

Но

$$\frac{df}{dx} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix} = J(x)$$

есть матрица Якоби вектор-функции $f(x)$. Далее, j-я компонента градиента функции $U(x)$

$$\frac{\partial U}{\partial x_j} = \frac{\partial}{\partial x_j}\Big\{\sum_{i=1}^{n}[f_i(x)]^2\Big\} = 2\sum_{i=1}^{n} f_i(x)\frac{\partial f_i(x)}{\partial x_j}, \qquad j = \overline{1,n}.$$

Поэтому весь вектор градиента

$$\nabla = 2\begin{bmatrix} \sum\limits_{i=1}^{n} f_i(x)\partial f_i(x)/\partial x_1 \\ \sum\limits_{i=1}^{n} f_i(x)\partial f_i(x)/\partial x_2 \\ \cdots\cdots\cdots\cdots\cdots \\ \sum\limits_{i=1}^{n} f_i(x)\partial f_i(x)/\partial x_n \end{bmatrix} = 2J^T(x)f(x)$$

оказывается равным удвоенному произведению транспонированной матрицы Якоби $J^T(x)$ на вектор-функцию f. Окончательно формула (11.4.6) принимает вид

$$\tau_k = \frac{(f_k, 2J_kJ_k^Tf_k)}{(2J_kJ_k^Tf_k, 2J_kJ_k^Tf_k)} = \frac{1}{2}\frac{(f_k, J_kJ_k^Tf_k)}{(J_kJ_k^Tf_k, J_kJ_k^Tf_k)},$$

где индекс k означает вычисление соответствующих векторов и матриц при $x = x_k$.

Пример 11.1. Покажем решение методом наискорейшего спуска системы нелинейных уравнений из примера 9.1. В табл. 11.1 сведены результаты первых 10 шагов метода.

Четыре верных знака во всех компонентах вектора были достигнуты на 14-м шаге.

В табл. 11.2 на основе примеров 9.1 и 9.3 дается сравнение метода скорейшего спуска с рассмотренными ранее. Нельзя не похвалить альтернирующий метод, требующий практически того же числа шагов, что основной, при много меньшей трудоемкости.

Таблица 11.1. Метод скорейшего спуска

Номер шага	Переменная		
	x_1	x_2	x_3
1	0.4257	0.3082	0.6586
2	0.5610	0.2779	0.8141
3	0.5381	0.2686	0.8115
4	0.5409	0.2624	0.8026
5	0.5375	0.2593	0.8038
6	0.5391	0.2571	0.8027
7	0.5382	0.2562	0.8033
8	0.5388	0.2555	0.8030
9	0.5385	0.2552	0.8032
10	0.5386	0.2550	0.8031

Таблица 11.2. Методы и число шагов

Метод решения	Требуемая точность		
	10^{-4}	10^{-5}	10^{-6}
Ньютона	4	5	5
Альтернирующий Ньютона	5	5	5
Модифицированный Ньютона	17	22	27
Скорейший спуск	14	17	20

11.4.5. Методы сопряженных градиентов

Некоторые варианты спуска для подавления обсуждавшейся выше возможной колебательности используют *сопряженные градиенты*. Два направления p и q сопряжены относительно симметричной положительной матрицы G, если произведение

$$p^T G q = 0.$$

Сопряженные направления взаимно ортогональны.

Известно (см. разд. 8.2), что в n-мерном пространстве существует не более n взаимно ортогональных направлений. Поэтому спуск идет циклами из n шагов. Первый шаг делается по антиградиенту, а каждое очередное направление ортогонально всем предыдущим в данном цикле. Если формировать это направление как сумму антиградиента и взвешенного направления предыдущего шага

$$d_k = -\nabla_k + \beta_k d_{k-1},$$

то при надлежащем выборе коэффициента β_k оно окажется сопряженным и со всеми предшествующими направлениями данного цикла.

Методы различаются выбором параметра сопряжения β_k и длины шага. В расчете на n-шаговый цикл методы сопряженных градиентов обладают квадратичной сходимостью [68, с. 222]. В окрестности решения все эти методы по скорости сходимости уступают методу Ньютона.

Доказано (см. [7]), что при минимизации *строго квадратичной* функции в пространстве n измерений поиск заканчивается не более чем за n шагов, т. е. в пределах одного цикла.

Далее для квадрата длины вектора мы будем пользоваться обозначениями вида $d_k^T d_k = d_k^2$.

Метод Флетчера—Ривса. В данном методе коэффициент сопряжения $\beta_k = \nabla_{k+1}^2 / \nabla_k^2$. Считается, что в этом варианте поиск не зависает на возможном изломе линий уровня, а идет вдоль линии, соединяющей точки излома и как правило проходящей через минимум.

Вариант Полака—Рибьера. Здесь параметр сопряжения текущего антиградиента с предыдущим направлением

$$\beta_k = \nabla_k^T (\nabla_k - \nabla_{k-1}) / (\nabla_{k-1}^T \nabla_{k-1}).$$

В литературе этот метод относят к наиболее предпочтительным.

11.5. Методы второго порядка

Напомним, что к этой группе методов относятся использующие производные целевой функции до второй включительно.

11.5.1. Многомерный ряд Тейлора

Будем записывать ряды Тейлора в векторно-матричной форме. Для наглядности начнем с двумерного случая:

$$\begin{aligned} F(x_1 + h_1, x_2 + h_2) &\approx F(x_1, x_2) + h_1 \frac{\partial F}{\partial x_1} + h_2 \frac{\partial F}{\partial x_2} \\ &+ \frac{1}{2}\Big[h_1^2 \frac{\partial^2 F}{\partial x_1^2} + h_1 h_2 \frac{\partial^2 F}{\partial x_1 x_2} + h_2 h_1 \frac{\partial^2 F}{\partial x_2 x_1} + h_2^2 \frac{\partial^2 F}{\partial x_2^2} \Big]. \end{aligned}$$

Обозначив

$$g_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j},$$

можно переписать предыдущее равенство в виде

$$\begin{aligned} F(x+h) &\approx F(x_1,x_2) + [h_1\ h_2] \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \frac{\partial F}{\partial x_2} \end{bmatrix} + \frac{1}{2}[h_1\ h_2] \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \\ &= F(x) + h^T \nabla + \frac{1}{2} h^T G h. \end{aligned}$$

Аналогичный результат имеет место и в случае n измерений.

11.5.2. Метод Ньютона

Приравнивая нулю производную последнего выражения по длине шага, из уравнения

$$\nabla + Gh = 0$$

получаем систему n линейных уравнений (Ньютона) относительно компонент шага

$$Gh = -\nabla.$$

Для функции n переменных

$$\begin{aligned} F(x_0+h) - F(x_0) &= \sum_{i=1}^{n} h_i \frac{\partial F}{\partial x_i} + \frac{1}{2!} \sum_{i=1}^{n} \sum_{j=1}^{n} h_i h_j \frac{\partial^2 F}{\partial x_i \partial x_i} + \dots \\ &= h^T \nabla F(x_0) + \frac{1}{2} h^T G(x_0) h + \dots \end{aligned}$$

При использовании метода Ньютона приращение векторного аргумента $h = -G^{-1}\nabla$. Однако это направление будет направлением спуска лишь при условии, что матрица G^{-1} положительно определена, т. е. $z^T G^{-1} z > 0$ для всех векторов $z \neq 0$. Если это условие на некотором шаге не выполнено, то можно на этом шаге заменить гессиан близкой положительно определенной матрицей $\bar{Q}$ и найти шаг, решив систему $\bar{Q}h = -\nabla$.

11.5.3. Идея квазиньютоновых методов

Если мы находимся в произвольной точке x, то векторные координаты минимума квадратичной функции

$$a = x - G^{-1}\nabla(x).$$

Обозначим $H = G^{-1}$. Матрица H есть оператор, который переводит локальное направление скорейшего спуска $-\nabla$ в правильное направление от x к центру эллипсоида уровней. Таким образом,

$$a = x - H\nabla(x).$$

Если гладкая функция F не является квадратичной, она тем не менее будет приблизительно квадратичной в такой окрестности минимума, где матрица Гессе не вырождается.

Полученные уравнения подсказывают возможный подход к задаче нахождения минимума произвольной функции F. Применим итерационный процесс вида

$$x_{k+1} = x_k - \tau_k H_k \nabla_k,$$

где $\nabla_k = \nabla(x_k)$, а H_k есть k-е приближение к матрице H, обратной для матрицы Гессе G в точке минимума a. Положительное число τ_k выбирается на каждом шаге итераций так, чтобы достигался локальный минимум в направлении $-H_k\nabla_k$ от точки x_k. В этом итерационном процессе одновременно определяют и точку минимума a, и обратную матрицу H.

Идея *квазиньютоновых* методов — строить H_k таким образом, чтобы после n шагов H_n совпала с H, будь F квадратичной функцией. Тогда при использовании точной арифметики n-е приближение x_n совпадет с a (квадратично завершающийся алгоритм).

Квазиньютоновские методы сходятся лишь при достаточно хорошем начальном приближении x_0. Для улучшения такового можно применить одномерный поиск. Пусть имеется квазиньютоново направление $d_k = -H_k\nabla_k$. Примем длину шага $\tau_k = 1$ и проверим неравенство

$$||F(x_k + \tau_k d_k)|| \le ||F(x_k)||.$$

Если оно выполняется, то поиск заканчивается и принимается $x_{k+1} = x_k + \tau_k d_k$ (возможна и попытка дальнейшего увеличения шага). В противном случае шаг дробится и вновь проверяется получение улучшения.

Подберем аппроксимацию гессиана из условия $G_{k+1}(x_{k+1} - x_k) = \nabla_{k+1} - \nabla_k$. Теперь в окрестности решения метод сходится со сверхлинейной скоростью, а в его реализации используются только первые производные.

Квазиньютоновы методы можно разделить на два класса. В одном итерации ведутся по формуле

$$x_{k+1} = x_k - \tau_k \bar{G}_k^{-1} \nabla_k,$$

где матрица $\bar{G}_k$ аппроксимирует матрицу Гессе G_k. В другом

$$x_{k+1} = x_k - \tau_k \bar{H}_k \nabla_k.$$

Здесь матрица $\bar{H}_k$ аппроксимирует *обратную* матрицу Гессе.

11.5.4. Метод Дэвидона—Флетчера—Пауэлла

В данном методе направление поиска на шаге k есть $-H_k \nabla_k$, где H_k — положительно определенная симметричная матрица, которая пересчитывается на каждом шаге и *в пределе* становится равной обратному гессиану.

Существует мнение ([110, с. 178]), что этот метод из всех градиентных методов наиболее эффективен.

Алгоритм метода сводится к следующим шагам:

1. Начать поиск из x_0; начальное значение $H_0 = I$.
2. На k-м шаге выбрать направление $d_k = -H_k \nabla_k$.
3. Проводя одномерный поиск на d_k минимума $F(x_k + \tau_k d_k)$, выбрать τ_k.
4. Вычислить векторное приращение аргумента $v_k = \tau_k d_k$ и положить $x_{k+1} = x_k + v_k$.
5. Вычислить $F(x_{k+1})$ и ∇_{k+1}. Заметим, что $\nabla_{k+1}^T v_k = 0$.
6. Если $|\nabla_{k+1}|$ или $|v_k|$ достаточно малы, перейти к этапу 10.
7. Положить $u_k = \nabla_{k+1} - \nabla_k$.
8. Пересчитать матрицу $H_{k+1} = H_k + A_k + B_k$, где

$$\begin{aligned} A_k &= v_k v_k^T / (v_k^T u_k), \\ B_k &= -H_k u_k u_k^T H_i / (u_i^T H_i u_i). \end{aligned}$$

9. Положить $k := k + 1$ и перейти к шагу 2.
10. Конец алгоритма.

11.6. Оптимизация при наличии ограничений

Задачи с ограничениями обычно значительно более трудоемки. Ограничения могут формулироваться как равенства либо неравенства, причем возможны и смешанные варианты. Считается [99, с. 170], что «ограничения-равенства выражают законы природы типа законов сохранения». Если целевая функция и все ограничения линейны, то множество допустимых решений будет выпуклым многогранником в n-мерном пространстве, а решение — вершиной этого многогранника. На каждом шаге решения переходят к смежной вершине, дающей наибольшее уменьшение целевой функции. Это — задача *линейного программирования*. Трудности при ее решении связаны с очень большими n.

В нелинейном случае оптимальные решения либо соответствуют одному из локальных экстремумов, либо находятся на одной из границ области допустимых решений — см. рис. 11.11.

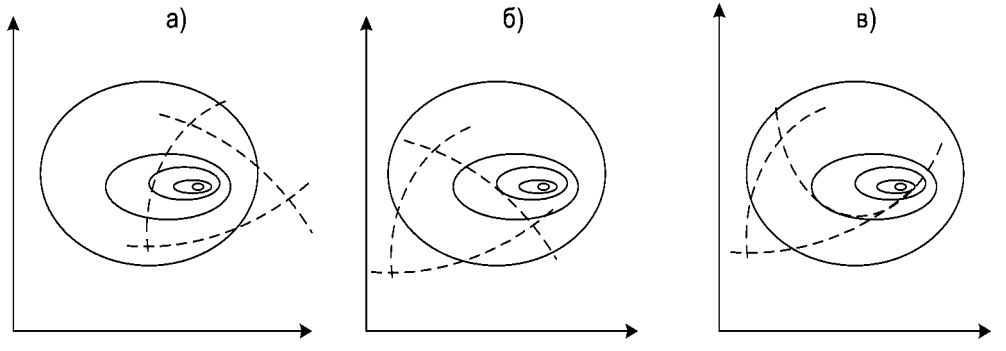

Рис. 11.11. Варианты влияния ограничений

При оптимизации с ограничениями и выходе на границу области направление движения проецируется на эту границу.

11.6.1. Методы прямого поиска

К этой группе мы вновь отнесем методы, не использующие производных от целевой функции. Пусть поставлена задача

$$
\begin{aligned}
& F(x) \to \min \\
& l_j \le x_j \le u_j, \qquad j = \overline{1,n} \qquad \text{— явные ограничения,} \\
& g_i(x) \le b_i, \qquad i = \overline{1,m} \qquad \text{— неявные ограничения.}
\end{aligned}
$$

Если $F(x)$ и все $\{g_i(x)\}$ выпуклы, то задача будет иметь единственное решение. Обязательное условие — границы должны быть безопасны, т. е. заведомо включать минимум. Необходимо выбрать $k = 2n$ точек, удовлетворяющих ограничениям, и вычислить в них целевую функцию. Сначала выбирается базовая точка x_1, остальные — согласно

$$x_j = l_j + r(u_j - l_j),$$

где r — равномерно распределенное на $[0, 1]$ случайное число. Если очередная точка не удовлетворяет неявным ограничениям, то она смещается на половину расстояния до центра тяжести множества уже принятых точек:

$$x_i' = (x_i + x_c)/2,$$

где $x_c = \frac{1}{i-1}\sum_{e=1}^{i-1} x_e$, пока не окажется допустимой. Удобно упорядочить точки в соответствии со значениями функции в них.

Итерационная часть процедуры поиска минимума:

1. Найти точку x_h с наибольшим значением функции и центр x_0 остальных $(k-1)$ точек.

2. «Отразить» x_h относительно x_0 с коэффициентом $\alpha > 1$: $x_r = (1+\alpha)x_0 - \alpha x_h$ (рекомендуется $\alpha \approx 1.3$).

3. Проверить ее допустимость. Если нарушается явная граница, выбрать координату вплотную (на 10^{-6}) к ее внутренней стороне. Если нарушены неявные границы, переместить x_r на половину расстояния между x_r и центром x_0 — пока не будет получена допустимая точка.

4. Если x_r допустима, вычислить $F(x_r)$ и сравнить с наибольшим из $\{F(x_k)\}$. Если новое значение больше, сместить x_r к x_0 на половину расстояния между ними. Перейти к шагу 3. В противном случае заменить x_0 на x_r, переупорядочить точки и значения.

5. Подсчитать дисперсию значений комплекса:

$$\sigma^2 = \left\{ \sum_{e=1}^{k} F^2(x_e) - \left(\frac{1}{k} \sum_{e=1}^{k} F(x_e) \right)^2 \right\} \Big/ k$$

и максимальное расстояние d между точками комплекса.

6. Если хотя бы одна из этих величин превосходит допуск, перейти к п. 1.

7. Конец алгоритма.

Здесь легко усмотреть аналогии с прямыми методами минимизации без ограничений.

При невыпуклой ОДР алгоритмы подобного типа должны дополняться проверкой принадлежности получаемых точек к допустимой области и при необходимости — коррекцией их координат.

Если невыпукла функция, надо применить запуск алгоритма при различных начальных точках.

11.6.2. Градиентные методы

Для простоты и наглядности рассмотрим задачу минимизации функции двух переменных

$$z = f(x, y),$$

где на x и y наложено неявное ограничение вида

$$g(x, y) = 0.$$

Пусть $y = h(x)$. Тогда

$$\frac{dy}{dx} = \frac{d}{dx}h(x) = -\frac{\partial g}{\partial x}\Big/\frac{\partial g}{\partial y},$$

и необходимое условие минимума z

$$\frac{dz}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} = 0,$$

т. е.

$$\frac{\partial f}{\partial x} - \frac{\partial f/\partial y}{\partial g/\partial y} \cdot \frac{\partial g}{\partial x} = 0.$$

Отсюда находим x^* и затем — y^*.

Этот результат можно представить и в другой форме. Если положить

$$\lambda = -\frac{\partial f}{\partial y}\Big/\frac{\partial g}{\partial y}$$

при $x = x^*, y = y^*$, то в точке минимума

$$g(x, y) \;\; = \;\; 0,$$

$$\begin{aligned}\frac{\partial f}{\partial x}+\lambda\frac{\partial g}{\partial x} &= 0,\\ \frac{\partial f}{\partial y}+\lambda\frac{\partial g}{\partial y} &= 0.\end{aligned}$$

Все эти условия непосредственно получаются дифференцированием функции Лагранжа

$$F(x,y,\lambda)=f(x,y)+\lambda g(x,y).$$

В случае m ограничений

$$F(x_1,\ldots,x_n,\lambda_1,\ldots,\lambda_m)=f(x_1,\ldots,x_n)+\sum_{i=1}^{m}\lambda_i g_i(x_1,\ldots,x_n).$$

Теперь условия минимума

$$\begin{aligned}\frac{\partial F}{\partial x_j} &= \frac{\partial F}{\partial x_j}+\sum_{i=1}^{m}\lambda_i\frac{\partial g_i}{\partial x_j}=0, \quad j=\overline{1,n},\\ \frac{\partial F}{\partial \lambda_i} &= g_i(x)=0, \qquad\qquad\qquad i=\overline{1,m}.\end{aligned}$$

Отметим, что для допустимых значений x

$$F(x,\lambda)=f(x)+\sum_{i=1}^{m}\lambda_i g_i(x)=F(x).$$

В точке минимума при наличии ограничений на x^*

$$f(x^*+h)-F(x^*)\geq 0,$$

где $g_i(x^*+h)=0$ для всех i. Таким образом,

$$F(x^*+h)-F(x^*)=\sum_{j=1}^{n}\frac{\partial F}{\partial x_j}h_j+\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}h_i\frac{\partial^2 F}{\partial x_i\partial x_j}h_j+\ldots\geq 0,$$

где производные вычислены в точке x^* при $\lambda=\lambda^*$. Значит, кроме равенства нулю производных, требуется выполнение условия

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}h_i\frac{\partial^2 F}{\partial x_i\partial x_j}h_j\geq 0,$$

т. е. положительной определенности квадратичной формы для значений h, удовлетворяющих ограничениям.

11.6.3. Ограничения в форме неравенств

Рассмотрим общую задачу математического программирования

$$\begin{array}{rcll} F(x) & \to & \min, & \\ g_i(x) & \le & b_i, & i=\overline{1,m}. \end{array}$$

Ограничения в виде неравенств могут быть преобразованы в равенства добавлением к каждому из них *ослабляющих компонент* $\{u_i^2\}$:

$$g_i(x)+u_i^2-b_i=0.$$

Сформируем функцию Лагранжа

$$F(x,\lambda,u)=f(x)+\sum_{i=1}^{m}\lambda_i[g_i(x)+u_i^2-b_i].$$

Необходимые условия минимума суть

$$\begin{aligned} \frac{\partial F}{\partial x_j} &= \frac{\partial f}{\partial x_j}+\sum_{i=1}^{m}\lambda_i\frac{\partial g_i}{\partial x_j}=0, \quad j=\overline{1,n}, \\ \frac{\partial F}{\partial \lambda_i} &= g_i(x)+u_i^2-b_i=0, \quad i=\overline{1,m}, \qquad (11.6.1) \\ \frac{\partial F}{\partial u_i} &= 2\lambda_i u_i=0, \quad i=\overline{1,m}. \qquad (11.6.2) \end{aligned}$$

Умножив последнее уравнение на $u_i/2$, получим $\lambda_i u_i^2=0$, т. е.

$$\lambda_i[b_i-g_i(x)]=0, \quad i=\overline{1,m}. \qquad (11.6.3)$$

Это уравнение означает, что либо $\lambda_i=0$, либо $b_i-g_i(x^*)=0$. Если $\lambda_i\neq 0$, то ограничение является *активным* (строгое равенство). При $g_i(x^*)<0$ ограничение пассивно, им можно пренебречь и принять $\lambda_i=0$. Какие ограничения будут активными, заранее неизвестно.

Предположим, что вышеуказанные условия в точке (x^*,λ^*,u^*) выполнены. Тогда минимум функции $z=F(x^*)$ можно рассматривать как функцию от $\{b_i\}$.

Поскольку $g_k(x)+u_k^2=b_k$, то

$$\frac{\partial g_k}{\partial b_i}=\sum_{j=1}^{k}\frac{\partial g_k}{\partial x_j}\frac{\partial x_j}{\partial b_i}=\begin{cases}0, & \text{если } i\neq k;\\ 1, & \text{если } i=k.\end{cases}$$

Тогда

$$\frac{\partial z}{\partial b_i} + \sum_{k=1}^{m} \lambda_k^* \frac{\partial g_k}{\partial b_i} = \frac{\partial z}{\partial b_i} + \lambda_i^* = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_j} + \sum_{k=1}^{m} \lambda_k^* \frac{\partial g_k}{\partial x_j} \right) \frac{\partial x_j}{\partial b_i}.$$

Но согласно первому из уравнений (11.6.2) выражение в скобках должно быть равно нулю. Таким образом,

$$\frac{\partial z}{\partial b_i} = -\lambda_i^*.$$

С возрастанием b_i область допустимых значений расширяется, что может привести лишь к уменьшению z. Значит,

$$\lambda_i^* \geq 0.$$

Итак, необходимые условия (Куна—Таккера) для минимума $f(x)$ при указанных выше ограничениях суть

$$\begin{array}{rcll} \frac{\partial F}{\partial x_j} + \sum\limits_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial x_j} & = & 0, & j = \overline{1,n}, \\ g_i(x) & \leq & b_i, & i = \overline{1,m}, \\ \lambda_i [g_i(x) - b_i] & = & 0, & i = \overline{1,m}, \\ \lambda_i & \geq & 0, & i = \overline{1,m}. \end{array} \qquad (11.6.4)$$

При решении задачи максимизации знаки $\{\lambda_i\}$ — противоположные.

11.6.4. Метод штрафных функций

При наличии в математической формулировке задачи ограничений вида

$$g_j(x) \geq 0, \qquad j = \overline{1,m},$$

можно ввести составную целевую функцию

$$M(x) = F(x) + \sum_{j=1}^{m} \varphi[g_j(x)].$$

Штрафная функция равна нулю во всех точках пространства, удовлетворяющих ограничениям, и стремится к бесконечности в прочих. Если все условия удовлетворяются, то новая целевая функция имеет прежний минимум. Практически здесь рассматривается серия задач, решения которых последовательно приближаются к условному экстремуму.

Дополнительным достоинством обсуждаемого подхода является то, что реальные ограничения часто являются нежесткими и могут быть ослаблены. В такой ситуации метод штрафных функций позволяет оценить целесообразную степень нарушения стартовых ограничений.

Сформулируем исходную задачу как

$$\begin{aligned} z = f(x) &\rightarrow \min, \\ g_j(x) &> 0, \quad j = \overline{1,m}. \end{aligned}$$

Преобразуем ее в

$$z = f(x) + P(x),$$

где $P(x)$ — штрафная функция. Последнюю удобно задать в виде

$$P(x) = r \sum_{j=1}^{m} \frac{1}{g_j(x)},$$

где $r > 0$. Влияние $P(x)$ создает гребень с крутыми краями вдоль каждой границы области допустимых решений. Чтобы влияние $P(x)$ в точке минимума было малым, r должно быть достаточно мало.

Если $f(x)$ выпукла, а $c_j(x)$ вогнута, то $\varphi(x,r)$ выпукла в выпуклой ОДР и для каждого r имеет единственный минимум.

Важно, чтобы все получаемые точки принадлежали к ОДР, поскольку $\varphi(x,r)$ разрывна на границе области ($\pm\infty$ соответственно внутри и снаружи).

Метод штрафных функций имеет тот недостаток, что не позволяет двигаться вдоль границы ОДР. При больших $\{\lambda_i\}$ его сходимость замедляется.

11.7. Понятие о линейном, целочисленном и динамическом программировании

Линейное программирование охватывает задачи оптимизации, в которых $x \in E_n$, все функции ограничений $g_j(x)$ линейны, а множество Q состоит из векторов с неотрицательными компонентами. Для решения этой задачи предложено множество методов: симплекс-метод, методы одновременного решения прямой и двойственной задач, методы для

ограничений специфической структуры (транспортная задача). Трудоемкость наиболее популярного симплекс-алгоритма в неблагоприятных случаях экспоненциально зависит от числа переменных, а в типичных ситуациях требует от m до $3m$ шагов (m — число ограничений) — см. [65]. Разработаны эффективные методы решения задач большой размерности с блочной структурой ограничений.

Целочисленное программирование имеет дело с принципиально целыми переменными (округление вещественных переменных при малых модулях последних обычно дает решения, далекие от оптимального). Иногда часть переменных или все они принимают значения только из множества $\{0,1\}$ (*булево*, оно же бивалентное программирование). К задачам целочисленного программирования сводятся весьма сложные комбинаторные задачи о коммивояжере, задачи теории расписаний и т. д. Для указанных задач используется упорядоченный перебор (метод ветвей и границ и др).

Метод ветвей и границ основан на построении дерева возможных решений. Для вершин дерева последовательно формируются оценки наилучших значений целевой функции, после чего область дальнейшего поиска сужается. Предпочтительность стратегий обхода дерева («вширь» или «в глубину») зависит от специфики задачи.

Динамическое программирование описывает многоэтапные процессы принятия решений с аддитивной (по этапам) целевой функцией. Оно конструктивно использует *принцип оптимальности* Р. Беллмана: для любых этапа и состояния независимо от предыстории поведение должно быть оптимально на всю совокупность оставшихся этапов.

Метод состоит из обратного и прямого хода. В процессе обратного хода для этапов от последнего до первого и для всех возможных исходных состояний строятся таблицы оптимального поведения и дохода, получаемого за все оставшиеся (включая последний) этапы. Прямой ход начинается с исходного состояния и реализуется с применением упомянутых таблиц.

В процессе реализации динамического программирования широко используются методы аппроксимации зависимостей и одномерной минимизации. С другой стороны, идеи и методы динамического программирования часто комбинируют с классическими моделями оптимизации.

Метод динамического программирования весьма поучителен с точки зрения «философии жизни», поскольку наглядно демонстрирует невыгодность стремления получить максимум «здесь и сейчас» и необходимость задумываться о долгосрочных последствиях принятых решений.

Глава 12. Численное дифференцирование и интегрирование

12.1. Численное дифференцирование

Не каждую функцию можно дифференцировать аналитически. Она может быть задана слишком громоздкой формулой или неявно (посредством численного алгоритма). В таких случаях приходится опираться на таблицу ее значений.

Идея *численного* дифференцирования непосредственно вытекает из определения производной: при ее вычислении разность двух значений функции делится на величину Δx. Теоретически при уменьшении шага результат должен приближаться к истинному значению производной. Однако при этом будет уменьшаться модуль разности истинных значений функции и соответственно возрастать влияние ошибок в их определении. В частности, это относится к влиянию ограниченности разрядной сетки. Если дифференцировать зашумленный сигнал, то погрешность резко увеличится.

Поскольку производные от многочленов вычисляются легко (в том числе в символьном виде), естественно пытаться дифференцировать построенный по упомянутой таблице интерполяционный многочлен. Однако даже в случае, когда последний хорошо аппроксимирует табличную функцию, его производные могут не иметь ничего общего с производными аппроксимируемой функции — сравните угловые коэффициенты касательных к функции и полиному в узлах интерполяции на рис. 12.1.

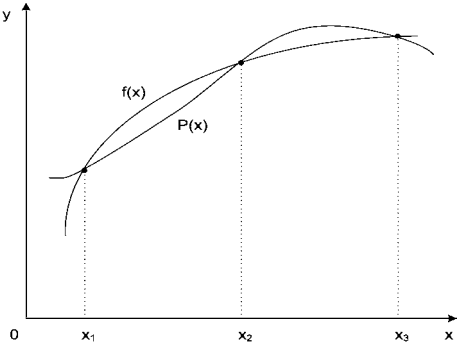

Рис. 12.1. Погрешности производной при полиномиальной аппроксимации функции

Это означает, что при решении задачи численного дифференцирования с применением интерполяционных формул необходимы грамотный выбор метода и его параметров и получение надежных оценок сопутствующей погрешности.

12.1.1. Общий подход

Пусть функция $f(x)$ определена в промежутке $[a,b]$ и известна лишь в точках $x_0, x_1, \ldots, x_n$ этого промежутка. Предположим, что для каких-то целей требуется вычислить производную $f'(x)$ в точках $x_0, x_1, \ldots, x_n$. Естественно образовать интерполяционный многочлен для функции $f(x)$, приняв либо все названные точки, либо часть их за узлы интерполяции, а производную интерполяционного многочлена считать приближенным значением $f'(x)$.

Применим этот прием для случая равноотстоящих (с шагом h) узлов интерполяции, а производную будем искать лишь в узлах интерполяции.

12.1.2. Расчетные формулы и оценка погрешности

Из приведенной в разд. 10.4 формулы для оценки погрешности интерполяции следует формула для оценки погрешности приближенного равенства $f'(x_j) \approx L'_f(x_j)$:

$$|f'(x_j) - L'_f(x_j)| \le \frac{\max|f^{(n+1)}(x)|}{(n+1)\,!}|\omega'(x_j)|. \qquad (12.1.1)$$

Рассмотрим частные случаи этой формулы.

1) $n = 1$. Здесь даны узлы x_0 и $x_1 = x_0 + h$. Тогда

$$L_f(x) = f(x_0) + \frac{x - x_0}{h}[f(x_1) - f(x_0)],$$

$$L'_f(x) = \frac{f(x_1) - f(x_0)}{h} \tag{12.1.2}$$

и погрешность дифференцирования

$$\left| f'(x_0) - \frac{f(x_1) - f(x_0)}{h} \right| \leq \frac{\max |f''(x)|}{2!} h.$$

Формула (12.1.2) имеет простой геометрический смысл: в качестве приближенного значения неизвестного углового коэффициента касательной к графику $y = f(x)$ в точке с абсциссой x_0 предлагается принять известный угловой коэффициент хорды графика. Это же значение можно считать приближенным значением для $f'(x_1)$.

2) При $n = 2$

$$L_f(x) = f(x_0) + \frac{x - x_0}{h}[f(x_1) - f(x_0)] + \frac{(x - x_0)(x - x_1)}{2h^2}[f(x_2) - 2f(x_1) + f(x_0)]$$

и, следовательно,

$$L'_f(x) = \frac{f(x_1) - f(x_0)}{h} + \frac{2x - (x_1 + x_0)}{2h^2}[f(x_2) - 2f(x_1) + f(x_0)]. \tag{12.1.3}$$

Здесь возможны два случая:

а) $x = x_1$ (производная определяется в средней точке). Тогда

$$2x - (x_1 + x_0) = x_1 - x_0 = h$$

и

$$L'_f(x) = \frac{1}{2h}\{2[f(x_1) - f(x_0)] + [f(x_2) - 2f(x_1) + f(x_0)]\} = \frac{f(x_2) - f(x_0)}{2h}. \tag{12.1.4}$$

Формула (12.1.1) в сочетании с (12.1.4) дает

$$\left| f'(x_1) - \frac{f(x_2) - f(x_0)}{2h} \right| \leq \frac{\max |f'''(x)|}{3!} h^2. \tag{12.1.5}$$

На этот раз (рис. 12.2) за приближенное значение неизвестного углового коэффициента касательной к графику $y = f(x)$ в точке с абсциссой x_1 предлагается принять известный угловой коэффициент хорды, соединяющей точки графика с абсциссами $x_1 - h$ и $x_1 + h$.

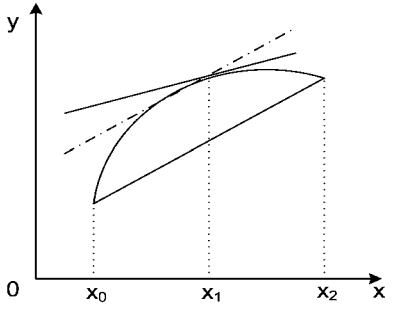

Рис. 12.2. Геометрия центральной формулы численного дифференцирования

б) Положим в равенстве (12.1.3) $x = x_0$ (производная определяется в крайней точке). Тогда $2x - (x_1 + x_0) = x_0 - x_1 = -h$, и (12.1.3) дает

$$\begin{aligned} L'_f(x_0) &= \frac{2[f(x_1) - f(x_0)] - [f(x_2) - 2f(x_1) + f(x_0)]}{2h} \\ &= \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h}. \end{aligned} \qquad (12.1.6)$$

Формула (12.1.1) в сочетании с (12.1.6) приводит к оценке

$$\left| f'(x_0) - \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h} \right| \le \frac{\max |f'''(x)|}{3!} 2h^2. \qquad (12.1.7)$$

Заметим, что оценка ошибки возросла вдвое — см. формулу (12.1.5). Аналогично можно получить формулы для вычисления производных высших порядков.

В общем случае при необходимости приближенного вычисления производной в некотором узле x_j желательно составлять интерполяционный многочлен по таким узлам, чтобы точка x_j была средней среди остальных узлов интерполяции. В этом смысле формула типа (12.1.4) рекомендуема, а типа (12.1.6) — не рекомендуема, и ею следует пользоваться лишь при отсутствии узлов слева от x_j.

Пример 12.1. Дана табл. 12.1 значений функции $y = \operatorname{tg} x$.

Таблица 12.1. К анализу методов численного дифференцирования

x	$\operatorname{tg} x$	x	$\operatorname{tg} x$
1.514	17.588	1.516	18.231
1.515	17.904	1.517	18.511

Требуется вычислить производную от $\operatorname{tg} x$ при $x = 1.515$. Рассмотрим несколько вариантов:

$$
\begin{aligned}
f'(1.515) &\approx \frac{f(1.516) - f(1.515)}{h} = \frac{18.231 - 17.904}{0.001} = 327;\\
f'(1.515) &\approx \frac{f(1.515) - f(1.514)}{h} = \frac{17.904 - 17.588}{0.001} = 316;\\
f'(1.515) &\approx \frac{f(1.516) - f(1.514)}{2h} = \frac{18.231 - 17.588}{2 \cdot 0.001} = 321.5;\\
f'(1.515) &\approx \frac{-3f(1.515) + 4f(1.516) - f(1.517)}{2h} = 320.5.
\end{aligned}
$$

Поскольку $(\operatorname{tg} x)' = 1/\cos^2 x$, то $\operatorname{tg}' 1.515 = 1/0.0577^2 = 321.5$.

Наилучший результат получился при третьем варианте — формула (12.1.4). Несколько худший итог дала формула (12.1.6) — четвертый вариант, и значительно худший — формула (12.1.2), реализующая первый и второй варианты.

12.1.3. О построении таблиц для численного дифференцирования

При численном дифференцировании таблично заданной функции $y = f(x)$ возникают погрешности:

- округления, порождаемые неточным заданием значений $\{y_i\}$;
- усечения, вызываемые заменой функции $y = f(x)$ интерполяционным многочленом $P_n(x)$ конечной степени n.

Выше были приведены теоретические оценки для погрешности усечения, из которых следует уменьшение погрешности усечения при $h \to 0$. Однако значения функции $y(x)$ всегда определяются с некоторой погрешностью вследствие округлений при ограниченной разрядной сетке, применения приближенных методов вычисления $y(x)$ и т. д. При очень малом шаге полезная информация (разность теоретических значений функций) окажется перекрытой случайными ошибками. Поэтому шаг не должен быть слишком мал, и существует некоторое оптимальное (для каждой формулы численного дифференцирования и погрешностей округления — свое) значение шага.

Найдем оптимальный шаг для простейшей центральной формулы типа (12.1.4). Из нее следует, что при абсолютной погрешности ε вычисления функции погрешность округления не превосходит $2\varepsilon/2h = \varepsilon/h$, а

погрешность усечения не больше $h^2M_3/6$, где M_3 — максимум модуля третьей производной на используемом участке таблицы. Суммарная погрешность $h^2M^3/6+\varepsilon/h$ достигает минимума при $2hM_3/6-\varepsilon/h^2=0$, или при

$$h=\sqrt[3]{3\varepsilon/M_3}.$$

Полученную формулу можно переписать в виде

$$\frac{h^2}{6}M_3=\frac{1}{2}\frac{\varepsilon}{h},$$

допускающем наглядное истолкование: погрешность усечения должна составлять половину погрешности округления.

Пример 12.2. Вычислить производную от e^x в точке $x=1.15$ с помощью четырехзначной таблицы показательной функции, варьируя h от 0.1 до 0.01 и применяя формулу (12.1.4). Точное значение производной равно $e^{1.15}=3.1582$. Результаты расчета представлены в табл. 12.2. Погрешность дифференцирования Δ указана в десятитысячных долях единицы.

Таблица 12.2. Влияние шага на точность численного дифференцирования

Шаг	$L'_f(1.15)$	Δ	Шаг	$L'_f(1.15)$	Δ
0.10	3.1630	-48	0.05	3.1590	-8
0.09	3.1622	-40	0.04	3.1588	-6
0.08	3.1613	-31	0.03	3.1583	-1
0.07	3.1607	-25	0.02	3.1575	7
0.06	3.1600	-18	0.01	3.1550	32

Погрешность вычисления функции $\varepsilon=0.5\cdot10^{-4}$, а $M_3=e^{1.25}=3.4903$. Теоретическое значение оптимального шага

$$h=\sqrt[3]{3\cdot0.5\cdot10^{-4}/3.4903}=\sqrt[3]{43\cdot10^{-6}}=0.035,$$

что примерно соответствует фактическому расчету.

В [110, с. 218–219] приведены таблицы безразностных формул аппроксимации по начальным и центральным разностям для производных до третьего порядка включительно и оценками погрешности аппроксимации (число точек берется от двух до пяти, но по крайней мере на одну больше, чем порядок производной).

12.2. Расчет моментов распределения через преобразование Лапласа

Ограниченная разрядность оказывает заметный эффект на процессы численного дифференцирования, в особенности многократного. Здесь разности высшего порядка могут оказаться соизмеримыми с погрешностью представления операндов. Для получения приемлемых результатов требуется переход к построению интерполирующих многочленов по *центральным* формулам и/или счет в увеличенной разрядной сетке.

Многие расчеты систем обслуживания (в особенности для приоритетных режимов) удается довести только до получаемых алгоритмически преобразований Лапласа—Стилтьеса (ПЛС) от распределений времени пребывания заявки $\varphi(s)$, вследствие чего расчет моментов $\{f_k\}$ упомянутых распределений приходится выполнять численно. Такой расчет основывается на построении интерполяционного многочлена для ПЛС и последующем численном дифференцировании этого многочлена в нуле:

$$f_k = (-1)^k \varphi^{(k)}(s)|_{s=0}, \qquad k = 1, 2, \ldots$$

Программная реализация определяется видом используемой в этой технологии интерполяционной формулы. Проведем соответствующие рассуждения для формулы Ньютона.

Интерполяционная формула *Ньютона* имеет вид

$$y(x) = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \frac{q(q-1)(q-2)}{3!}\Delta^3 y_0 + \ldots, \quad (12.2.1)$$

где

$$q = x/h. \quad (12.2.2)$$

В процедуре `DIFNEWT` объявлены массивы:

- `Y(0:KK,0:KK)` конечных разностей дифференцируемой табличной функции (первый индекс — номер точки, второй — порядок разности);
- `A(1:KK,0:KK)` коэффициентов многочленов от Q при конечных разностях в формуле (12.2.1) (первый коэффициент — порядок разности, второй — степень Q).

Массив A предварительно обнуляется. В нулевой столбец Y засылаются разности нулевого порядка, т. е. табличные значения функции. Затем в цикле по K последовательно формируются разности до

K-го порядка включительно (каждая разность — для тех строк, в которых она определена). В процессе этого расчета вычисляется максимальный модуль разности каждого порядка. Он сравнивается с аналогичной величиной для предшествующего порядка. При обнаружении возрастания указанной характеристики, что указывает на проявление эффекта ограниченности разрядной сетки, количество используемых в расчете разностей ограничивается.

Вычисление коэффициентов многочленов основано на представлении многочлена при K-й разности в форме

$$P_k(q) = a_{k0} + a_{k1}q + a_{k2}q^2 + \cdots + a_{kk}q^k,$$

начальном условии $P_1 = Q$ и очевидном из (12.2.1) законе пересчета

$$P_k = P_{k-1}\frac{q-k+1}{k} = P_{k-1}\Big[\frac{q}{k} - (1-1/k)\Big], \qquad k = \overline{2,K}.$$

В последующем цикле по K идет собственно вычисление производных. Здесь сначала почленно дифференцируются все еще не обратившиеся в константу многочлены, а затем суммируются произведения их свободных членов на соответствующую разность (все остальные слагаемые при $Q = 0$ обращаются в нуль). Поскольку производные отыскиваются по $X = QH$, для получения каждой очередной производной делитель B умножается на H.

```
subroutine difnewt(kk,f,h,der)
!************************************************
!* Многократное числ. дифференцирование в нуле   *
!*      таблично заданной функции                *
!* Интерполяция - многочленом Ньютона            *
!*      Разработчик Ю. И. Рыжиков                *
!*                  (c)                          *
!*        Версия 26.02.1995                      *
!************************************************

   integer          kk   !!  расчетная степень многочлена
   double precision f    !!  значения функции
   double precision h    !!  шаг изменения аргумента функции
   double precision der  !!  последоват. производные
   dimension        f(0:kk),der(kk)

   double precision b,p,s,s1
```

```
   integer           i,j,k,kmax
   double precision, allocatable ::  a(:,:), y(:,:)

   allocate   (a(kk,0:kk),y(0:kk,0:kk))
   s=abs(f(0))
   do k=0,kk
      y(k,0)=f(k); p=abs(f(k))
      if (p.gt.s) s=p
   end do
   kmax=kk
   do j=1,kk
      y(0,j)=y(1,j-1)-y(0,j-1)
      s1=abs(y(0,j))
      do k=1,kk-j
         y(k,j)=y(k+1,j-1)-y(k,j-1)
         p=abs(y(k,j))
         if (p.gt.s1) s1=p
      end do
      if (s1.lt.s) then
         s=s1
      else
         kmax=j-1; goto 1
      end if
   end do
 1 do i=1,kk
      do j=0,kk; a(i,j)=0; end do
   end do
   a(1,1)=1d0
   do k=2,kmax
      do j=1,k-1
         a(k,j)=a(k-1,j-1)/k-(1d0-1d0/k)*a(k-1,j)
      end do
      a(k,k)=a(k-1,k-1)/k
   end do
   b=1d0
   do k=1,kmax
      b=b*h; s=0
      do i=k,kmax
         do j=0,i-k;   a(i,j)=(j+1d0)*a(i,j+1); end do
```

```
        a(i,i-k+1)=0; s=s+a(i,0)*y(0,i)
      end do
      der(k)=s/b
   end do
   deallocate (a,y)
   end subroutine difnewt
```

Процедуру удобно тестировать на расчете моментов показательной плотности распределения $b(t) = \mu e^{-\mu t}$, равных $k!/\mu^k$, $k = 1, 2, \ldots$, численным дифференцированием ее преобразования Лапласа $\beta(s) = \mu/(\mu + s)$.

Таблица 12.3. Тестирование процедур численного дифференцирования

№	Точное значение производной	погрешности		
		difnewt	difndif	difstir
1	-.333333333333333e+01	-.760e-12	-.285e-11	-.225e-12
2	.222222222222222e+02	-.431e-08	-.184e-07	-.620e-09
3	-.222222222222222e+03	-.754e-05	-.418e-04	.114e-05
4	.296296296296296e+04	-.517e-02	-.442e-01	.229e-02
5	-.493827160493827e+05	-.122e+01	-.229e+02	-.130e+01
6	.987654320987655e+06	-.143e+02	-.472e+04	-.228e+04

Здесь DIFNEWT работала по формуле Ньютона, DIFNDIF — по ее безразностному варианту, а DIFSTIR — по формуле Стирлинга с центральными разностями.

Таблица вычислялась в 9 точках. Как и ожидалось, уменьшение шага повышало точность младших производных и ухудшало — старших. В ходе тестирования было определено рациональное значение шага построения таблицы ПЛС $h \approx 10^{-3}\mu$, при котором распределение погрешностей четырех младших моментов «наиболее гармонично».

Заметим, что в таблице приведены абсолютные погрешности дифференцирования. Естественный переход к относительным погрешностям с учетом бóльших допусков на высшие моменты убеждает в приемлемости всех приведенных результатов.

Обращаем внимание читателя на форму представления результатов в этой таблице: эталон и *отклонения* от него. Она заметно повышает наглядность сопоставления сравниваемых методов и облегчает анализ «тонких эффектов».

12.3. Общие сведения об интерполяционных квадратурных формулах

В этой главе нас будет интересовать вычисление интеграла

$$I = \int_a^b f(x)\,dx.$$

Аналог термина «интегрирование» — квадратура — как раз и означает вычисление площади.

Если подынтегральная функция задана аналитически, определенный интеграл часто удается вычислить с помощью первообразной по формуле Ньютона—Лейбница

$$\int_a^b f(x)\,dx = F(x)\Big|_a^b = F(b) - F(a).$$

Однако этот подход не работает, если:

- первообразную нельзя выразить в элементарных функциях — к примеру, для $\sin x/x$, $1/\ln x$, e^{-x^2} это невозможно;
- значения $f(x)$ заданы только в отдельных точках.

Методы численного интегрирования основаны на замене $f(x)$ более простыми выражениями — например, многочленами. Вместо того чтобы вычислить $I = \int_a^b f(x)\,dx$ непосредственно, сперва вычисляют значения функции $f(x)$ в заданных точках $x_i \in [a,b]$. Если $p(x)$ — полином, проходящий через точки $\{x_i, f(x_i)\}$, то можно считать

$$\int_a^b p(x)\,dx \approx \int_a^b f(x)\,dx.$$

Поскольку интегрирование полинома не представляет труда, этот подход легко реализуем численно.

Численное интегрирование — процесс, гораздо более устойчивый к наложенному случайному шуму, чем численное дифференцирование.

Квадратурное правило записывается в виде

$$I = \int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_n.$$

Здесь $\{x_i\}$ — узлы, $\{w_i\}$ — веса, R_n — остаток (он же погрешность) квадратурной формулы. Говорят, что n-точечное квадратурное правило имеет алгебраическую степень точности d, если его остаток R_n равен нулю для любого полинома степени d, но отличен от нуля для некоторого полинома степени $d+1$. Степень многочлена, для которого квадратура точна, определяется числом свободных параметров ее.

От большинства квадратурных программ требуется не только вычисление интеграла, но и оценка погрешности результата. Анализ последней важен не только сам по себе; он позволяет повысить эффективность процесса интегрирования — например, сосредоточить большую часть усилий на участке быстрого изменения функции.

Методы вычисления квадратур различаются способом выбора узлов, их количеством и построением интерполяционных многочленов.

12.3.1. Соотношение между узлами и весами

Пусть функция $f(x)$ определена на отрезке $[\alpha, \beta]$ и известна лишь в точках $x_0, x_1, \ldots, x_k$. Тогда для вычисления $\int_\alpha^\beta f(x)\,dx$ естественно построить интерполяционный многочлен $L_f(x)$ по узлам $x_0, x_1, \ldots, x_k$ и $\int_\alpha^\beta L_f(x)\,dx$ принять за приближенное значение искомого интеграла.

Интерполяционный многочлен Лагранжа будет иметь вид

$$L_f(x) = y_0 \frac{\omega_0(x)}{\omega_0(x_0)} + y_1 \frac{\omega_1(x)}{\omega_1(x_1)} + \ldots + y_k \frac{\omega_k(x)}{\omega_k(x_k)}.$$

Здесь

$$\begin{aligned} \omega(x) &= (x - x_0)(x - x_1)\ldots(x - x_k), \\ \omega_j(x) &= \frac{\omega(x)}{x - x_j}, \qquad j = \overline{0, k}. \end{aligned}$$

Заметим, что $\omega_j(x_j) = \omega'(x_j)$. Теперь можно записать равенство

$$\int_\alpha^\beta L_f(x)\,dx = y_0 \Big[\frac{1}{\omega'(x_0)} \int_\alpha^\beta \omega_0(x)\,dx\Big] + \ldots + y_k \Big[\frac{1}{\omega'(x_k)} \int_\alpha^\beta \omega_k(x)\,dx\Big].$$

Введем числа

$$B_j = \frac{1}{\omega'(x_j)} \int_\alpha^\beta \omega_j(x)\,dx, \qquad j = \overline{0,k}, \tag{12.3.1}$$

определяемые только набором узлов. Тогда получим

$$\int_\alpha^\beta f(x)\,dx \approx \int_\alpha^\beta L_f(x)\,dx = \sum_{j=0}^{k} B_j f(x_j). \tag{12.3.2}$$

Подчеркнем, что числа $\{B_j\}$ *не зависят* от функции f.

Формулы вида (12.3.2) называются формулами *интерполяционных квадратур*. Если пределы интегрирования α и β являются узлами интерполирования, то формула вида (12.3.2) относится к «замкнутому» типу, в противном случае — к открытому.

Для вычисления коэффициентов можно воспользоваться интегралами (12.3.1). Однако более удобен другой путь. Заметим, что если $f(x)$ есть многочлен степени не выше k, то $L_f(x) \equiv f(x)$. В этом случае равенство (12.3.2) становится точным. Полагая поочередно $f(x)$ равной $1, x, \ldots, x^k$, получим систему из $(k+1)$ линейных уравнений

$$\begin{array}{lllll}
\int\limits_\alpha^\beta 1\cdot dx & = B_0 & +B_1 & +\ldots\ +B_k & = \beta-\alpha;\\
\int\limits_\alpha^\beta x\,dx & = B_0x_0 & +B_1x_1 & +\ldots\ +B_kx_k & = (\beta^2-\alpha^2)/2;\\
\ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots\\
\int\limits_\alpha^\beta x^k\,dx & = B_0x_0^k & +B_1x_1^k & +\ldots\ +B_kx_k^k & = (\beta^{k+1}-\alpha^{k+1})/(k+1)
\end{array} \tag{12.3.3}$$

для отыскания неизвестных коэффициентов $\{B_j\}$. Определитель этой системы (так называемый определитель Вандермонда) при различных числах $x_0, x_1, \ldots, x_k$ отличен от нуля, и система (12.3.3) имеет единственное решение — набор $\{B_j\}$. Эти числа будем называть *весами* квадратурной формулы.

12.3.2. Преобразование промежутка интегрирования

Линейная функция $x = \frac{\beta-\alpha}{2}t + \frac{\beta+\alpha}{2}$ такова, что при $t=-1$ имеем $x=\alpha$, а при $t=1$, соответственно, $x=\beta$. Следовательно, указанное

преобразование переводит отрезок [-1,1] в отрезок $[\alpha, \beta]$. Делая замену переменной, получаем

$$\int_{\alpha}^{\beta} f(x)\,dx = \frac{\beta-\alpha}{2}\int_{-1}^{1} f\Big(\frac{\beta-\alpha}{2}t + \frac{\beta+\alpha}{2}\Big)\,dt. \qquad (12.3.4)$$

Поэтому, если для всех многочленов $f(t)$ степени не выше l верна формула

$$\int_{-1}^{1} f(t)\,dt = A_0 f(t_0) + A_1 f(t_1) + \ldots + A_k f(t_k),$$

то для вычисления $\int_{\alpha}^{\beta} f(x)\,dx$ следует пользоваться весами и узлами

$$\begin{aligned} B_j &= \frac{\beta-\alpha}{2}A_j, \\ x_j &= \frac{\beta-\alpha}{2}t_j + \frac{\beta+\alpha}{2} \end{aligned} \qquad (12.3.5)$$

соответственно. При этом новая квадратурная формула также будет точна для всех многочленов степени не выше l. В дальнейшем, имея в виду возможность такого перехода, мы будем иметь дело преимущественно со стандартным отрезком [-1,1]. Весовые коэффициенты для интегрирования на нем будем обозначать через $\{A_j\}$, а на нестандартном отрезке — через $\{B_j\}$.

12.4. Квадратурные формулы Котеса

Разобьем отрезок $[\alpha, \beta]$ на равные части с шагом h. Получаемые при этом формулы типа (12.3.2) называются формулами $Котеса$. Выведем их частные варианты для некоторых значений k и интервала $[0, kh]$.

а) $\underline{k=1.}$ В этом случае узлы $x_0 = 0$, $x_1 = h$, так что

$$\begin{aligned} B_0 \quad &+B_1 \quad &= \quad h; \\ B_0 \cdot 0 \quad &+B_1 h \quad &= \quad h^2/2. \end{aligned}$$

Таким образом, $B_0 = B_1 = h/2$, а потому

$$\int_0^h f(x)\,dx = \frac{h}{2}[f(0) + f(h)].$$

Для произвольного промежутка

$$\int_\alpha^\beta f(x)\,dx = \frac{\beta-\alpha}{2}[f(\alpha) + f(\beta)]. \tag{12.4.1}$$

Это — простейшая формула трапеций.

б) $\underline{k=2}$. Теперь узлы $x_0 = 0$, $x_1 = h$, $x_2 = 2h$, и система (12.3.3) принимает вид

$$\begin{array}{llll} B_0 & +B_1 & +B_2 & = 2h; \\ B_0\cdot 0 & +B_1 h & +B_2(2h) & = (2h)^2/2 \quad = 2h^2; \\ B_0\cdot 0^2 & +B_1 h^2 & +B_2(2h)^2 & = (2h)^3/3 \quad = 8h^3/3. \end{array}$$

Умножая второе уравнение на h и вычитая из третьего, получаем $2h^2 B_2 = \frac{2}{3}h^3$, или $b_2 = h/3$; из второго уравнения находим $B_1 = \frac{1}{h}(2h^2 - \frac{2}{3}h^2) = \frac{4}{3}h$. Теперь из первого получаем $B_0 = h/3$. Итак,

$$\int_0^{2h} f(x)\,dx = \frac{h}{3}\,[f(0) + 4f(h) + f(2h)].$$

Вообще для интегрирования на произвольном участке

$$\int_\alpha^\beta f(x)\,dx = \frac{\beta-\alpha}{6}\Big[f(\alpha) + 4f\Big(\frac{\beta+\alpha}{2}\Big) + f(\beta)\Big]. \tag{12.4.2}$$

Это — формула *Симпсона*.

Аналогичным методом можно получить формулы более высокого порядка для интегрирования на отрезке $[0, kh]$. Приведем только коэффициенты этих формул:

$$\begin{array}{lll} k=3 & : & \dfrac{3}{8}h \quad (1,3,3,1) \\ k=4 & : & \dfrac{4}{90}h \quad (7,32,12,32,7) \\ k=5 & : & \dfrac{5}{288}h \quad (19,75,50,50,75,19) \\ k=6 & : & \dfrac{6}{840}h \quad (41,216,27,272,27,216,41). \end{array} \tag{12.4.3}$$

Для произвольного интервала $[\alpha, \beta]$ ординаты в этих формулах берутся через шаг $h = (\beta - \alpha)/k$, начиная с α. Если $f(x)$ определена во всех точках интервала $[\alpha, \beta]$ и имеет непрерывную производную порядка $k + 2$, то можно указать оценку ошибки от замены $\int_\alpha^\beta f(x)\,dx$ на правую часть соответствующей формулы Котеса (табл. 12.4). В ней M_n обозначает максимум $|f^{(n)}(x)|$ на отрезке $[\alpha, \beta]$, а $L = \beta - \alpha$.

Таблица 12.4. Предельные ошибки квадратурных формул Котеса

k	Ошибка	k	Ошибка
1	$8.333 \cdot 10^{-2} L^3 M_2$	4	$5.167 \cdot 10^{-7} L^3 M_6$
2	$3.472 \cdot 10^{-4} L^3 M_4$	5	$2.910 \cdot 10^{-7} L^3 M_6$
3	$1.543 \cdot 10^{-4} L^3 M_4$	6	$6.379 \cdot 10^{-10} L^3 M_8$

Эти и приводимые ниже оценки относятся к «погрешности усечения». К ним должна быть добавлена погрешность взвешенной суммы ординат, равная $\Delta \sum_i B_i = \Delta(\beta - \alpha)$ [см.(12.3.3)], где Δ — погрешность вычисления ординат. Из табл. 12.4 следует, что формула Котеса для $k = 2$ (формула Симпсона) точна также для многочлена третьей степени, поскольку $f^{(4)}(x)$ для полинома третьей степени равна нулю. Аналогично формула для $k = 4$ верна для многочленов пятой степени. Таким образом, в смысле порядка точности квадратурные формулы с нечетным числом ординат являются более выигрышными.

Покажем, как на Фортране 90 программируется вычисление интегралов от *произвольных* функций. Здесь приходится применять функции в роли параметров процедур. Пусть необходимо подсчитать

$$\int_0^b e^{-y} \sin y\,dy - \int_0^c \cos z\,dz$$

с помощью функции, вычисляющей $\int_0^a f(x)\,dx$ по формуле трапеций с разбивкой интервала на 10 частей. Эту задачу решает программа

```
program numint
   real b,c,z,z1,z2
   intrinsic cos
   interface
      real function esin(y)
         real y
      end function esin
```

```
      end interface
      b=2.0; c=1.6
      z1=sintg(b,esin); z2=sintg(c,cos)
      z=z1-z2
      print *, ' z = ',z ! -5.347379e-01
   contains
      real function sintg(a,f)
         real  a,f
         real  h,x,s
         integer i
         s=(f(0.0)+f(a))/2.0
         h=a/10.0; x=h
         do i=1,9; s=s+f(x); x=x+h; end do
         sintg=s*h
      end function sintg
   end program numint

   real function esin(y)
      real y
      esin=exp(-y)*sin(y)
   end function esin
```

Обратите внимание на блок интерфейса к esin и обязательную спецификацию косинуса как встроенной функции (INTRINSIC).

12.5. Квадратурные формулы Чебышева

В формулах интерполяционных квадратур коэффициенты $\{A_j\}$, вообще говоря, различаются между собой, что усложняет вычисления. Кроме того, когда значения $\{f(x_j)\}$ находятся с ошибками (например, получены экспериментально), то сумма $\sum_{j=0}^{k} A_j f(x_j)$ также будет иметь случайную ошибку, величина которой зависит от выбора $\{A_j\}$. Доказано, что эта ошибка в среднем будет наименьшей, если коэффициенты $\{A_j\}$ равны между собой.

Для определенности будем рассматривать стандартный интервал [-1,1]. Выясним, нельзя ли выбрать узлы $x_0, x_1, \ldots, x_k \in [-1, 1]$ так, чтобы выполнялось $A_0 = A_1 = \ldots = A_k = A$ и квадратурная формула

была верна для некоторых многочленов степени выше k. Если такой выбор возможен, то из первой формулы (12.3.3) следует

$$A = \frac{2}{k+1},$$

и квадратурная формула имеет вид

$$\int_{-1}^{1} f(x)\,dx \approx \frac{2}{k+1}[f(x_0) + f(x_1) + \ldots + f(x_k)]. \qquad (12.5.1)$$

Для определения узлов выпишем уравнения типа (12.3.3) для степеней $x^m, \quad m = \overline{1, k+1}$, и учтем равенство

$$\sum_{j=0}^{k} x_j^m = \begin{cases} 0, & \text{если } m \text{ нечетно;} \\ \dfrac{k+1}{m+1}, & \text{если } m \text{ четно.} \end{cases} \qquad (12.5.2)$$

Все такие узлы оказываются вещественными и принадлежащими стандартному интервалу только для случаев $k = 0, 1, 2, 3, 4, 5, 6, 8$. При этом квадратурная формула (12.5.1) точна для всех многочленов степени $(k+1)$, если k четно, и степени $(k+2)$, если k нечетно.

Найдем соответствующие квадратурные формулы для $k = 0, 1, 2$.

1) $k = 0$. Коэффициент $A = 2/(0+1) = 2$; $x_0 = 0$ — единственное уравнение системы (12.5.1) для $k = 0$. Получаем квадратурную формулу

$$\int_{-1}^{1} f(x)\,dx \approx 2f(0).$$

2) $k = 1$. Коэффициент $A = 2/(1+1) = 1$. Система (12.5.2) принимает вид

$$\begin{aligned} x_0 + x_1 &= 0; \\ x_0^2 + x_1^2 &= \frac{1+1}{2+1} = \frac{2}{3}. \end{aligned}$$

Очевидно ее решение: $-x_0 = x_1 = \sqrt{1/3}$. Заметим, что выполняется также равенство $x_0^3 + x_1^3 = 0$, а это означает, что квадратурная формула

$$\int_{-1}^{1} f(x)\,dx \approx f(-1/\sqrt{3}) + f(1/\sqrt{3})$$

верна и для многочленов третьей степени.

3) $k = 2$. Коэффициент $A = 2/(2+1) = 2/3$. Система (12.5.2) принимает вид

$$\begin{aligned} x_0 + x_1 + x_2 &= 0; \\ x_0^2 + x_1^2 + x_2^2 &= 3/3 = 1; \\ x_0^3 + x_1^3 + x_2^3 &= 0, \end{aligned}$$

из которой (если считать $x_0 < x_1 < x_2$) следует $-x_0 = x_2 = 1/\sqrt{2}$ и $x_1 = 0$. Таким образом, получаем квадратурную формулу

$$\int_{-1}^{1} f(x)\,dx \approx \frac{2}{3}[f(-1/\sqrt{2}) + f(0) + f(1/\sqrt{2})],$$

которая точна для всех многочленов степени не выше третьей. Можно показать, что она неточна для многочленов четвертой степени.

В заключение приведем таблицу узлов Чебышева для значений $k = \overline{0,4}$.

Таблица 12.5. Узлы квадратурных формул Чебышева

k	Узлы
0	x_0=0
1	$x_1 = -x_0 = 0.$
2	$x_2 = -x_0 = 0.707106781$; $x_1 = 0$
3	$x_3 = -x_0 = 0.794654472$; $x_2 = -x_1 = 0.187592474$
4	$x_4 = -x_0 = 0.832497487$; $x_3 = -x_1 = 0.374541410$; $x_2 = 0$

12.6. Квадратурные формулы Гаусса

12.6.1. Многочлены Лежандра

Выбор узлов в квадратурных формулах *Гаусса* связан с многочленами Лежандра. Пусть n — целое положительное число. Рассмотрим функции вида

$$\tilde{P}_n(x) = \frac{d^n}{dx^n}(x^2 - 1)^n. \qquad (12.6.1)$$

Могут быть установлены следующие свойства этих функций:

1. $\tilde{P}_n(x)$ есть многочлен степени n с четными степенями (если n четно) и с нечетными, если n нечетно.

2. $\tilde{P}_n(x)$ ортогонален любому многочлену $q(x)$ степени ниже n на отрезке [-1,1], иначе говоря, $\int\limits_{-1}^{1} q(x)\tilde{P}_n(x)\,dx = 0$.

Для доказательства вычислим

$$\begin{aligned}\int\limits_{-1}^{1} q(x)\tilde{P}_n(x)\,dx &= \int_{-1}^{1} q(x)\frac{d}{dx}\left\{\frac{d^{n-1}}{dx^{n-1}}(x^2-1)^n\right\}dx \\ &= q(x)\,\frac{d^{n-1}(x^2-1)^n}{dx^{n-1}}\Big|_{-1}^{1} - \int_{-1}^{1} q'(x)\frac{d}{dx}\left\{\frac{d^{n-2}(x^2-1)^n}{dx^{n-2}}\right\}dx.\end{aligned}$$

Первое слагаемое на границах отрезка [-1,1] обращается в нуль, так как производная содержит множитель (x^2-1). Поэтому

$$\int_{-1}^{1} q(x)\tilde{P}_n(x)\,dx = -\int_{-1}^{1} q'(x)\frac{d}{dx}\left\{\frac{d^{n-2}(x^2-1)^n}{dx^{n-2}}\right\}dx.$$

Проделав операцию взятия интеграла по частям n раз, получим

$$\int_{-1}^{1} q(x)\tilde{P}_n(x)\,dx = (-1)^n\int_{-1}^{1} q^{(n)}(x)(x^2-1)^n\,dx = 0.$$

Поскольку степень $q(x)$ меньше n, убеждаемся в $q^{(n)}(x)\equiv 0$, что и является условием ортогональности. Отсюда следует попарная ортогональность различных многочленов $P_n(x)$ и $P_m(x)$ в случае $m\neq n$ (один из многочленов обязательно имеет более низкую степень, чем другой). ▲

Замечание. Всякий многочлен $\omega(x)$ степени n, ортогональный ко всем многочленам $q(x)$ степени ниже n, отличается от $\tilde{P}_n(x)$ только постоянным множителем. Действительно, при любом выборе постоянной C многочлен $r(x) = \omega(x) - C\tilde{P}_n(x)$ также ортогонален ко всем $q(x)$. С другой стороны, при значении C, равном отношению старших коэффициентов $\omega(x)$ и $\tilde{P}_n(x)$, многочлен $r(x)$ имеет степень ниже n и поэтому ортогонален сам себе, т. е. $\int_{-1}^{1} r^2(x)\,dx = 0$. Из этого равенства и непрерывности $r(x)$ следует $r(x)\equiv 0$, т. е. $\omega(x) = \tilde{P}_n(x)$.

3. Все корни $\tilde{P}_n()$, $n\geq 1$, вещественны, различны и лежат на интервале (-1,1).

В самом деле, на основании второго свойства имеем

$$\int_{-1}^{1} \tilde{P}_n(x) \cdot 1 \, dx = 0.$$

Но это означает, что $\tilde{P}_n(x)$ наверняка меняет знак на отрезке [-1,1]. Пусть $x_1, x_2, \ldots, x_k$ — все точки этого отрезка, в которых $\tilde{P}_n(x)$ меняет знак. Рассмотрим многочлен $g(x) = (-x_1)(-x_2)\ldots(-x_k)$ степени $k \le n$. Поскольку смена знаков сомножителей происходит одновременно, произведение $\tilde{P}_n(x)g(x)$ знака на отрезке [-1,1] не меняет. Поэтому $\int_{-1}^{1} \tilde{P}_n(x)g(x)\,dx \neq 0$. Отсюда следует, что должно быть $k = n$, причем корни $g(x)$ все вещественны и различны (напомним, что в каждом из них предполагалась смена знака $g(x)$). ▲

Многочленами Лежандра называются многочлены вида

$$P_n(x) = \frac{1}{n\,!2^n}\tilde{P}_n(x) = \frac{1}{n\,!2^n}\frac{d^n}{dx^n}(x^2-1)^n. \qquad (12.6.2)$$

Они обладают всеми перечисленными выше свойствами. Кроме того, $P_n(+1) = 1$ и $P_n(-1) = (-1)^n$. Полиномы Лежандра могут быть найдены по рекуррентным формулам

$$P_n(x) = \frac{1}{n}[(2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)], \qquad n = 2, 3, \ldots$$

Приведем первые пять полиномов Лежандра:

$$\begin{array}{lcl} P_0(x) & = & 1; \\ P_1(x) & = & x; \\ P_2(x) & = & (3x^2-1)/2; \\ P_3(x) & = & (5x^3-3x)/2; \\ P_4(x) & = & (35x^4-30x^2+3)/8. \end{array} \qquad (12.6.3)$$

12.6.2. Выбор узлов квадратурной формулы Гаусса

Для определенности будем рассматривать отрезок [-1,1]. Если коэффициенты $\{A_j\}$ определяются формулами (12.3.1), то при любом выборе узлов

$$x_0, x_1, \ldots, x_k, \qquad x_j \in [-1, 1], \qquad x_i \neq x_j \text{ для } i \neq j \qquad (12.6.4)$$

равенство

$$\int_{-1}^{1} Q(x)\,dx = \sum_{j=0}^{n} A_j Q(x_j) \tag{12.6.5}$$

имеет место для всех полиномов $Q(x)$ степени не выше k.

Можно ли подобрать узлы (12.6.4) так, чтобы равенство (12.6.5) оказалось верным для многочленов степени выше k, именно для всех многочленов степени не выше $(k+r)$, где $r>0$ — целое число? Набор узлов (12.6.4), удовлетворяющий этому условию, назовем r-*удачным*.

Для произвольного набора (12.6.4) образуем многочлен

$$\omega(x) = (x-x_0)(x-x_1)\dots(x-x_k)$$

степени $(k+1)$. Через $q(x)$ и $R(x)$ обозначим частное и остаток от деления любого многочлена $Q(x)$ степени не выше $(k+r)$ на $\omega(x)$. Тогда имеет место тождество

$$Q(x) \equiv \omega(x)q(x) + R(x). \tag{12.6.6}$$

Отметим, что степень $q(x)$ не превосходит $(k+r)-(k+1) = r-1$, а степень $R(x)$ не превосходит k.

Так как для $j=\overline{0,k}$ $\omega(x_j)=0$, то $Q_j(x)=R_j(x)$ и поэтому

$$\sum_{j=0}^{k} A_j Q(x_j) = \sum_{j=0}^{k} A_j R(x_j). \tag{12.6.7}$$

Поскольку степень $R(x)$ не превосходит k, то для $R(x)$ должно выполняться равенство (12.6.5):

$$\int_{-1}^{1} R(x)\,dx = \sum_{j=0}^{k} A_j R(x_j). \tag{12.6.8}$$

Следовательно, левые части равенств (12.6.7) и (12.6.8) равны между собой при любом выборе узлов (12.6.4). Из равенства (12.6.5) вытекает, что набор (12.6.4) является r-удачным [выполняется (12.6.5)], если равны между собой левые части равенств (12.6.5) и (12.6.8), т. е. интеграл от разности $Q(x)-R(x)$ обращается в нуль. На основании (12.6.6) заключаем: набор (12.6.4) является r-удачным, если и только если

$$\int_{-1}^{1} \omega(x)q(x)\,dx = 0 \tag{12.6.9}$$

для любого многочлена $q(x)$ степени не выше $(r-1)$ — иначе говоря, если многочлен $\omega(x)$ степени $(k+1)$ ортогонален любому многочлену степени не выше $(r-1)$.

Равенство (12.6.9) невозможно при $q(x) = \omega(x)$, поскольку $\int\limits_{-1}^{1} \omega^2(x)\,dx > 0$. Это означает, что при $r-1 \geq k+1$ упомянутое равенство не может иметь место для *всех* многочленов $q(x)$ степени не выше $(r-1)$. Таким образом, для $r \geq k+2$ не существует r-удачных наборов (12.6.4).

Пусть $r = k+1$. Как было показано в разд. 12.6.1, многочлен $\omega(x)$ степени $(k+1)$, ортогональный ко всем многочленам $q(x)$ степени не выше k на отрезке [-1,1], с точностью до постоянного множителя совпадает с многочленом Лежандра $P_{k+1}(x)$. Отсюда вывод: при $r = k+1$ существует единственный r-удачный набор узлов — корни многочлена $P_{k+1}(x)$. Коэффициенты $\{A_j\}$ могут быть найдены из системы уравнений типа (12.3.3). Соответствующая квадратурная формула называется *формулой Гаусса* порядка k. Она точна для всех многочленов степени $k+r = k+(k+1) = 2k+1$. Для каждого r, $0 < r < k+1$, существует много r-удачных наборов. ▲

Найдем узлы и веса квадратурной формулы Гаусса для $k = 0, 1, 2$.

1) $\underline{k=0}$. Степень многочлена Лежандра $k+1 = 1$, а сам многочлен $P_1(x) = x$. Его единственный корень $x_0 = 0$. Из первого уравнения системы (12.3.1) находим $A_0 = 2$. Получаем формулу

$$\int\limits_{-1}^{1} f(x)\,dx \approx 2f(0),$$

которая точна для всех многочленов степени не выше $2k+1 = 1$. Геометрический смысл ее очевиден. Она совпадает с формулой Чебышева для $k = 0$.

2) $\underline{k=1}$. Теперь $k+1 = 2$, и узлы могут быть найдены из уравнения $P_2(x) = \frac{1}{2}(3x^2-1) = 0$; следовательно, $-x_0 = x_1 = 1/\sqrt{3}$. Первые два уравнения из (12.3.3) дают систему для отыскания коэффициентов

$$\begin{aligned} A_0 \quad &+A_1 \quad &&= 2; \\ -\frac{A_0}{\sqrt{3}} \quad &+\frac{A_1}{\sqrt{3}} \quad &&= 0 \end{aligned}$$

с очевидным решением $A_0 = A_1 = 1$. Итак, формулы Чебышева и Гаусса совпадают и при $k = 1$.

3) $\underline{k = 2.}$ Из уравнения $P_3(x) = \frac{1}{2}(5x^3 - 3x) = 0$ находим $-x_0 = x_2 = \sqrt{3/5}$. Для определения коэффициентов A_0, A_1, A_2 имеем систему трех линейных уравнений

$$\begin{array}{llll} A_0 & +A_1 & +A_2 & = 1 - (-1) = 2; \\ A_0(-\sqrt{3/5}) & +A_1 \cdot 0 & +A_2\sqrt{3/5} & = \dfrac{1^2 - (-1)^2}{2} = 0; \\ A_0(-\sqrt{3/5})^2 & +A_1 \cdot 0^2 & +A_2(\sqrt{3/5})^2 & = \dfrac{1^3 - (-1)^3}{3} = \dfrac{2}{3}. \end{array}$$

Из нее выводим $A_0 = A_2 = 5/9, \quad A_1 = 8/9$.

Приведем табл. 12.6 узлов и весов интегрирования по Гауссу для нескольких первых значений k.

Таблица 12.6. Узлы и веса квадратурной формулы Гаусса

k	Узлы	Веса
0	$x_0 = 0$	$A_0 = 2.0$
1	$-x_0 = x_1 = 0.577350269$	$A_0 = A_1 = 1.0$
2	$-x_0 = x_2 = 0.774596669$ $x_1 = 0$	$A_0 = A_2 = 0.555555556,$ $A_1 = 0.888888889$
3	$-x_0 = x_3 = 0.861136312$ $-x_1 = x_2 = 0.339981044$	$A_0 = A_3 = 0.347854845$ $A_1 = A_2 = 0.652145155$
4	$-x_0 = x_4 = 0.906179846$ $-x_1 = x_3 = 0.538169310$ $x_2 = 0$	$A_0 = A_4 = 0.236926885$ $A_1 = A_3 = 0.478628670$ $A_2 = 0.568888889$

Использование формул Чебышева и Гаусса при больших k затрудняется необходимостью оценивания производных высоких порядков. Выход можно найти в построении составных формул (см. разд. 12.8). Здесь полезны квадратурные формулы *Маркова*. Их алгебраическая точность ниже, чем у соответствующих формул Гаусса, но зато некоторые узлы могут фиксироваться (в частности, совпадать с границами интервала интегрирования).

12.7. Погрешности квадратурных формул

При использовании квадратурных формул возникает вопрос о выборе типа квадратуры, узлов, оценке точности и т. д. Неудачное расположение узлов может дать сильно искаженные результаты — в особенности при наличии в интервале интегрирования особенностей или большого числа нулей интегрируемой функции. Рекомендуется построить график последней и разбить его на интервалы с постоянным знаком. Для сильно колеблющихся функций следует применять специальную технику интегрирования.

Приведем оценки для остатков квадратурных формул.

Формула трапеций:

$$R \le \frac{b-a}{12} h^2 M_2.$$

Формула Симпсона:

$$R \le \frac{b-a}{180} h^4 M_4.$$

Формула Гаусса при k узлах:

$$R_k \le \frac{(b-a)^{2k+1}(k!)^4 M_{2k}}{[(2k)!]^3(2k+1)}.$$

В частности,

$$\begin{aligned} R_2 &\le \frac{1}{135}\Big(\frac{b-a}{2}\Big)^5 M_4; \\ R_3 &\le \frac{1}{15750}\Big(\frac{b-a}{2}\Big)^7 M_5; \\ R_4 &\le \frac{1}{3472875}\Big(\frac{b-a}{2}\Big)^9 M_8 \end{aligned}$$

и т. д.

При численном интегрировании следует помнить о том, что бывают функции с очень большими производными высокого порядка. Такой эффект наблюдается для простейшей гармонической функции с большой круговой частотой ω, поскольку

$$\begin{aligned} \sin' \omega t &= \omega \cos \omega t, \\ \sin'' \omega t &= -\omega^2 \sin \omega t, \\ \sin''' \omega t &= -\omega^3 \cos \omega t \end{aligned}$$

и т. д. Для достижения сравнимой точности при интегрировании слабо меняющейся функции шаг можно выбирать больше, чем в случае резко меняющихся. Если подынтегральная функция на разных отрезках интервала интегрирования ведет себя по-разному, необходим адаптивный алгоритм с переменным шагом.

Оценки погрешности, полученные аналитически, оказываются слишком сложными или слишком пессимистическими. Во многих прикладных задачах подинтегральная функция задается даже не формулой, а алгоритмически. Поэтому в реальных задачах остаток оценивается на основе не аналитических, а *числовых* данных. Распространенный метод оценки погрешностей состоит в комбинировании нескольких квадратурных правил. В простейшем случае применяют два правила (например, трапеций и Симпсона) или варианты составных формул и за оценку погрешности менее точного из них принимают модуль разности двух приближений.

12.8. Составные квадратурные формулы

12.8.1. Идея и достоинства

Рассмотренные выше квадратурные формулы k-го порядка (назовем их простыми) связаны с непосредственным размещением $(k+1)$ узлов по всему интервалу интегрирования. При необходимости вычисления интеграла по большому промежутку для получения высокой точности интегрирования число узлов приходится выбирать весьма значительным. Это затрудняет расчет узлов и весовых коэффициентов, а в некоторых случаях (например, в формулах Чебышева при $k > 8$) делает их определение невозможным.

Составные правила хороши тем, что позволяют легко строить правила с большим числом узлов. Здесь интервал интегрирования разбивается на n частей, к каждой из них применяется квадратурная формула с малым числом узлов и полученные результаты складываются.

Простые формулы Котеса относятся к замкнутому типу, симметричны и предполагают вычисление подынтегральной функции в крайних точках отрезка $[a, b]$. Поэтому при дополнительном разбиении на подынтервалы и последующем суммировании частичных сумм веса значений функции на границах подынтервалов (кроме внешних) удваиваются. Приведем только частные варианты составных формул Котеса, получившие

наибольшее распространение.

Составная формула трапеций имеет вид

$$\int_a^b f(x)\,dx \approx h\left[\frac{f(a)+f(b)}{2}+\sum_{i=1}^{n-1} f(a+ih)\right], \qquad (12.8.1)$$

где $h=(b-a)/n$.

Составная формула Симпсона

$$\begin{aligned}\int_a^b f(x)\,dx &\approx \frac{h}{3}[f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+\ldots\\ &+ 2f(a+4h)+\ldots+4f[a+(2n-1)h]+f(b)]\\ &= \frac{h}{3}\Big[f(a)+f(b)+4\sum_{i=1}^{n} f[a+(2i-1)h]+\sum_{i=1}^{n-1} f(a+2ih)\Big].\end{aligned} \qquad (12.8.2)$$

В этой формуле $h=(b-a)/(2n)$.

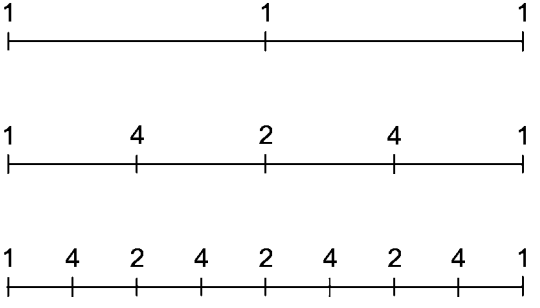

Рис. 12.3. К пересчету сумм ординат для составной формулы Симпсона

Из рассмотрения рис. 12.3 следует, что ординаты в граничных точках исходного интервала можно вычислить однократно и учитывать в интегральной сумме с весом «1». На каждом очередном (n-м) шаге последовательного дробления интервалов с весом «2» берется сумма ординат, на предыдущем шаге учитываемых с весами «2» и «4», и вычисляются 2^n новых ординат, учитываемых с весом «4».

Если процесс заканчивается на шаге m, то общее число обращений к подынтегральной функции составит

$$\sum_{n=0}^{m} 2^n+2=\frac{2^m\cdot 2-1}{2-1}+2=2^{m+1}+1.$$

Рекомендуем читателю провести аналогичные рассуждения для формулы «трех восьмых».

12.8.2. Процессы Ромберга и Эйткена

Дополнительное преимущество составных формул связано с поведением их остатков при удвоении k.

Принцип Рунге. Пусть к приближенному вычислению значения I данного интеграла применяется некая квадратурная формула p-го порядка точности h^p из семейства составных формул Ньютона—Котеса. При условии непрерывности p-й производной это означает существование такой постоянной C, что

$$I = I^{(p)}(h) + Ch^p. \tag{12.8.3}$$

При уменьшении шага вдвое по той же формуле можно записать аналогичное равенство, но с другой постоянной:

$$I = I^{(p)}(h/2) + C_1(h/2)^p. \tag{12.8.4}$$

Можно принять

$$C \approx C_1 \approx \frac{I^{(p)}(h/2) - I^{(p)}(h)}{h^p - (h/2)^p}.$$

Подставив это выражение в (12.8.3), получаем

$$I \approx I^{(p)}(h/2) + \frac{I^{(p)}(h/2) - I^{(p)}(h)}{2^p - 1}. \tag{12.8.5}$$

Эта формула позволяет по результатам двух расчетов метода порядка p найти уточненное значение с порядком точности $p+1$.

Процесс Ромберга, опираясь на принцип Рунге, позволяет контролировать точность численного интегрирования по результатам двойного счета (с исходным и с половинным шагом), а также уточнить последнее полученное приближение. Например, для составного правила трапеций $|R_{2k+1}| \approx |R_{k+1}|/4$. Из условия

$$I = T_{k+1} + R_{k+1} = T_{2k+1} + R_{2k+1}$$

получаем оценку

$$|T_{k+1} - T_{2k+1}| = |R_{k+1} - R_{2k+1}| \approx \frac{3}{4}|R_{k+1}|.$$

Таким образом, разумно принять в качестве приближения к интегралу T_{2k+1}, а за оценку погрешности взять $\frac{4}{3}|T_{k+1} - T_{2k+1}|$.

Аналогичный процесс можно применить и к другим типам квадратур — в частности, к формулам Симпсона. Его эффективность основывается на трех обстоятельствах:

- использовании уже найденных ординат интегрируемой функции;
- экономном хранении их в виде перевычисляемых сумм;
- возможной остановке процесса при достижении заданной точности.

Приведем программу экономного численного интегрирования по составной формуле Симпсона для вычисления

$$\int_0^1 e^x\,dx = e - 1 = 1.718281828459045.$$

```
program integs
   real*8 a,b,h,e1,f,s1,s2,s4,v,v1,v2,dv,x

   e1=exp(1d0)-1d0;         ! Контрольное значение
   a=0; b=1.0; h=(a+b)/2.0
   s1=f(a)+f(b);            ! Сумма крайних ординат -
   !                          позже не пересчитывается
   s2=0; s4=f(a+h)          ! Простая формула Симпсона
   v=(s1+4*s4)*h/3.0;  v1=0
   n=1
   print *, 'n = ', n, '  dv = ', v-e1

   do n=2,8
      v1=v; h=h/2.0
      s2=s4+s2; s4=0         ! Пересчет накопленных сумм
      x=a+h                  ! Нач. абсцисса для новых ординат
      do while (x <= b-h/2)
         s4=s4+f(x); x=x+2*h! Накопление суммы новых ординат
      end do
      v=(s1+2*s2+4*s4)*h/3.0! Очередной Симпсон-интеграл
      print *, 'n = ',n
      write (*,30) 'Simpson-delta = ',v-e1
```

```
      dv=(v-v1)/15.0; v2=v+dv  ! Ромберг-поправка
      write (*,30) 'Romberg-delta = ',v2-e1
   end do
30 format(1x,A,e12.3)
end

      real*8 function f(x)
      real*8 x
      f=exp(x)
      end
```

Эта программа дала следующие результаты:

Таблица 12.7. Погрешности интегрирования по Симпсону и Ромбергу

Метод	Номер шага							
	1	2	3	4	5	6	7	8
Симпсона	5.79e-4	3.70e-5	2.33e-6	1.40e-7	9.10e-09	5.69e-10	3.56e-11	2.22e-12
Ромберга		8.59e-7	1.38e-8	2.11e-10	3.39e-12	5.24e-14	7.85e-16	-1.48e-16

Процесс Эйткена требует трехкратного расчета при различных шагах разбиения, причем $h_2/h_1 = h_3/h_2 = q < 1$. При делении шага пополам $q = 0.5$. Пусть в результате численного интегрирования получены значения I_1, I_2, I_3. Тогда уточненное значение интеграла

$$I = I_1 - \frac{(I_1 - I_2)^2}{I_1 - 2I_2 + I_3}.$$

Приведем программу численного интегрирования по Гауссу (7 узлов) с двукратным дроблением шага и поправкой по Эйткену.

```
program gauss7
   real*8  a,b,e1,f,h,s,t,pt,wt,x,y,y2,z
   integer i,j,k,m
   dimension pt(7),wt(7),y(3),z(3)
   data pt /-0.949107912342758,   -0.741531185599394,  &
           -0.405845151377397, 0, 0.405845151377397,  &
            0.741531185599394,    0.949107912342758/, &
        wt / 0.129484966168870,    0.279705391489277,  &
             0.381830050505119,    0.417959183673469,  &
             0.381830050505119,    0.279705391489277,  &
             0.129484966168870/
   a=0; b=1d0
```

```
   e1=exp(1d0)-1.0

   k=1; h=(b-a)/2
   do m=1,3
      s=0                  ! Обнуление интегральной суммы
      x=a+h                ! Середина начального интервала
      print *, 'm = ',m    ! Номер шага
      do j=1,k
         do i=1,7
            t=x+h*pt(i)    ! Очередной узел
            s=s+wt(i)*f(t)! Накопление суммы ординат
         end do
         x=x+2*h           ! Середина следующего интервала
      end do
      z(m)=s*h
      y(m)=z(m)-y1         ! Погрешн. составной формулы Гаусса
      print *, 'Delta = ', y(m)
      h=h/2; k=2*k
   end do
   t=(z(1)-z(2))**2
   x=z(1)-2*z(2)+z(3)
   y2=z(1)-t/x             ! Исправленный по Эйткену
   print *, 'После Эйткена '
   print *, 'Delta = ', y2-y1
end
.............................
```

Опущенное описание интегрируемой функции аналогично предыдущему случаю.

В данной программе поучительна работа с составными формулами открытого типа (вычисление начальной и срединной точек очередного интервала). Количество перевычисляемых ординат уменьшается при использовании квадратур *А. А. Маркова*, которые предполагают совпадение по крайней мере одной из точек разбиения с его концом.

Вычисленные по шагам данной программы погрешности составили соответственно 1.29e-9, 3.35e-10, 8.46e-11. Погрешность результата, исправленного по Эйткену, составила -4.58e-12.

При необходимости дальнейшего дробления шага можно наладить циркуляцию данных в переменных, представляющих смежные при-

ближения: `z(1)=z(2); z(2)=z(3);` переменная `z(3)` получает значение последнего приближения. Таким образом можно получать значения интеграла, исправленные по Эйткену, на *каждом* шаге процесса, начиная с третьего.

Отметим принципиальную особенность эффекта от введения поправок по Рунге и Эйткену при вычислении определенных интегралов. В отличие от итерационных процессов решения уравнений, здесь исправленное значение *не может использоваться как база для дальнейших уточнений*.

12.9. Квадратуры Гаусса—Кронрода

Выше было показано, что формулы Гаусса являются наилучшими при заданном числе узлов. Теперь нас интересует, как выбрать пару правил Гаусса, дающую и приближенное значение интеграла, и оценку погрешности. Обозначим G_n n-точечное правило Гаусса. Мы вправе ожидать, что G_{2n-1} — гораздо более точный результат, и величину $|G_n - G_{2n-1}|$ можно принять за оценку погрешности G_n. Но разные формулы Гаусса не имеют общих узлов, исключая середину отрезка интегрирования. Поэтому в данном варианте потребуется $3n-1$ вычислений подынтегральной функции.

Советский математик А. Кронрод в 1965 г. предложил правило

$$K_{2n+1} = \sum_{i=1}^{n} a_i f(x_i) + \sum_{j=1}^{n+1} b_j f(y_j),$$

которое имеет n общих узлов с G_n. Кроме того, K_{2n+1} имеет $n+1$ дополнительных узлов и новые веса $\{a_i, b_j\}$. Алгебраическая степень точности правила Гаусса G_n равна $2n-1$. Значения $3n+2$ новых параметров правила K_{2n+1} были найдены Кронродом так, что алгебраическая степень точности этого правила равна $3n+1$. Цена вычисления пары составляет $2n+1$ значений подынтегральной функции. Опыт показывает ([37, с. 187]), что $|G_n - K_{2n+1}|$ существенно превосходит погрешность более точного приближения, и можно принять

$$\Delta = (200|G_n - K_{2n+1}|)^{1.5}.$$

Пара Гаусса—Кронрода (стандарт G_7, K_{15}) вместе с этой оценкой считается одним из самых эффективных методов вычисления интегралов

общего вида. В табл. 12.8 приведены соответствующие узлы и веса (первые четыре строки — гауссовы, последующие — по Кронроду).

Таблица 12.8. Узлы и веса квадратуры Гаусса — Кронрода

Узлы	Веса
$\mp$ 0.94910 79123 42758	0.12948 49661 68870
$\mp$ 0.74153 11855 99394	0.27970 53914 89277
$\mp$ 0.40584 51513 77397	0.38183 00505 05119
0.00000 00000 00000	0.41795 91836 73469
$\mp$ 0.99145 53711 20813	0.02293 53220 10529
$\mp$ 0.94910 79123 42759	0.06309 20926 29979
$\mp$ 0.86486 44233 59769	0.10479 00103 22250
$\mp$ 0.74153 11855 99394	0.14065 32597 15525
$\mp$ 0.58608 72354 67691	0.16900 47266 39267
$\mp$ 0.40584 51513 77397	0.19035 05780 64785
$\mp$ 0.20778 49550 07898	0.20443 29400 75298
$\mp$ 0.00000 00000 00000	0.20948 21410 84728

12.10. Несобственные интегралы

«Ценным подарком» исследователю являются несобственные интегралы, под которыми понимаются интегралы от разрывных функций и/или с бесконечными пределами. Поскольку интегралы с разрывами первого рода (конечными скачками подынтегральной функции) сводятся к сумме интегралов по отрезкам непрерывности, в дальнейшем обсуждаются только случаи с бесконечными разрывами подынтегральной функции. Ясно, что такие функции (и любые, интегрируемые на бесконечном интервале) невозможно хорошо приблизить обсуждавшимися нами ранее полиномами[1], и для вычисления квадратур должна применяться особая стратегия.

Здесь помогают четыре метода:

- интегрирование по частям;
- устраняющая особенность замена переменной;

[1] Можно использовать дробно-рациональную аппроксимацию, в которой степень знаменателя выше степени числителя.

- выделение особенности;
- «наступление на особенность»,

которые можно применять как порознь, так и в комбинации. Например,

$$\int\limits_0^1 \frac{e^x}{\sqrt{x}}\,dx = 2\int\limits_0^1 e^x d\sqrt{x} = 2e^x\sqrt{x}\Big|_0^1 - 2\int\limits_0^1 e^x\sqrt{x}\,dx = 2e - \tilde{I},$$

где остаточный интеграл

$$\tilde{I} = \int\limits_0^1 e^x\sqrt{x}\,dx$$

имеет удобную для численного интегрирования подынтегральную функцию.

Для преобразования области $[0,\infty)$ в конечную и/или для ускорения стремления подынтегральной функции к нулю рекомендуются подстановки $x = 1/z$, $x = -\ln(t)$ и $x = t/(1-t)$. Пусть нужно вычислить

$$I = \int\limits_1^\infty \frac{dx}{\sqrt{x}(1+x)}.$$

Заменой переменной $x = 1/z$ и дифференциала $dx = -dz/z^2$ получаем

$$I = -\int\limits_1^0 \frac{dz\sqrt{z}}{z^2(1+1/z)} = \int\limits_0^1 \frac{dz}{\sqrt{z}(1+z)}.$$

Интервал сведен к конечному, но осталась особенность в нуле. Применим интегрирование по частям:

$$I = \int\limits_0^1 \frac{1}{1+z}\,d(2\sqrt{z}) = \frac{2\sqrt{z}}{z+1}\Big|_0^1 - \int\limits_0^1 2\sqrt{z}\left(-\frac{dz}{(z+1)^2}\right) = 1 + 2\int\limits_0^1 \frac{\sqrt{z}\,dz}{(z+1)^2}.$$

Последний интеграл численно берется без проблем. Однако еще лучше использовать его явное представление

$$\int \frac{\sqrt{z}\,dz}{(z+1)^2} = -\frac{\sqrt{z}}{z+1} + \arctan\sqrt{z}.$$

Это пример необходимости хорошего знания классического анализа — или необходимости предварительного обращения к автоматизированным средствам символьной математики (Maple, Mathematica и др.).

Теперь выделим из имеющей особенность подынтегральной функции $f(x)$ функцию $g(x)$, имеющую ту же особенность, но интегрируемую элементарно и оставляющую достаточно гладкую разность:

$$\int_a^b f(x)\,dx = \int_a^b g(x)\,dx + \int_a^b [f(x) - g(x)]\,dx.$$

К интегралу от разности может быть применена обычная технология численного интегрирования.

Рассмотрим случай, когда процесс интегрирования ведется на полуоси. Сходимость интеграла определяется только поведением функции $f(x)$ на особенности, т. е. при $x \to \infty$. Как правило, $f(x)$ сравнивается со степенной функцией C/x^p, интеграл от которой

$$\int f(x)\,dx = \frac{C}{(p-1)x^{p-1}}.$$

При подстановке в эту формулу «бесконечности» конечное значение получается только при $p > 1$. Например, функция

$$f(x) = \frac{x^{2/3}}{\sqrt[n]{x^4+1}}$$

асимптотически ведет себя как степенная с показателем степени в знаменателе $4/n - 2/3$, и интеграл сходится при $n < 12/5$. Пусть $n = 2$. Положим $\varphi(x) = x^{-4/3}$. Тогда

$$\int \varphi(x)\,dx = \frac{1}{(4/3-1)x^{1/3}} = 3/\sqrt[3]{x}.$$

Поэтому

$$I = \int_a^\infty f(x)\,dx = \int_a^\infty \varphi(x)\,dx + \int_a^\infty [f(x) - \varphi(x)]\,dx = I_1 + I_2.$$

После подстановки пределов первый интеграл дает

$$I = \int_a^\infty f(x)\,dx = -3/\sqrt[3]{x}.$$

Подобрав асимптотически эквивалентную $f(x)$ степенную функцию, можно аналогично предыдущему пункту аналитически вычислить главную часть интеграла. Гладкий остаток будет значительно быстрее сходиться к нулю и может быть проинтегрирован численно на конечном интервале.

«Наступление на особенность» предполагает сведение области интегрирования к конечным отрезкам разной длины и контроль малости вклада «хвостов». Обычно оно должно подкрепляться физическими соображениями. Задача ставится в виде

$$\int_a^\infty f(x)\,dx \approx \int_a^c f(x)\,dx + \int_c^d f(x)\,dx = I_1 + I_2.$$

Например, при интегрировании по распределению случайной величины можно назначить правую границу $c = \bar{x} + 3\sigma$, где $\bar{x}$ — математическое ожидание случайной величины, а σ — среднеквадратическое отклонение. Прежде всего достаточно точно вычисляется I_1. Назначается базовое приращение Δ правой границы интервала, например $\Delta = 2\sigma$. Далее выполняются следующие действия:

1. Вычислить $d = c + \Delta$ и $I_2 = \int_c^d f(x)\,dx$. Положить $I = I_1 + I_2$.
2. Если $|I_2/I| < \varepsilon$, перейти к этапу 4.
3. Положить $c = c + \Delta$. Увеличить Δ вдвое и перейти к этапу 1.
4. Конец вычислений.

Сходимость или расходимость интеграла с особенностью (бесконечным значением подынтегральной функции) в *начале* интервала интегрирования опять же устанавливается сопоставлением подынтегральной функции со степенной. Можно показать, что

$$\int_a^b \frac{C}{(x-a)^p}\,dx$$

сходится только при $p < 1$.

Если необходимо вычислить $\int_0^b f(x)\,dx$, причем $f(0) = \infty$, то

задача представляется в виде

$$\int_0^b f(x)\,dx = I_1 + I_2,$$

где

$$I_1 = \int_0^a f(x)\,dx, \quad I_2 = \int_a^b f(x)\,dx.$$

Точка a выбирается достаточно близкой к нулю. После этого численным (предпочтительно адаптивным) методом находится I_2 и с применением одного из *открытых* квадратурных методов (например, Гаусса) — I_1. «Открытость» формулы необходима, чтобы избежать вычисления $f(0)$.

Дальнейший процесс описывается следующими этапами:

1. Разделить $[0, a]$ пополам. Вычислить
$I_1' = \int_0^{a/2} f(x)\,dx\,, \quad I_1'' = \int_{a/2}^a f(x)\,dx\,, \quad I_3 = I_1' + I_1''\,.$

2. Если $|(I_3 - I_1)/(I_3 + I_2)| < \varepsilon$, положить $I = I_3 + I_2$ и перейти к этапу 4.

3. Положить $I_2 = I_2 + I_1''$, $I_1 = I_1'$, $a = a/2$. Перейти к этапу 1.

4. Конец алгоритма.

Для интегрирования на полуоси $[0, \infty)$ с показательным весом применяются квадратуры Гаусса—Лагерра:

$$\int_0^\infty e^{-x} f(x)\,dx = \sum_{j=1}^k A_j f(x_j).$$

Предполагается, что экспонента *выделена из интегрируемой функции*. Узлами данной формулы служат корни многочлена Лагерра

$$L_n(x) = (-1)^n e^x \frac{d^n}{dx^n}(x^n e^{-x}).$$

Интегралы указанного вида приходится вычислять при решении задач теории очередей, а также при выполнении преобразования Лапласа.

Приведем фрагмент программы численного интегрирования на полуоси функции $f(t)$ с особенностью в нуле. Здесь интеграл $z(1)$ сначала вычисляется на интервале [0,1] по квадратурной формуле Гаусса с восемью узлами. В дальнейшем «гауссов» интервал делится пополам. Для его левой части интеграл $z(1)$ снова вычисляется по Гауссу, для правой $z(2)$ — по составной квадратурной формуле Симпсона с последовательным делением шага пополам и применением правил Ромберга для оценки текущей погрешности и поправки. «Симпсонов» интеграл *накапливается* в переменной $z(4)$ — без пересчета ранее обработанных подынтервалов. Текущее значение интеграла равно $z(4)+z(1)$ и сравнивается с обновляемой в каждом цикле предыдущей оценкой $z(5)$.

При стабилизации упомянутой суммы к ней добавляются интегралы по очередным интервалам единичной длины, вычисляемые по Симпсону—Ромбергу. Процесс заканчивается при достаточной относительной малости уточнения.

```
.....................................................
  z(1)=0; z(4)=0; z(5)=1.0   ! Начальные суммы
  a=0; c=1.0      ! Начальные границы левого интервала
! Интегрирование на нем
  do while (abs(z(1)+z(4)-z(5)).gt.eps)
     z(5)=z(4)+z(1);
     b=c;         ! Правая граница := старая середина
     c=(a+b)/2    ! Новая середина
     sum=0        ! Интегральная сумма
     do i=1,8
        t=(1d0+pt(i))*c/2  ! Аргумент функции
        sum=sum+f(t)*wt(i) ! Накопление интегральной суммы
     end do
     z(1)=sum*c/2          ! Левая половина  - по Гауссу
     as=c; bs=b            ! Границы добавления к правой
     call simpromb(as,bs,f,eps,z(2)) ! По Симпсону
     z(4)=z(4)+z(2)
  end do                   ! Конец интегрирования на [0,1]

  z(4)=z(4)+z(1)           ! Конец для начального участка

  ds=1d0; bs=1d0
  do while (abs(ds).gt.eps)
      as=bs; bs=bs+1d0        ! Пересчет границ для Симпсона
```

```
    call simpromb(as,bs,f,eps,z(2)) ! Интегр. по Симпсону
    ds=abs(z(2)/z(4))       ! Относит. глобальное уточнение
    z(4)=z(4)+z(2)          ! Накопление интеграла
end do
q0=z(4)                     ! Результат
```

Техника эффективного интегрирования на отрезке по Симпсону—Ромбергу обсуждалась в разд. 12.8.2. Переменные `z(1)`, `z(2)`, `z(4)` представляют *суммы* ординат, входящих в квадратурную формулу с соответствующим весом — сами ординаты не запоминаются, что уменьшает потребную память. Отметим также, что никакие ординаты повторно не вычисляются.

12.11. Интегрирование осциллирующих функций

Пусть необходимо вычислить интеграл вида

$$I = \int_a^b f(x)\sin(\omega x + \alpha)\,dx, \tag{12.11.1}$$

причем ω велико. Ясно, что интеграл будет мал, поскольку его положительные и отрицательные составляющие в значительной степени нейтрализуются. Произведя интегрирование по частям, находим

$$I = \frac{1}{\omega}[f(a)\cos(\omega a + \alpha) - f(b)\cos(\omega b + \alpha)] + \frac{1}{\omega}\int_a^b f'(x)\cos(\omega x + \alpha)\,dx. \tag{12.11.2}$$

Полученный интеграл имеет вид (12.11.1), а потому второе слагаемое имеет порядок малости выше $1/\omega$. Первое слагаемое дает оценку искомого интеграла. Оно может быть записано через подынтегральную функцию. Тогда

$$I \approx \frac{1}{\omega^2}[F'(a) - F'(b)].$$

Дальнейшие уточнения см. [10, с. 116] и [29, 80].

12.12. Выбор метода и шага интегрирования

Выбор квадратурной формулы зависит от многих обстоятельств и прежде всего — от вида подынтегральной функции и выбранного шага. Любая попытка сравнения методов численного интегрирования ставит вопрос типа «Что больше — h^2y'' или h^4y^{IV}?» Ответ может быть только один: все зависит от вида функции и ее производных.

Для любой квадратурной формулы существуют функции, ошибка интегрирования которых будет сколь угодно велика (пример: квадрат многочлена с корнями в узлах квадратурной формулы, умноженный на достаточно большую постоянную). Таким образом, *выбор большого числа узлов еще не гарантирует высокой точности интегрирования.*

Непосредственное определение погрешности интегрирования по формулам, приведенным в разд. 12.4–12.6, требует оценки модуля производной высокого порядка. Если получение такой оценки затруднено, можно составить таблицу конечных разностей подынтегральной функции в узлах элементарного участка интегрирования. Практическое постоянство разностей k-го порядка свидетельствует о возможности замены подынтегральной функции полиномом k-й степени и гарантирует хорошую точность в случае применения квадратурной формулы Котеса k-го порядка. Обычно искомый интеграл I вычисляют по выбранной формуле дважды: с шагами h_1 и $h_2 = h_1/2$, после чего применяют правило Рунге.

Количество и размещение узлов квадратурной формулы выбираются с учетом числа и положения экстремумов подынтегральной функции. При фиксированном числе узлов наивысшую точность обычно дают формулы Гаусса, а затем — формулы Чебышева. Покажем это на примере.

Пример 12.3. Пусть требуется вычислить интеграл

$$I = \int_0^{\infty} \sqrt{2+x}\,dx = \frac{2}{3}\sqrt{2+x}\Big|_{-1}^{1} = \frac{2}{3}[\sqrt{3^3}-\sqrt{1^3}] = 2[\sqrt{3}-1/3] = 2.797435,$$

пользуясь различными формулами при $k = 2$. Формула Симпсона дает

$$I \approx \frac{1}{3}[\sqrt{2-1} + 4\sqrt{2+0} + \sqrt{2+1}] = 2.796302.$$

Формула Чебышева приводит к значению интеграла

$$I \approx \frac{2}{3}\left[\sqrt{2-\sqrt{2}/2} + \sqrt{2+0} + \sqrt{2+\sqrt{2}/2}\right]$$

$$= \frac{1}{3}\left[\sqrt{8-2\sqrt{2}}+2\sqrt{2}+\sqrt{8+2\sqrt{2}}\right] = 2.797731;$$

по формуле Гаусса получаем

$$I \approx \frac{1}{9}\left[5\sqrt{2-\sqrt{3/5}}+\sqrt{2\sqrt{3/5}}+8\sqrt{2+0}\right] = 2.797463.$$

Сравнение полученных приближенных значений с точным значением интеграла показывает, что формула Гаусса дает 5 верных знаков, формула Чебышева — 4, а формула Симпсона — только 3.

При сохранении класса и порядка формулы интерполяционных квадратур дальнейшее повышение точности обычно может быть достигнуто применением составных формул. Имеет смысл на участках, где подынтегральная функция колеблется сильнее, сосредоточивать большее количество узлов и накрывать их более короткими отрезками.

С другой стороны, повышение точности может быть достигнуто без дополнительного разбиения — повышением порядка формулы. Возникает вопрос о том, какой из этих способов (или какое сочетание их) выгоднее при одинаковом общем числе узлов. Допустим, что интервал $[a,b]$ разбит на 6 равных частей. Для вычисления интеграла по этому интервалу его можно считать составленным из трех интервалов длиной $2h$ и к каждому из них применить формулу Симпсона либо иметь дело с двумя интервалами длиной $3h$ и использовать формулу «трех восьмых». Вычисления $\int\limits_0^{\pi/2} \sin x\,dx = -\cos x|_0^{\pi/2} = 1$ с шагом $h = \pi/12$ дают для первого способа 1.000026, а для второго 1.000060. Таким образом, ошибка для $k = 2$ оказалась вдвое меньше.

Как правило, уменьшение длины частичного интервала оказывается эффективнее перехода к более точной квадратурной формуле.

Недостатком формул Чебышева и Гаусса является невозможность использования ранее вычисленных ординат при автоматическом уменьшении шага. В то же время формулы Котеса с четным k позволяют при каждом новом значении h вычислять только половину ординат; остальные берутся из ранее выполненного расчета при двойном шаге. Напомним, что нет никакой необходимости запоминать сами ординаты: достаточно передавать от шага к шагу суммы ординат, входящих в составные квадратурные формулы с одинаковыми весами.

Описанная выше реализация составных формул может порождать избыточные вычисления на участках достаточной гладкости подынте-

гральной функции. Современные квадратурные программы [37] применяют *адаптивный* алгоритм интегрирования. Интегрирование выполняется по отдельным участкам. Пока суммарная оценка погрешности превосходит допуск, подотрезок, имеющий наибольшую суммарную оценку погрешности, делится пополам и формируются частичные интегралы и оценки погрешности. Предельное число подотрезков задается параметром процедуры интегрирования.

12.13. Понятие о кубатурных формулах

Кубатурные формулы применяются для приближенного вычисления *кратных* интегралов. Пусть необходимо вычислить двойной интеграл от функции $f(x,y)$ по области, ограниченной графиками двух непрерывных однозначных функций и двумя вертикалями $x=a$ и $x=b$ (рис. 12.4). Тогда

$$I = \iint\limits_D f(x,y)\,dxdy = \int\limits_a^b dx \int\limits_{\psi(x)}^{\varphi(x)} f(x,y)dy.$$

Внутренний интеграл может рассматриваться как функция x:

$$F(x_i) \approx \sum_{j=1}^{m_i} B_{ij} f(x_i, y_j).$$

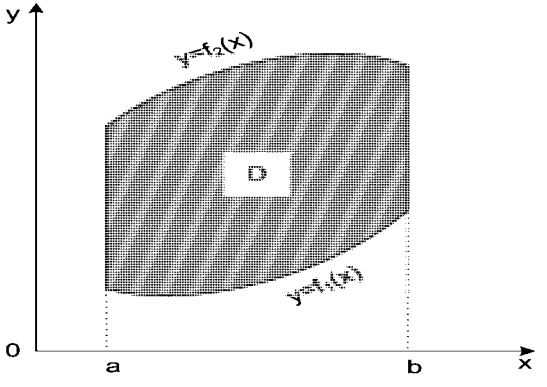

Рис. 12.4. К вычислению двойного интеграла

Применяя одну из квадратурных формул вида (12.3.2) и к внешнему интегралу, получим

$$\int\int_D f(x,y)\,dxdy \approx \sum_{i=1}^{n} A_i \sum_{j=1}^{m_i} B_{ij} f(x_i, y_j). \qquad (12.13.1)$$

Аналогично вычисляются интегралы более высокой кратности. Если внутренний интеграл вычисляется неточно, то для внешнего интегрирования он будет представляться пилообразной функцией с соответствующим поведением производных. Поэтому он должен вычисляться на порядок точнее внешнего.

Для частных случаев областей интегрирования (прямоугольник, круг) существуют прямые обобщения рассмотренных выше формул однократного интегрирования, сводящиеся к однократному суммированию. При двумерном интегрировании также эффективна глобальная адаптивная стратегия. Здесь вместо одномерных отрезков рассматриваются треугольники, а деление отрезков пополам заменяется разбиением на подтреугольники.

12.14. Интегрирование по методу Монте-Карло

Идея вычисления интеграла $\int_a^b f(x)\,dx$ методом Монте-Карло в простейшем случае состоит в том, что график интегрируемой функции мажорируется прямой $y = m$. Далее для интервала интегрирования $[a,b]$ генерируются N случайных точек с равномерно распределенными координатами $x \in [a,b]$ и $y \in [0,m]$. Если $y < f(x)$, то к счетчику z добавляется единица. Оценка значения интеграла дается формулой $m(b-a)\cdot z/N$.

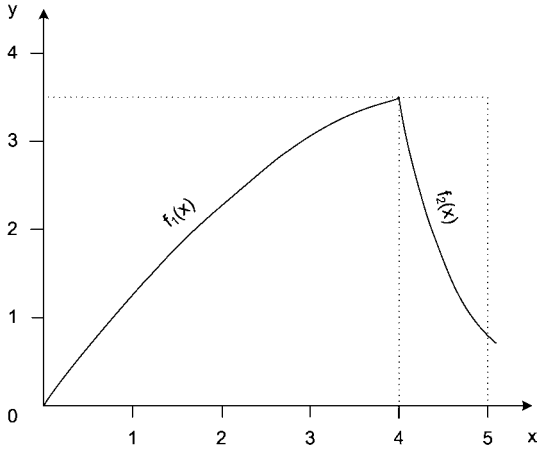

Рис. 12.5. К вычислению определенного интеграла методом Монте-Карло

Пусть необходимо вычислить интеграл от кусочно-гладкой функции $f(x)$, которая на отрезке [0,4] описывается формулой $f_1(x) = 5(1 - e^{-0.3x})$, а при $4 < x \leq 5$ — зависимостью $f_2(x) = me^{-1.5(x-4)}$. График этой функции показан на рис. 12.5. Обе эти функции задаются внутренними процедурами, причем вторая «для интереса» считается с помощью степенного ряда. Множитель перед экспонентой $m = 3.494029$ выбран так, чтобы обеспечить отсутствие разрыва при $x = 4$. Точное значение интеграла равно 9.482537. Приведем программу расчета:

```
program Monte_Carlo
   real,    parameter  :: eps=1e-7, p=0.3, q=1.5
   integer, parameter  :: N=1e6     !! Миллион точек
   real                   m,smaj,sint,rand,x,y,z
   integer                j
   m=5*(1-exp(-4.0*p))
   smaj=4.0*m          !! Площадь объемлющего прямоугольника
   sint=0
   do j=1,N
           x=1.0+4.0*rand()
           y=m*rand()
           if (x<4.0) then
                   z=f1(x)
           else
                   z=f2(x)
           end if
```

```
            if (y<z) sint=sint+1.0
   end do
   z=smaj*sint/N
   print *, "Оценка значения интеграла есть ", z

contains
   real function f1(x)
      real x
      f1=5.0*(1.0-exp(-p*x))
   end function f1

   real function f2(x)
      real x,s,b,k,y
      y=(t-4.0)*q
      b=1.0; s=b; k=1.0
      do while(abs(b)>eps)
               b=-b*y/k; s=s+b; k=k+1.0
      end do
      f2=m*s
   end function f2
end
```

Найденное значение составило 9.489867.

Метод Монте-Карло применим почти для любой функции и любой области — если есть удобный способ выбора случайных точек в области. Он особенно эффективен при вычислении многомерных интегралов по областям сложной формы и с успехом используется для моделирования различных физических явлений (его первое применение было связано с исследованием в рамках Манхэттенского проекта процесса размножения нейтронов). Погрешность метода убывает как $N^{-1/2}$.

Глава 13. Численное интегрирование дифференциальных уравнений

Изложение содержания данной главы мы начнем цитатой из книги В. Э. Милна «Численное решение обыкновенных дифференциальных уравнений»:

«В том случае, когда научная или техническая проблема допускает математическую формулировку, наиболее вероятно, что задача сводится к одному или нескольким дифференциальным уравнениям. Это заведомо верно для обширного класса проблем, связанных с силой и движением. Хотим ли мы знать будущий путь Юпитера на небе или траекторию электрона в электронном микроскопе, мы прибегаем к дифференциальным уравнениям. Это же верно в отношении изучаемых явлений в непрерывной среде, распространения колебаний, передачи тепла, диффузии — с той лишь разницей, что здесь мы имеем дело с дифференциальными уравнениями в частных производных.

Элементарные курсы дифференциальных уравнений преподносят длинный перечень искусных приемов, посредством которых предполагается решать дифференциальные уравнения. Но неизбежно появляется неприятное чувство разочарования и безнадежности, когда в конечном счете выясняется, как сравнительно мало имеется дифференциальных уравнений, которые можно проинтегрировать этими приемами, и как много проблем, которые можно точно сформулировать в виде дифференциальных уравнений, но для которых ничего большего сделать нельзя, так как решение полученных уравнений неизвестно».

Поэтому все большее применение находят *численные* методы интегрирования дифференциальных уравнений, к описанию которых мы и переходим.

В данной главе, в отличие от прочих, буквой Δ по традиции обозначается не погрешность вычисления, а приращение функции.

13.1. Постановка задачи Коши

Задачей Коши для дифференциального уравнения первого порядка

$$y' = f(x, y) \tag{13.1.1}$$

называется задача отыскания решения уравнения (13.1.1), удовлетворяющего заданному начальному условию

$$y|_{x=x_0} = y_0, \tag{13.1.2}$$

где x_0, y_0 — заданные числа. Уравнению (13.1.1) на плоскости $x0y$ отвечает *поле направлений*, а графиком решений служат интегральные кривые, касающиеся направлений поля в каждой своей точке (рис. 13.1).

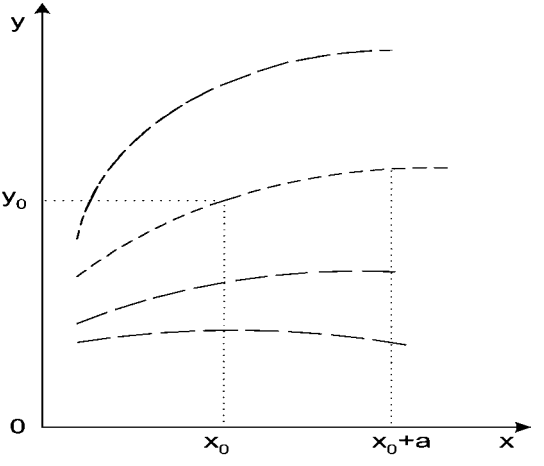

Рис. 13.1. К постановке задачи Коши

Дифференциальное уравнение описывает, как изменяется функция. Это позволяет моделировать движения и процессы, непрерывно меняющиеся во времени. Подобные процессы можно реализовать «электротехнически» на аналоговых вычислительных машинах и наблюдать на

экране осциллографа поведение решений исходных дифференциальных уравнений. В частности, чрезвычайно поучительно рассмотрение *фазовой траектории* — в осях «положение — скорость», причем независимая переменная (время) играет роль параметра. Изменяя коэффициенты уравнений, можно менять поведение решения (например, подобрать оптимальные параметры контура управления).

Геометрический смысл задачи Коши состоит в построении интегральной кривой, проходящей через заданную точку (x_0, y_0). В тех редких случаях, когда удается найти общее решение уравнения (13.1.1), задача Коши сводится к отысканию входящего в нее значения произвольной постоянной и подстановке ее в общее решение. Обычно приходится решать задачу Коши приближенно.

Существует несколько источников возникновения ошибок при решении дифференциальных уравнений. Так, ошибки появляются при дискретизации дифференциальной задачи. Во многих случаях повторное решение задачи на более частой сетке позволяет добиться большей точности. Однако иногда малые ошибки, внесенные в начале вычислений, могут совершенно исказить решение. Это явление называется *неустойчивостью* метода. Уменьшение шага увеличивает трудоемкость решения задачи и накопленную ошибку округлений.

Приближенные методы решения задачи Коши можно грубо разделить на два класса: аналитические и численные. В аналитических методах приближенное решение разыскивается в виде функции (многочлен, тригонометрический многочлен и т. д.). Приближенные методы используют различные упрощения самих уравнений путем обоснованного отбрасывания некоторых их членов, а также поиска решений в ограниченном классе функций. Широко используется разложение решения в ряд по степеням входящего в постановку задачи малого параметра, а также асимптотические методы.

В численных методах решение представляется таблицей приближенных его значений для ряда значений аргумента. По этой таблице могут быть построены (при современных программных средствах — автоматически) графики.

Задача Коши для уравнений порядка выше первого сводится к решению задачи Коши для системы уравнений первого порядка. Методы решения последней по существу не отличаются от методов решения задачи Коши для одного уравнения. Поэтому излагаемый ниже материал без труда переносится на уравнения высшего порядка и системы уравнений первого порядка.

13.2. Представление решения задачи Коши в виде степенного ряда

Предположим для простоты, что $x_0 = 0$, чего всегда можно добиться введением новой переменной $t = x - x_0$. Считая, что такая замена произведена, вместо t будем писать x. Представим искомое решение рядом Маклорена

$$y = y_0 + a_1 x + a_2 x^2 + \dots \qquad (13.2.1)$$

и укажем способ нахождения коэффициентов ряда. Очевидно,

$$a_k = \frac{1}{k!}\, y^{(k)}\big|_{x=0}.$$

В частности,

$$a_1 = \frac{1}{1!}\, y'|_{x=0} = \frac{1}{1!}\, f(x,y)|_{x=0} = \frac{1}{1!} f(0,y_0) = f(0,y_0).$$

Далее, $a_2 = \frac{1}{2!}\, y''|_{x=0}$, и для определения a_2 нужно найти y''. Дифференцируя обе части равенства (13.1.1), получим

$$y'' = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} = f'_x(x,y) + f'_y(x,y)y'. \qquad (13.2.2)$$

Следовательно,

$$a_2 = \frac{1}{2!}\, y''|_{x=0} = \frac{1}{2!}[f'_x(0,y_0) + f'_y(0,y_0)f(0,y_0)].$$

Дифференцированием (13.2.2) находим выражение для y''' и затем для a_3 и т. д. Этот процесс при условии, что $f(x,y)$ имеет частные производные любого порядка, можно продолжать сколь угодно далеко. При довольно широких предположениях об $f(x,y)$ доказывается сходимость ряда (13.2.1) в некотором промежутке $(-a,a)$ к решению исходной задачи Коши. Усеченная сумма ряда (13.2.1) будет приближенным ее решением — тем более точным, чем меньше $|x|$.

Пример 13.1. Пусть $y' = x^2 + y^2$, $y(0) = 1$. Тогда $y'(0) = 0^2 + 1^2 = 1$. Дифференцируя, находим

$$y'' = 2x + 2yy'$$

и $y''(0) = 2 \cdot 0 + 2 \cdot 1 \cdot 1 = 2$. Следовательно, $a_2 = \frac{1}{2!} \cdot y''|_{x=0} = 1$.

Третья производная

$$y''' = 2 + 2(y')^2 + 2yy''; \qquad y'''|_{x=0} = 2 + 2 \cdot 1^2 + 2 \cdot 1 \cdot 2 = 8;$$

значит, $a_3 = 8/3! = 4/3$.

Четвертая производная

$$y^{(4)} = 6y'y'' + 2yy'''; \qquad y^{(4)}\big|_{x=0} = 28; \quad a_4 = \frac{28}{4!} = \frac{7}{6}.$$

Итак,

$$y \approx 1 + x + x^2 + \frac{4}{3}x^3 + \frac{7}{6}x^4.$$

Для $|x| \le 0.1$ ошибка этого представления не превосходит 0.0002.

Разложением в ряд Маклорена можно пользоваться для нескольких первых значений функции в точках $h, 2h, \ldots$ при малых значениях шага h. Это потребуется нам в дальнейшем.

13.3. Идея численных методов решения задачи Коши

Численные методы дают решение задачи Коши в виде таблицы приближенных значений искомой функции $y(x)$ для ряда значений аргумента (узлов), обычно выбираемых равноотстоящими. Эти узлы будем обозначать через $x_0, x_1, \ldots, x_k, \ldots$, а соответствующие приближенные значения через $y(x_k) \approx y_k$; стартовая точка (x_0, y_0) задается начальными условиями.

Уравнение называется *устойчивым*, если кривые из семейства решений сходятся (сближаются) с ростом x, и неустойчивым в противном случае. Если уравнение неустойчиво, то ошибки имеют тенденцию к росту. Устойчивые уравнения подавляют ошибки.

Пусть $\tilde{y}_k(x)$ есть точное решение уравнения (13.1.1), удовлетворяющее условию $\tilde{y}_k(x_k) = y_k$. Через $\bar{\Delta} y_k$ обозначим приращение $\tilde{y}_k(x)$ при переходе от x_k к x_{k+1}. Так как $\tilde{y}'(x) \equiv f(x, \tilde{y}_k(x))$, то, полагая $h_k = x_{k+1} - x_k$, имеем:

$$\bar{\Delta} \stackrel{\text{def}}{=} \tilde{y}_k(x_{k+1}) - y_k = \int\limits_{x_k}^{x_k + h_k} f(x, \tilde{y}_k(x))\, dx. \qquad (13.3.1)$$

Если бы мы умели точно находить отдельные части равенства (13.3.1), то были бы в состоянии составить таблицу точных значений искомой функции, отправляясь от y_0 и находя последовательно $\bar{\Delta}y_0$; $y_1 = \tilde{y}(x_1) = y_0 + \bar{\Delta}y_0$; $\bar{\Delta}y_1$; $y_2 = y_1 + \bar{\Delta}y_1$ и т. д.

Рассматриваемые ниже различные методы численного интегрирования отличаются друг от друга подходом к вычислению Δy_k — приближенного значения для $\bar{\Delta}y_k$. Если указан прием для вычисления Δy_k, то расчет y_{k+1} ведется по формуле

$$y_{k+1} = y_k + \Delta y_k, \qquad k = 0, 1, 2, \ldots \tag{13.3.2}$$

В последующем иногда вместо x_k, y_k, h_k, $\bar{\Delta}y_k$, Δy_k для простоты будем писать $\bar{x}$, $\bar{y}$, h, $\bar{\Delta}y$, Δy. В предположении ограниченности $f(x, y)$ в окрестности точки $(\bar{x}, \bar{y})$ при $h \to 0$ величина $\bar{\Delta}y$ есть бесконечно малая по меньшей мере первого порядка относительно h. Естественно требовать того же от Δy. Тогда погрешность приращения $\bar{\Delta}y - \Delta y$ есть бесконечно малая порядка не ниже первого.

Число l назовем *порядком пошаговой погрешности* численного метода, если существует такое число $c \neq 0$, зависящее от значений $f(x, y)$ и ее частных производных в точке $(\bar{x}, \bar{y})$, что

$$\lim_{h \to 0} \frac{\bar{\Delta}y - \Delta y}{h^l} = c. \tag{13.3.3}$$

Порядок пошаговой погрешности — одна из важнейших характеристик численного метода. Локальная ошибка усечения совпадает с остаточным членом формулы Тейлора.

При решении задачи Коши с постоянным шагом h на промежутке $[x_0, X]$ число шагов $n = (X - x)/h$, и порядок малости суммарной погрешности в точке X на единицу меньше порядка пошаговой погрешности, т. е. равен $l - 1$.

Происхождение *локальных* ошибок при решении дифференциальных уравнений аналогично разностной аппроксимации производной. Если локальная ошибка может быть записана в виде $O(h^{p+1})$, то говорят о методе p-го порядка. В соответствии с принципом Рунге повторный счет от точки t_0 до T с шагом $h' < h$ может сократить ошибку в точке T примерно в $(h'/h)^p$ раз.

Глобальная ошибка получается суммированием локальной ошибки текущего шага с глобальной ошибкой на предыдущем шаге, умноженной на пересчетный множитель. Последний зависит от якобиана системы J и выбранного шага интегрирования и должен быть по модулю меньше

единицы. Поэтому численный метод может быть неустойчивым даже для устойчивых уравнений — в зависимости от размеров шага.

Стратегии выбора шага должны опираться на теорию, численный эксперимент и интуицию. Последнюю надо наработать!

13.4. Метод Эйлера

Простейшим из численных методов интегрирования дифференциальных уравнений является метод Эйлера. Он основан на замене неизвестной подынтегральной функции $f(x, \tilde{y}_k(x))$ в интеграле (13.3.1) на единственное известное ее значение в точке x_k: $f(x_k, y_k)$. Таким образом,

$$\bar{\Delta} y_k \approx \Delta y_k = y_{k+1} - y_k = h_k f(x_k, y_k),$$

что приводит к расчетной формуле метода Эйлера

$$y_{k+1} = y_k + h_k f(x_k, y_k), \qquad k = 0, 1, \ldots \tag{13.4.1}$$

Для метода Эйлера порядок пошаговой погрешности $l = 2$. Следовательно, порядок погрешности на всем интервале равен $l - 1 = 1$. При переходе от t_k к t_{k+1} накопленная глобальная ошибка умножается на $(1 - h_k J)$. Если этот множитель больше 1, то ошибки будут возрастать. Этот подход естественно обобщается на системы уравнений [37, с. 343].

В *модифицированном* методе Эйлера выполняется усреднение значения производной в начале и в конце очередного интервала (в конце — вычисленное приближенно).

Пример 13.2. Пусть дано дифференциальное уравнение $y' = -y$ и начальное условие $y(0) = 1$. Известно точное решение задачи $y = e^{-x}$.

Подставив в (13.4.1) $f(x, y) = -y$, получим

$$y_{k+1} = y_k + h(-y_k) = y_k(1 - h), \qquad k = 0, 1, 2, \ldots;$$

следовательно, с учетом $y_0 = 1$ имеем

$$y_k = (1 - h)^k.$$

Пусть даны $X > 0$ и n — целое положительное число. Полагая $h = X/n$ и обозначая через $\tilde{y}(x)$ приближенное решение задачи, найдем

$$\tilde{y}(X) = (1 - h)^n = (1 - h)^{X/n}.$$

Отсюда следует

$$\begin{aligned}\ln y(X) &= \frac{X}{n}\ln(1-h) = \frac{X}{h}\left(-h-\frac{h^2}{2}-\frac{h^3}{3}-\dots\right)\\ &= -X - X\left(h+\frac{h^2}{2}+\frac{h^3}{3}+\dots\right).\end{aligned}$$

Учитывая, что $\ln y(X) = \ln e^{-X} = -X$, находим

$$\ln y(X) - \ln\tilde{y}(X) = X\left(h+\frac{h}{2}+\frac{h^2}{3}+\dots\right),$$

и поэтому

$$\lim_{h\to 0}\frac{y(X)-\tilde{y}(X)}{h} = \frac{X}{2}e^{-X} \neq 0,$$

что подтверждает установленный ранее порядок погрешности метода Эйлера.

Недостатком исправленного метода Эйлера и других методов более высоких порядков, основанных на пошаговом представлении решения задачи Коши по формуле Тейлора и последовательном дифференцировании уравнения (13.1.1) для получения тейлоровых коэффициентов, является необходимость вычисления на каждом шаге частных производных функции $f(x,y)$.

13.5. Методы Рунге—Кутты

Идея построения явных методов Рунге—Кутты p-го порядка заключается в получении приближений к значениям $f(x_{i+1})$ по формуле вида

$$y_{i+1} = y_i + h\varphi(x_i, y_i, h),$$

где $\varphi(x,y,h)$ — некоторая функция, приближающая отрезок ряда Тейлора до p-го порядка и не содержащая частных производных функции $f(x,y)$. Функцию $\varphi(x,y,h)$ берут многопараметрической и подбирают ее параметры сравнением с нужной степени многочленом Тейлора для $y(x)$.

Для простоты вместо x_k, y_k, h_k будем писать $\bar{x}, \bar{y}, h$. Пусть $r \geq 2$ есть целое число; $\{\alpha_1, \alpha_2, \dots, \alpha_{r-1}\}$ — положительные числа;

пусть числа $\{\beta_{sm}\}$, $s = \overline{1, r-1}$; $m = \overline{1, s}$ и $\gamma_1, \gamma_2, \ldots, \gamma_r$ удовлетворяют условиям

$$\beta_{11} = \alpha_1; \quad \sum_{m}^{s} \beta_{sm} = \alpha_s; \quad \sum_{j=1}^{r} \gamma_j = 1. \tag{13.5.1}$$

Один этап метода Рунге—Кутты (переход от x_k к x_{k+1}) осуществляется следующим образом:

1) последовательно одно за другим вычисляются следующие r чисел:

$$\begin{array}{lcl}
\Delta_1 & = & hf(\bar{x}, \bar{y}); \\
\Delta_2 & = & hf(\bar{x} + \alpha_1 h, \bar{y} + \beta_{11}\Delta_1); \\
\Delta_3 & = & hf(\bar{x} + \alpha_2 h, \bar{y} + \beta_{21}\Delta_1 + \beta_{22}\Delta_2); \\
\ldots\ldots & \ldots\ldots & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\Delta_{j+1} & = & hf(\bar{x} + \alpha_j h, \bar{y} + \beta_{j1}\Delta_1 + \ldots + \beta_{jj}\Delta_j); \\
\ldots\ldots & \ldots\ldots & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\Delta_r & = & hf(\bar{x} + \alpha_{r-1} h, \bar{y} + \beta_{r-1,1}\Delta_1 + \ldots + \beta_{r-1,r-1}\Delta_{r-1});
\end{array} \tag{13.5.2}$$

2) вычисляется

$$\Delta = \sum_{j=1}^{r} \gamma_j \Delta_j; \tag{13.5.3}$$

3) находится ордината очередной точки

$$y_{k+1} = \bar{y} + \Delta = y_k + \Delta. \tag{13.5.4}$$

Числа $\{\alpha_s, \beta_{sm}, \gamma_j\}$ при заданном r выбираются так, чтобы разность

$$\bar{\Delta}\bar{y} - \Delta\bar{y} = \int_{\bar{x}}^{\bar{x}+h} f(x, \tilde{y}(x))\, dx - \Delta \equiv [\tilde{y}(h) - \bar{y}] - \Delta, \tag{13.5.5}$$

где $\tilde{y}(x)$ — точное решение задачи Коши $y' = f(x, y)$, $y|_{x=\bar{x}} = \bar{y}$, рассматриваемая как функция переменной h, была бесконечно малой возможно более высокого порядка l относительно h. Числа $\{\alpha, \beta, \gamma\}$ этим требованием не определяются однозначно, и при их выборе стремятся к простоте формул вида (13.5.2), (13.5.3). Вычисления по таким формулам называют методом *типа* Рунге—Кутты.

Рассмотрим вывод конкретного вида этих формул для простейшего случая $r = 2$. Ниже символы f, f'_x, f'_y, f''_{xy} и т. д. без указания аргумента будут означать значения соответствующих функций в точке $(\bar{x}, \bar{y})$, а символы y_0, y'_0, y''_0 — значения решения и его производных в точке $\bar{x}$. Имеем

$$y_0 = \bar{y}; \quad y'_0 = f(\bar{x}, \bar{y}) = f; \quad y''(x) = f'_x + y' f'_y,$$

и поэтому $y''_0 = f'_x + f f'_y$. Далее,

$$y''' = f''_{xx} + y' f''_{xy} + y'(f''_{x,y} + f''_{y,y} y') + y'' f'_y,$$

откуда

$$y'''_0 = f''_{xx} + 2f f''_{xy} + f^2 f''_{yy} + f'_y(f'_x + f f'_y).$$

Тогда

$$\tilde{y}(h) - \bar{y} = fh + \frac{f'_x + f f'_y}{2!} h^2 + \frac{f''_{xx} + 2f f''_{xy} + f^2 f''_{yy} + f'_y(f'_x + f f'_y)}{3!} h^3 + O(h^4). \tag{13.5.6}$$

С другой стороны, система (13.5.2) при $r = 2$ с учетом $\beta_{1,1} = \alpha_1$ дает

$$\begin{aligned}
\Delta_1 &= hf(\bar{x}, \bar{y}) = hf; \\
\Delta_2 &= hf(\bar{x} + \alpha_1 h, \bar{y} + \alpha_1 \Delta_1) = hf(\bar{x} + \alpha_1 h, \bar{y} + \alpha_1 hf) \\
&= hf + \alpha_1 h^2 [f'_x + f f'_y] \\
&+ \frac{h^3}{2} [f''_{xx} \alpha_1^2 + 2f \alpha_1^2 f''_{xy} + \alpha_1^2 f^2 f''_{yy}] + O(h^4).
\end{aligned}$$

Далее находим

$$\begin{aligned}
\Delta &= \gamma_1 \Delta_1 + \gamma_2 \Delta_2 = hf(\gamma_1 + \gamma_2) + \alpha_1 \gamma_2 h^2 (f'_x + f f'_y) \\
&+ \frac{\gamma_2 h^3 \alpha_1^2}{2} (f''_{xx} + 2f f''_{xy} + f^2 f''_{yy}) + O(h^4).
\end{aligned}$$

Учитывая (13.5.6), убеждаемся, что с точностью $O(h^4)$

$$\begin{aligned}
\tilde{y}(h) - \bar{y} - \Delta &= hf(1 - (\gamma_1 + \gamma_2)) + h^2 (f'_x + f f'_y)(1/2 - \alpha_1 \gamma_2) \\
&+ h^3 \left[(f''_{xx} + 2f f''_{xy} + f^2 f''_{yy}) (1/6 - \alpha_1^2 \gamma_2 / 2) \right. \\
&+ \left. f'_y (f'_x + f f'_y)/6) \right].
\end{aligned} \tag{13.5.7}$$

Следовательно, если потребовать

$$\gamma_1 + \gamma_2 = 1; \quad \alpha_1 \gamma_2 = 1/2, \tag{13.5.8}$$

то будем иметь

$$\tilde{y}(h) - \bar{y} - \Delta = O(h^3),$$

т. е. порядок пошаговой погрешности $l = 3$. Никаким выбором $\alpha_1, \gamma_1, \gamma_2$ нельзя добиться того, чтобы для произвольных $f'_x, f'_y, f''_{xx}, f''_{xy}, f''_{yy}$ коэффициент при h^3 в правой части выражения (13.5.7) обратился в нуль.

Условия (13.5.8) не определяют $\alpha_1, \gamma_1, \gamma_2$ однозначно. Для простоты расчетов обычно полагают $\alpha_1 = 1, \gamma_1 = \gamma_2 = 1/2$, и формулы Рунге—Кутты для $r = 2$ принимают вид

$$\begin{array}{lcl} \Delta_1 & = & hf(\bar{x}, \bar{y}); \\ \Delta_2 & = & hf(\bar{x} + h, \bar{y} + \Delta_1); \\ \Delta & = & (\Delta_1 + \Delta_2)/2; \\ y_{k+1} & = & y_k + \Delta. \end{array} \qquad (13.5.9)$$

При этом $l = 3$.

Геометрическая интерпретация этого метода показана на рис. 13.2.

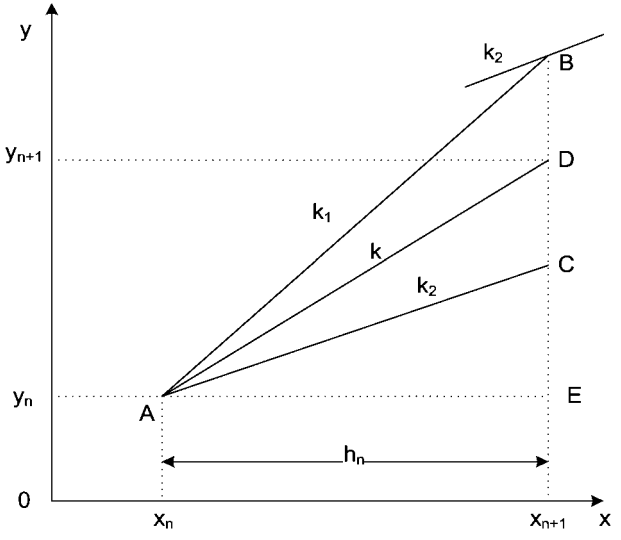

Рис. 13.2. Метод Рунге—Кутты 2-го порядка

Укажем порядок построения точки (x_{k+1}, y_{k+1}):

а) через точку $A(x_k, y_k)$ провести прямую поля направлений до пересечения ее с вертикалью $x = x_{k+1}$ в точке B;

б) через точку B провести прямую поля направлений до пересечения ее в точке C с вертикалью $x = x_{k+1} + h_k = x_k + 2h_k$. Заметим,

что ордината точки C совпадает с ординатой y_{k+2}, полученной по методу Эйлера при условии $h_{k+1} = h_k$;

в) точку A соединить с точкой C; точка D пересечения прямой AC с вертикалью $x = x_{k+1}$ и есть очередная точка, ордината которой принимается за y_{k+1}.

Аналогичным образом строятся методы Рунге—Кутты любого порядка. Обычно используется случай $r = 4$, в котором

$$\begin{aligned}
\Delta_1 &= hf(\bar{x}, \bar{y}); \\
\Delta_2 &= hf(\bar{x} + h/2, \bar{y} + \Delta_1/2); \\
\Delta_3 &= hf(\bar{x} + h/2, \bar{y} + \Delta_2/2); \\
\Delta_4 &= hf(\bar{x} + h, \bar{y} + \Delta_3); \\
\Delta &= (\Delta_1 + 2\Delta_2 + 2\Delta_3 + \Delta_4)/6.
\end{aligned}$$

Эта схема отвечает следующему выбору чисел α, β и γ:

$$\begin{aligned}
&\alpha_1 = \alpha_2 = \beta_{1,1} = \beta_{2,2} = 1/2; \quad \alpha_{3,3} = \beta_{3,3} = 1; \\
&\beta_{2,1} = \beta_{3,1} = \beta_{3,2} = 0; \\
&\gamma_1 = \gamma_4 = 1/6; \quad \gamma_2 = \gamma_3 = 1/3.
\end{aligned}$$

Ей соответствует $l = 5$. Говоря о методе Рунге—Кутты, обычно имеют в виду именно этот вариант.

Обычно интегрирование от t_n до t_{n+1} проводится дважды: первый раз с шагом h, второй — двумя шагами $h/2$. Сравнивая две оценки, можно оценить локальную ошибку. Таким образом, при успешно выполненном шаге требуется 11-кратное вычисление правых частей.

Методы Рунге—Кутты имеют значительное число свободных параметров, которые могут использоваться для повышения вычислительной эффективности этих методов. Руководящая идея этих модификаций — получить формулы из данного семейства, которые:

- используют одни и те же значения функции;
- определяют разные версии метода одного или смежных порядков;
- позволяют по разности результатов судить о точности счета.

В варианте Кутты—Мерсона на каждом шаге вычисляют:

$$\eta_1^i = f(x_i, y_i),$$

$$\begin{aligned}
\eta_2^i &= f(x_i + \frac{h}{3}, y_i + \frac{h}{3}\eta_1^i), \\
\eta_3^i &= f(x_i + \frac{h}{3}, y_i + \frac{h}{6}\eta_1^i + \frac{h}{6}\eta_2^i), \\
\eta_4^i &= f(x_i + \frac{h}{2}, y_i + \frac{h}{8}\eta_1^i + \frac{3h}{8}\eta_2^i), \\
\eta_5^i &= f(x_i + h, y_i + \frac{h}{2}\eta_1^i - \frac{3h}{2}\eta_3^i + 2h\eta_4^i), \\
\tilde{y}_{i+1} &= y_i + \frac{h}{2}(\eta_1^i - 3\eta_3^i + 4\eta_4^i), \\
y_{i+1} &= y_i + \frac{h}{6}(\eta_1^i + 4\eta_4^i + \eta_5^i).
\end{aligned}$$

После этого подсчитывают величину

$$R = 0.2|y_{i+1} - \tilde{y}_{i+1}|$$

и проводят сравнения. Если значение R окажется больше заданного допустимого уровня абсолютных погрешностей ε, то шаг уменьшают вдвое и возвращаются к началу второго этапа. В противном случае полагают $y_{i+1} = \tilde{y}_{i+1}$ с погрешностью не более ε. Если $R \le \varepsilon/64$, то расчет продолжается с шагом $2h$.

Фельберг исходил из шестиэтапного метода Рунге—Кутты и подобрал коэффициенты так, что метод имеет пятый порядок. По четырем значениям k можно, скомбинировав их с другими весами γ, построить неклассический вариант метода Рунге—Кутты и, следовательно, получить вторую оценку решения в точке t_{n+1}. Их разность можно использовать для оценки локальной ошибки решения и длины следующего шага.

Вариант метода с контролем пошаговой погрешности ([10, с. 362]):

$$\begin{aligned}
k_1 &= hf(x,y); \quad k_2 = hf\left(x + \frac{h}{2}, y + \frac{k_1}{2}\right); \\
k_3 &= hf\left(x + \frac{h}{2}, y + \frac{k_1 + k_2}{4}\right); \quad k_4 = hf(x + h, y - k_2 + 2k_3); \\
k_5 &= hf\left(x + \frac{2h}{3}, y + \frac{7k_1 + 10k_2 + k_4}{27}\right);
\end{aligned}$$

$$k_6 = hf\left(x+\frac{h}{5}, y+\frac{28k_1-125k_2+546k_3+54k_4-378k_5}{625}\right);$$
$$\Delta y = \frac{k_1+4k_3+k_4}{6}.$$

Главный член погрешности

$$r = -\frac{42k_1+224k_3+21k_4-162k_5-125k_6}{336}+O(h^6).$$

13.6. Экстраполяционные разностные методы

В методах Эйлера и Рунге—Кутты для вычисления Δy_k использовалось лишь значение y_k. Ниже будут описаны методы, в которых для вычисления Δy_k используются $y_k, y_{k-1}, \ldots, y_{k-r}$ — т. е. значения решения (а, возможно, и производных) в нескольких предшествующих точках. Такие методы называют *многошаговыми*, и при их применении обычно используют равноотстоящие узлы. С этого момента будем считать $x_k = x_0 + kh, \quad k = 1, 2, \ldots$ Через f_k будем обозначать $f(x_k, y_k)$.

Предположим, что помимо y_0 каким-либо способом уже найдены значения $\{y_m\}, \quad m = 1, 2, \ldots, k$, где $k \geq r$. По $(r+1)$ точкам $(x_{k-r}, y_{k-r}), (x_{k-r+1}, y_{k-r+1}), \ldots, (x_k, y_k)$ построим интерполяционный многочлен $I_r(x)$ степени не выше r. Хотя интервал (x_k, x_{k+1}) не содержит узлов интерполяции, все же естественно ожидать, что на нем $f(x, \tilde{y}_k(x)) \approx I_r(x)$, т. е. многочлен $I_r(x)$ *экстраполирует* $f(x, \tilde{y}_k(x))$ на отрезок $[x_k, x_{k+1}]$. Заменив в интеграле (13.3.1) подынтегральную функцию $f(x, \tilde{y}_k(x))$ на $I_r(x)$, приходим к расчетным формулам экстраполяционного метода Адамса порядка r

$$\Delta y_k = \int_{x_k}^{x_k+h} I_r(x)\,dx. \qquad (13.6.1)$$

Рассмотрим различные формы и порядок вычисления правой части формулы (13.6.1).

13.6.1. Разностная форма метода Адамса

Запишем интерполяционный многочлен $I_r(x)$ в форме Ньютона для интерполирования назад и перейдем в нем к новой переменной t по формуле $x = x_k + th$. Получим

$$I_r(x) = f_k + \sum_{j=1}^{r} \frac{\Delta^j f_{k-j}}{j!} t(t+1)\dots(t+j-1).$$

Подставив этот многочлен в (13.6.1) и заменив переменную интегрирования x на t, находим

$$\Delta y_k = h\Big(f_k + \sum_{j=1}^{r} a_j \Delta^j f_{k-j}\Big), \tag{13.6.2}$$

где

$$a_j = \frac{1}{j!} \int_0^1 t(t+1)\dots(t+j-1)\,dt. \tag{13.6.3}$$

Приведем табл. 13.1 коэффициентов экстраполяционной формулы Адамса для $j = \overline{1,5}$.

Таблица 13.1. «Экстраполяционные» коэффициенты (разностная форма)

j	1	2	3	4	5
a_j	1/2	5/12	3/8	251/720	95/288

13.6.2. Безразностная форма метода Адамса

Запишем интерполяционный многочлен $I_r(x)$ в форме Лагранжа и перейдем в нем к новой переменной t по формуле $x = x_k + th$. Получим

$$I_r(x) = \sum_{j=0}^{r} f_{k-j} \frac{(-1)^j}{j!(r-j)!} \prod_{\substack{i=0,\\ i\neq j}}^{r} (t+i),$$

после чего формула (13.6.1) принимает вид

$$\Delta y_k = h \sum_{j=0}^{r} a_j^r f_{k-j}, \tag{13.6.4}$$

где

$$a_j^r = \frac{(-1)^j}{j!(r-j)!} \int_0^1 \Big[\prod_{\substack{i=0,\\ i\neq j}}^{r} (t+i) \Big]\, dt. \qquad (13.6.5)$$

Приведем табл. 13.2 коэффициентов безразностной экстраполяционной формулы Адамса.

Таблица 13.2. «Экстраполяционные» коэффициенты (безразностная форма)

r	a_0	a_1	a_2	a_3
1	3/2	-1/2		
2	23/12	-4/3	5/12	
3	55/24	-59/24	37/24	-3/8

К формуле (13.6.4) можно перейти, заменив в (13.6.2) разности $\{\Delta^j f_{k-j}\}$ их выражениями через $\{f_m\}$.

13.6.3. Порядок вычислений

Начальные условия задачи Коши дают число y_0. Для применения метода Адамса порядка r надо, чтобы были известны также значения $y_1, y_2, \ldots, y_r$. Этот участок таблицы искомых значений называют *разгонным*. Предполагается, что разгонный участок получен каким-либо другим методом (например, методом Рунге—Кутты).

При машинном счете следует пользоваться безразностной формой, ссылающейся непосредственно на вычисленнные значения $\{f_k\}$. Формулы (13.6.4) первый раз применяются для $k = r$, а затем для $k = r+1, r+2$ и т. д.

13.6.4. Пошаговый порядок погрешности метода Адамса

Пусть $\tilde{y}(x)$ есть точное решение уравнения (13.1.1) при условии $\tilde{y}(x_k) = y_k$. Положим $y_{k-j} = \tilde{y}(x_k - jh), \quad j = 1, 2, \ldots r$, и воспользуемся формулой (13.6.1) для вычисления Δy_k. В этих условиях порядок l пошаговой погрешности в смысле формулы (13.3.3) для метода Адамса порядка r равен $r+2$.

Из вышеизложенного следует, что основной метод Рунге—Кутты и метод Адамса 3-го порядка имеют одинаковые порядки пошаговой погрешности.

13.7. Интерполяционные разностные методы (с пересчетом)

Пусть $r \geq 1$ — целое число и известны значения $y_0, y_1, \ldots, y_k$, где $k \geq r$. Хотя еще неизвестно значение y_{k+1}, при построении интерполяционного многочлена для $f(x,y)$ по $r+1$ узлам привлечем и точку (x_{k+1}, f_{k+1}). Итак, обозначим через $\tilde{I}_r(x)$ интерполяционный многочлен по $r+1$ точкам $(x_{k-r+1}, f_{k-r+1}), (x_{k-r+2}, f_{k-r+2}), \ldots, (x_k, f_k), (x_{k+1}, f_{k+1})$ и положим

$$\Delta y_k = \int_{x_k}^{x_k+h} \tilde{I}_r(x)\,dx. \tag{13.7.1}$$

Выбрав форму Лагранжа для многочлена $\tilde{I}_r(x)$ и использовав формулу (13.3.2), найдем безразностную форму интерполяционного метода Адамса порядка r:

$$y_{k+1} = y_k + h\Big[b_{-1}f(x_{k+1}, y_{k+1}) + \sum_{j=0}^{r-1} b_j^r f_{k-j}\Big], \tag{13.7.2}$$

где для $\{b_j^r\}$ можно указать формулы, аналогичные (13.6.5). Приведем значения коэффициентов интерполяционной формулы Адамса $\{b_j^r\}$ для $r =$ 1,2,3 (табл. 13.3).

Таблица 13.3. «Интерполяционные» коэффициенты (безразностная форма)

r	b_{-1}	b_0	b_1	b_2
1	1/2	1/2		
2	5/12	2/3	-1/12	
3	3/8	19/24	-5/24	1/24

Если известны значения $y_{k-r+1}, y_{k-r+2}, \ldots, y_k$, то в формуле (13.7.2) определена лишь сумма $\sum_{j=0}^{r-1} b_j^r f_{k-j}$. Таким образом, формула (13.7.2) содержит неизвестное как в левой части, так и в правой — в слагаемом $b_{-1}f(x_{k+1}, y_{k+1})$.

Если правую часть (13.7.1) обозначить через $\varphi(y_{k+1})$, то формулу (13.7.2) можно рассматривать как *уравнение* вида $y_{k+1} = \varphi(y_{k+1})$, пригодное для итерации. Для достаточно малого шага h это уравнение имеет корень, близкий к y_k, и итерации будут сходиться к этому корню.

Величина последнего и принимается в качестве y_{k+1}. Для получения его с достаточной степенью точности надо проводить итерацию (пересчет) по формуле

$$y_{k+1}^{(m+1)} = y_k + h\left[b_{-1}^r f(x_{k+1}, y_{k+1}^{(m)}) + \sum_{j=0}^{r-1} b_j^r f_{k-j}\right], \quad m = 0, 1, 2, \ldots, \quad (13.7.3)$$

выбрав в качестве начального приближения $y_{k+1}^{(0)}$ — например, значение y_{k+1}, полученное по экстраполяционной формуле Адамса того же порядка. Таким образом, возможна следующая вычислительная схема:

а) сначала используется экстраполяционный метод Адамса для получения $y_{k+1}^{(0)}$; эта часть вычислений называется предсказанием (прогнозом);

б) затем проводится итерация по формуле (13.7.3); эта часть вычислений называется уточнением (коррекцией).

Как правило, для малых значений h ограничиваются одной итерацией, т. е. вычисляют $y_{k+1}^{(1)}$, полагают $y_{k+1} \approx y_{k+1}^{(1)}$ и переходят к вычислению y_{k+2}.

Численные методы, в которых вычисляемое значение y_{n+1} входит в правую часть f, называются *неявными*. Неявные методы как правило устойчивее явных и могут иметь неограниченную область устойчивости. Для них выбор шага определяется соображениями, связанными с локальной ошибкой.

В случае $r = 1$ формулы (13.7.2) и (13.7.3) принимают вид

$$y_{k+1} = y_k + \frac{h}{2}[f(x_k, y_k) + f(x_{k+1}, y_{k+1})]; \quad (13.7.4)$$

$$y_{k+1}^{(m+1)} = y_k + \frac{h}{2}[f(x_k, y_k) + f(x_{k+1}, y_{k+1}^{(m)})]. \quad (13.7.5)$$

Если положить $y_{k+1}^{(0)} = y_k + hf(x_k, y_k)$, т. е. вычислить $y_{k+1}^{(0)}$ по методу Эйлера, то получим метод Эйлера с пересчетом. Эта схема удобнее описанной выше тем, что не требует знания y_{k-1}, а потому не нуждается в разгонном участке (является одношаговой).

Как правило, полученное описанным способом начальное приближение грубее найденного методом Адамса первого порядка, и тогда число итераций, необходимых для достижения заданной точности, может оказаться больше, чем при выборе $y_{k+1}^{(0)}$ по методу Адамса.

Заметим, что порядок пошаговой погрешности интерполяционного метода Адамса порядка r совпадает с порядком для экстраполяционного метода и равен $r+2$. Однако сама погрешность как правило меньше и имеет противоположный знак.

В случае $r=1$ порядок пошаговой погрешности равен 3 и, следовательно, порядок суммарной погрешности на некотором промежутке равен 2. Покажем это на примере интерполяционной формулы Адамса для $r=1$ (формула Эйлера с пересчетом).

Пример 13.3. Рассмотрим задачу Коши примера 11.2. В данном случае $f(,y)=-y$, и уравнение, соответствующее (13.7.4), принимает вид

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

Это уравнение позволяет явно (без итераций) найти

$$y_{k+1}=\frac{1-h/2}{1+h/2}y_k,$$

откуда в силу условия $y_0=1$ следует

$$y_k=\left(\frac{1-h/2}{1+h/2}\right)^k, \qquad k=0,1,2,\ldots$$

Пусть $X>0$ — некоторое число, n — целое положительное число. Полагая $h=X/n$ и обозначая через $\tilde{y}(x)$ приближенное решение задачи, находим

$$\tilde{y}(X)=\left(\frac{1-h/2}{1+h/2}\right)^{X/h}$$

и, следовательно,

$$\begin{aligned}\ln\tilde{y}(X) &= \frac{X}{h}\ln\left[\left(1-\frac{h}{2}\right)-\ln\left(1+\frac{h}{2}\right)\right]\\ &= \frac{X}{h}\left[\left(-\frac{h}{2}-\frac{1}{2}\left(\frac{h}{2}\right)^2-\frac{1}{3}\left(\frac{h}{2}\right)^3-\ldots\right)-\left(\frac{h}{2}-\frac{1}{2}\left(\frac{h}{2}\right)^2+\frac{1}{3}\left(\frac{h}{2}\right)^3-\ldots\right)\right]\\ &= -\frac{X}{h}\left(h+\frac{2}{3}\left(\frac{h}{2}\right)^3+\ldots\right)=-X-X\left(\frac{h^2}{12}+\frac{h^4}{80}+\ldots\right).\end{aligned}$$

С другой стороны, точное решение задачи $y(x)=e^{-x}$, и поэтому

$$\ln y(x)-\ln\tilde{y}(x)=X\left(\frac{h^2}{12}+\frac{h^4}{80}+\ldots\right).$$

Отсюда легко найти

$$\lim_{h\to 0}\frac{y(X)-\tilde{y}(X)}{h^2}=\frac{X^2}{12}e^{-X}\neq 0,$$

т. е. порядок суммарной погрешности $y(X)-\tilde{y}(X)$ равен 2.

Замечание. Пусть s — целое число, $0\le s\le r$. Если $y(x)$ — решение уравнения (13.1.1), то очевидна формула

$$y(x_{k+1})-y(x_{k-s})=\int\limits_{x_{k-s}}^{x_{k+1}}y'(x)\,dx=\int\limits_{x_{k-s}}^{x_{k+1}}f(x,y(x))\,dx.$$

Заменяя $f(x,y(x))$ на интерполяционный многочлен $I_r(x)$ или $\tilde{I}_r(x)$, получают другие расчетные экстраполяционные или интерполяционные формулы типа

$$y_{k+1}=y_{k-s}+h\sum_j c_j^r f_{k-j}$$

(Милна, Нистрема, Коуэлла и др.). Рассматривать их мы не будем.

13.8. Решение задачи Коши для системы дифференциальных уравнений

Дифференциальные уравнения высшего порядка для их численного решения обычно преобразуются в нормальную форму: систему уравнений первого порядка, разрешенных относительно производной. Число уравнений нормальной системы должно быть равно числу неизвестных функций. Такое преобразование дает и некоторую дополнительную выгоду, поскольку производные искомой величины (в особенности первая) также могут представлять определенную ценность, а достаются «бесплатно».

Обсуждавшееся выше понятие устойчивости применимо также к системам уравнений. Его определяет *якобиан* системы уравнений — матрица $J=\{\partial f_i/\partial y_j\}$. Устойчивость системы непосредственно связана с собственными значениями матрицы J. Устойчивостью управляет собственное значение с алгебраически наибольшей вещественной частью. Система устойчива, если у нее нет собственных значений с положи-

тельными вещественными частями. Характеристики устойчивости могут зависеть от t.

В частности, для системы второго порядка постановка задачи Коши приобретает вид: решить систему

$$\begin{aligned} y' &= f_1(x,y,z); \\ z' &= f_2(x,y,z) \end{aligned} \tag{13.8.1}$$

с начальными условиями

$$\begin{aligned} y|_{x=x_0} &= y_0; \\ z|_{x=x_0} &= z_0. \end{aligned} \tag{13.8.2}$$

Для иллюстрации общей идеи численной реализации также достаточно системы второго порядка, так как все рассмотренные в разд. 13.4–13.6 численные методы решения дифференциального уравнения первого порядка без труда переносятся на системы типа (13.8.1)–(13.8.2) или более общие; на каждом шаге интегрирования ведется параллельный счет приращений всех неизвестных функций по аналогичным формулам. Например, расчетная схема метода Эйлера для системы двух уравнений и $k = 0, 1, \ldots$ запишется в форме

$$\begin{aligned} y_{k+1} &= y_k + \Delta y_k; \\ z_{k+1} &= z_k + \Delta z_k, \end{aligned}$$

где

$$\begin{aligned} \Delta y_k &= h f_1(x_k, y_k, z_k); \\ \Delta z_k &= h f_2(x_k, y_k, z_k). \end{aligned}$$

Для эффективного решения систем дифференциальных уравнений важно организовать рациональное (однократное) вычисление общих подвыражений.

13.9. «Жесткие» системы

Постоянная времени T дифференциального уравнения первого порядка — это промежуток времени, по истечении которого величина нестационарной части решения убывает в e раз. Например, уравнение $y' = -\lambda y$ при $\lambda > 0$ имеет решение $Ce^{-\lambda t}$, и $T = 1/\lambda$.

В общем случае дифференциальное уравнение n-го порядка имеет n постоянных времени. Если любые две из них сильно различаются по величине или одна из них мала в сравнении с интервалом времени, для которого отыскивается решение, то задача называется «жесткой», и ее практически невозможно решить обычными методами. Здесь шаг должен быть достаточно мал, чтобы можно было учитывать динамику наиболее быстро изменяющихся членов уравнения даже после того, как их вклад станет практически незаметным. Уменьшение величины шага резко увеличивает время счета и приводит к такому накоплению погрешностей усечения и округления, которое может сделать результат бессмысленным. При увеличении шага решение становится неустойчивым.

Термин жесткости связывается с *комплексом понятий*: задача, отрезок интегрирования, численный метод. Здесь процесс, начатый из исходной точки, очень быстро стремится к некоторой относительно плавной кривой. Его поведение в начальной области («пограничный слой») качественно отлично от «дальней перспективы» — см. рис. 13.3.

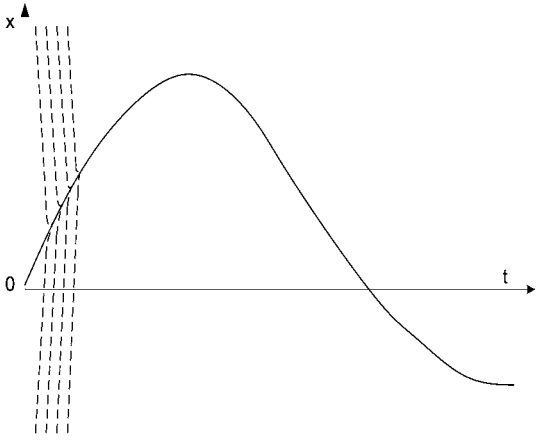

Рис. 13.3. «Жесткая» система

Интерес к той или иной области зависит от приложения. Жесткие задачи особенно характерны для расчета электрических сетей, проблем управления и задач химической кинетики (см. [37, с. 365–366] относительно концентрации озона в атмосфере, где скорости учитываемых реакций различаются на 10 порядков).

Большинство стандартных методов не приспособлено для решения жестких уравнений, и прежде чем программа классифицирует систему как жесткую, произойдет резко замедляющее счет многократное умень-

шение длины шага. Для получения численного решения жесткого уравнения на достаточно большом промежутке естественно построить с мелким шагом его решение в переходной фазе, а затем продолжить численный процесс с крупным шагом в области медленных изменений решения.

По современным взглядам, методы «предиктор-корректор» с фиксированным числом итераций для жестких задач недостаточны. Итерации должны продолжаться до достижения сходимости, причем выбираемый шаг $h \approx |J|^{-1}$.

13.10. Контроль пошаговой погрешности

Пусть $\delta > 0$ — заданное малое число. Как выбрать шаг h, чтобы пошаговая погрешность $|\bar{\Delta}y - \Delta y|$ не превосходила $\delta > 0$ и в то же время не была значительно меньше δ (в последнем случае шаг целесообразно было бы увеличить)? Для определенности предположим, что вычисления ведутся основным методом Рунге—Кутты (l=5). В этом случае шаговая погрешность примерно равна h^5 (C — некоторая постоянная), и, следовательно, при переходе от точки $\bar{x}$ к $x + 2h$ с шагом h накопится погрешность $2Ch^5$. Таким образом, будем иметь

$$(\bar{\Delta}y)_{2h} - (\Delta y)_{h+h} \approx 2Ch^5. \qquad (13.10.1)$$

С другой стороны, если от точки $\bar{x}$ перейти к точке $\bar{x} + 2h$ одним шагом $2h$, то погрешность окажется примерно равной $(2h)^5 = 32Ch^5$ и, следовательно, получим

$$(\bar{\Delta}y)_{2h} - (\Delta y)_{2h} \approx 32Ch^5. \qquad (13.10.2)$$

Вычитая (13.10.1) из (13.10.2), находим

$$\frac{(\bar{\Delta}y)_{h+h} - (\Delta y)_{2h}}{15} \approx 2Ch^5. \qquad (13.10.3)$$

Из сравнения (13.10.1) и (13.10.3) заключаем: если величина

$$\varepsilon = \frac{|(\bar{\Delta}y)_{h+h} - (\Delta y)_{2h}|}{15} \approx 2Ch^5$$

несколько меньше δ, то шаг h удовлетворителен и вычисления можно продолжать, находя через каждые два шага h величину $(\Delta y)_{2h}$; если ε

окажется значительно меньше δ (в 30 и более раз), то можно сделать попытку шаг удвоить; если $\varepsilon > \delta$, то шаг следует уменьшить вдвое.

Не следует увеличивать шаг в каждой точке. Если применяется метод p-го порядка, то увеличение шага допускается не чаще одного раза за каждые $p+1$ узлов, чтобы оценки ошибок могли стабилизироваться [37, с. 379].

Иногда автоматический выбор шага оказывается «слишком эффективным»: набор точек становится чрезмерно редким и непригодным для графопостроителя.

Аналогичный анализ возможен и для многошаговых схем. Пусть, например, счет ведется одновременно по экстраполяционной и интерполяционной схемам Адамса порядка $r = 3$. Пусть $y_{\text{пр}}$ и $y_{\text{ут}}$ означают результаты прогноза и уточнения соответственно. Тогда

$$\varepsilon = \frac{1}{14}|y_{\text{пр}} - y_{\text{ут}}|,$$

и в зависимости от соотношения между ε и δ делаются те же выводы о выбранном шаге h. Поскольку замена шага h на $h/2$ потребует создания нового разгонного участка, новые стартовые точки проще всего получить интерполяцией между ранее полученными $\{y_k\}$.

13.11. Сравнительный анализ методов численного решения задачи Коши

Сконструировать программу, которая эффективно решала бы любую задачу Коши для обыкновенных дифференциальных уравнений, невозможно. Помимо прочего, нужно принимать во внимание специальные свойства конкретных задач: требуемую точность, желаемое время счета, «жесткость», сложность вычисления правых частей, устойчивость.

Стандартные программы численного решения задачи Коши обычно оценивают пошаговую погрешность и при ее малости в целях ускорения счета увеличивают шаг. Динамический выбор порядка делает соответствующие программы самостартующими.

Каждый из описанных методов обладает некоторыми преимуществами перед другими. Сравним эти методы на задаче решения диффе-

ренциального уравнения $y' = 6y/(1+x)$ при $y_0 = 1$ для сопоставимых вариантов обсуждавшихся методов. Точное решение упомянутого уравнения есть $y(x) = (1+x)^6$.

Одношаговые методы удобны тем, что не требуют разгонных участков и гибки в выборе шага на любом этапе решения задачи. В этом преимущество основного метода Рунге—Кутты перед методом Адамса третьего порядка (напомним, что порядок пошаговой погрешности для них одинаков).

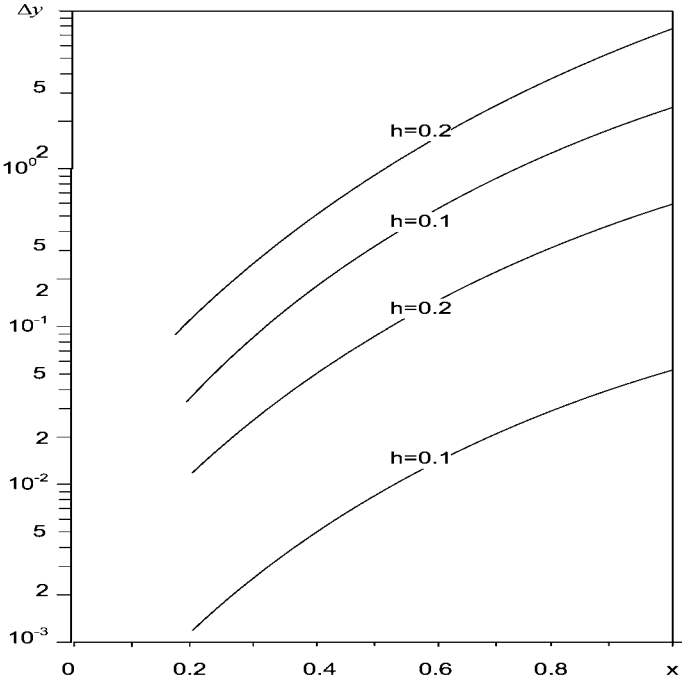

Рис. 13.4. Погрешности метода Рунге—Кутты

Экстраполяционные многошаговые методы требуют лишь однократного вычисления $f(x,y)$ при переходе от $\bar{x}$ к $\bar{x}+h$; для этой же цели в методе Рунге—Кутты требуется четырехкратное вычисление $f(x,y)$. Если функция $f(x,y)$ сложна, то преимущество метода Адамса будет ощутимым.

Интерполяционные многошаговые методы (с пересчетом), обладая лучшей точностью по сравнению с аналогичными экстраполяционными методами, имеют тот недостаток, что число необходимых итераций (пересчетов) может оказаться большим. По-видимому, целесообразны комбинированные методы (прогноза и коррекции).

Рис. 13.5 дает погрешности метода Адамса 2-го, 3-го и 4-го порядков для шага интегрирования $h = 0.1$.

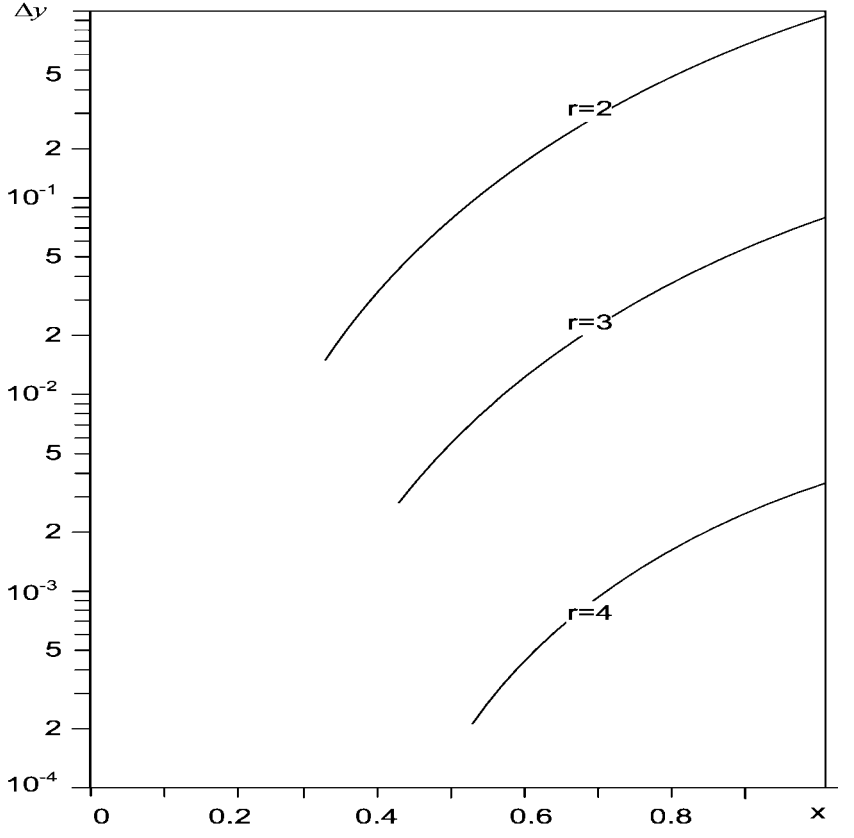

Рис. 13.5. Погрешности метода Адамса

Рис. 13.6 показывает погрешности метода Милна для шагов интегрирования 0.1 и 0.2 в зависимости от числа итераций «прогноз — коррекция». В каждом пучке кривых верхняя кривая соответствует одной итерации, средняя — двум, а нижняя — трем.

В [110, с. 96] можно найти более развернутое сопоставление методов задачи Коши.

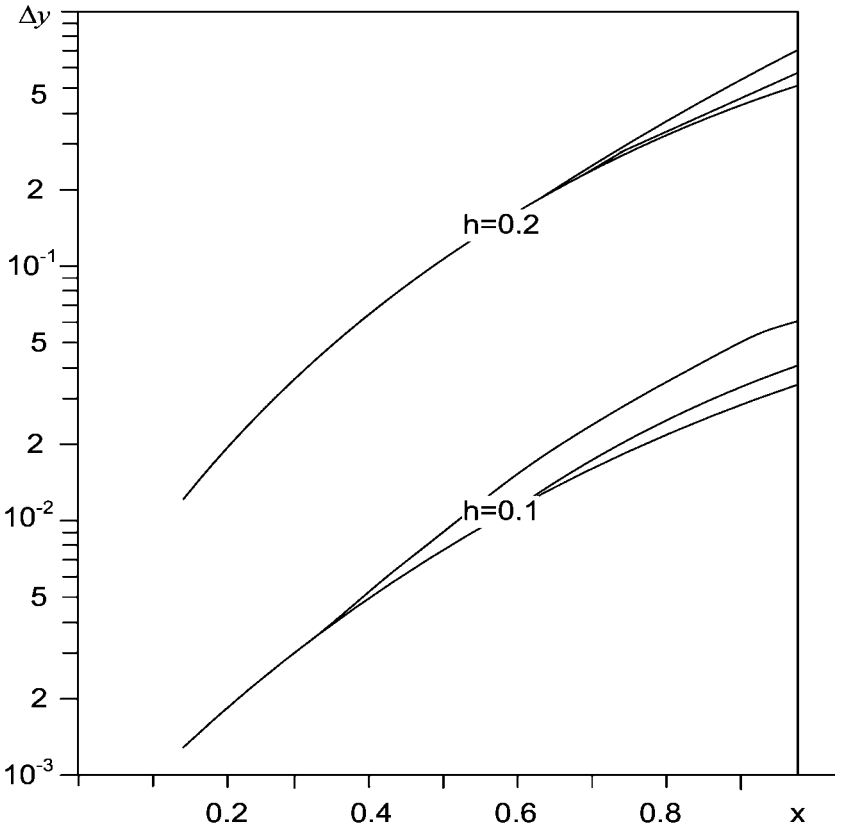

Рис. 13.6. Погрешности метода Милна

Глава 14.
Интегральные уравнения

14.1. Определения и классификация

Интегральным называется уравнение, в которое искомая функция входит под знаком интеграла. Основанием для составления уравнений обычно служат общие физические (в широком смысле слова) законы. Известные законы сохранения массы, импульса, энергии имеют интегральную формулировку и приводят к интегральным уравнениям. Интегральными уравнениями отражаются законы газовой и гидродинамики, электродинамики, экологии, теории упругости, сохранения стационарной очереди в теории массового обслуживания [84] и т. д.

Любое из известных в анализе преобразований (варианты преобразования Фурье, преобразования Лапласа, Меллина, Ханкеля и др.) можно рассматривать как интегральное уравнение относительно функции, над которой это преобразование выполнено.

В отличие от дифференциальных, интегральные уравнения не содержат производных искомой функции и потому не накладывают ограничений на гладкость решения. Здесь ограничения формулируются в терминах их интегрируемости. Говорят, что $f(x)$ есть функция с интегрируемым на $[a,b]$ квадратом, если интеграл $\int_a^b f^2(x)\,dx$ существует (конечен). Совокупность всех функций с интегрируемым на $[a,b]$ квадратом обозначим $L_2[a,b]$ (далее в данном контексте — просто L_2). Основные свойства функций из L_2:

1. Произведение двух функций с интегрируемым квадратом есть интегрируемая функция.

2. Сумма двух функций из L_2 также принадлежит L_2.

3. Если $f(x) \in L_2$ и λ — произвольное вещественное число, то $\lambda f(x) \in L_2$.

4. Если $f(x) \in L_2$ и $g(x) \in L_2$, то выполняется неравенство Буняковского—Шварца

$$\Big[\int\limits_a^b f(x)g(x)\,dx\Big]^2 \le \int\limits_a^b f^2(x)\,dx \int\limits_a^b g^2(x)\,dx.$$

Нормой функции $f(x) \in L_2$ называют неотрицательное число

$$||f(x)|| = \sqrt{(f,f)} = \Big[\int\limits_a^b f^2(x)\,dx\Big]^{1/2}.$$

Линейное интегральное уравнение общего вида записывается как

$$r(x)y(x) = \int\limits_a^b K(x,t)y(t)\,dt + f(x). \qquad (14.1.1)$$

Функции $r(x)$, $K(x,t)$, $f(x)$ предполагаются известными. Функция $K(x,t)$, $a \le x,t \le b$, называется *ядром* уравнения, а $f(x)$, $a \le x \le b$ — его свободным членом. Все они считаются принадлежащими к L_2.

В случае $r(x) = 0$ имеем уравнение *первого* рода, в противном случае — *второго*. Уравнение второго рода приводится к стандартному виду

$$y(x) = \int\limits_a^b K(x,t)y(t)\,dt + f(x). \qquad (14.1.2)$$

Такое уравнение называется уравнением *Фредгольма*. При переменном верхнем пределе

$$y(x) = \int\limits_a^x K(x,t)y(t)\,dt + f(x) \qquad (14.1.3)$$

получаем уравнение *Вольтерра* второго рода.

Однородное интегральное уравнение Фредгольма 2-го рода

$$y(x) = \lambda \int_a^b K(x,t)y(t)\,dt \tag{14.1.4}$$

по разъясняемым ниже причинам записывается с параметром λ. Оно всегда имеет *тривиальное* решение $y(x) \equiv 0$. Значения параметра λ, при которых существуют и ненулевые решения, суть *характеристические числа* уравнения (14.1.4) [ядра $K(x,t)$], а каждое ненулевое решение называется *собственной функцией*, соответствующей собственному числу λ. При решении прикладных задач эти понятия приобретают четкую физическую интерпретацию. Известно, например, что при некоторой скорости вращения вала, которая называется критической, вал начинает колебаться около своей продольной оси. Квадрат этой скорости равен характеристическому числу соответствующего интегрального уравнения ([49, с. 54–55]).

Уравнения *Фредгольма первого рода* имеют вид

$$\int_a^b K(x,t)y(t)\,dt = f(x). \tag{14.1.5}$$

Они чрезвычайно чувствительны к малым изменениям правой части. Пусть, например, задано уравнение

$$\int_0^1 x^2 t y(t)\,dt = 3x^2 + \varepsilon. \tag{14.1.6}$$

Оно приводится к

$$x^2 \int_0^1 t y(t)\,dt = 3x^2 + \varepsilon.$$

Предположим, что для функции $y(t)$ имеет место $\int_0^1 ty(t)\,dt = 3$ (таких функций существует бесконечное множество). При $\varepsilon = 0$ эту функцию можно считать решением уравнения (14.1.6); в случае же $\varepsilon \neq 0$ решение последнего не существует. Задачи данного вида относятся к *некорректным* и при необходимости в их решении требуют регуляризации. Разрешимость уравнений Фредгольма первого рода с симметричным ядром устанавливается теоремой Пикара — см. [49, с. 137].

Примером интегрального уравнения Вольтерра 1-го рода является уравнение Абеля

$$\int_0^x \frac{y(t)}{\sqrt{x-t}}\,dt = f(x), \tag{14.1.7}$$

где $f(x)$ — заданная функция, а $y(t)$ — искомая. В случае $f(x)=\text{const}$ имеем классическую задачу о *таутохроне*: найти кривую, скользя вдоль которой без трения, тяжелая частица достигает своего самого низкого положения за одно и то же время независимо от ее начального положения. Ее решением является циклоида.

Интегральное уравнение Вольтерра 1-го рода

$$\int_0^x K(x-t)y(t)\,dt = f(x), \tag{14.1.8}$$

ядро которого зависит лишь от разности аргументов $x-t$, называется интегральным уравнением типа *свертки*. К уравнениям этого типа сводятся многие задачи о распределениях временных интервалов и задачи об управлении запасами — см. [49, с. 130–131]. Частным случаем подобных уравнений с разностным ядром на полуоси является имеющее многочисленные приложения уравнение *Винера–Хопфа*.

Поскольку уравнения Вольтерра первого рода могут быть приведены к одноименным уравнениям второго рода, в дальнейшем мы ограничимся только уравнениями второго рода, дополнительно это обстоятельство не оговаривая.

14.2. Теоремы существования и единственности решения

Линейные интегральные уравнения называются *однородными*, если $f(x)\equiv 0$. Соответственно имеем однородные уравнения Фредгольма

$$y(x) = \int_a^b K(x,t)y(t)\,dt \tag{14.2.1}$$

и Вольтерра

$$y(x) = \int_a^x K(x,t)y(t)\,dt. \qquad (14.2.2)$$

В обоих случаях очевидным решением является нулевое: $y(x) \equiv 0$. Доказано, что для уравнения типа Вольтерра оно является единственным. Для уравнения Фредгольма могут существовать и ненулевые решения. В этом случае они называются *собственными функциями* ядра $K(x,t)$.

Относительно решений интегральных уравнений установлены следующие теоремы:

1. Уравнение Вольтерра (14.1.3) имеет единственное решение при любом выборе свободного члена $f(x)$.

2. Для интегральных уравнений Фредгольма имеет место альтернатива: либо неоднородное линейное уравнение 2-го рода

$$y(x) - \lambda \int_a^b K(x,t)y(t)\,dt = f(x) \qquad (14.2.3)$$

имеет единственное решение при любой функции $f(x)$ из некоторого достаточно широкого класса, либо соответствующее *однородное* уравнение

$$y(x) - \lambda \int_a^b K(x,t)y(t)\,dt = 0 \qquad (14.2.4)$$

имеет по крайней мере одно нетривиальное решение. Доказательство отсутствия таковых означает существование решения неоднородного уравнения.

3. Достаточным условием существования и единственности решения уравнения (14.1.2) является неравенство

$$|b-a| \cdot \max_{a\le x,t\le b} |K(x,t)| < 1. \qquad (14.2.5)$$

В дальнейшем при обсуждении приближенных методов решения уравнения (14.1.2) существование и единственность этого решения предполагаются.

14.3. Уравнения Фредгольма с вырожденным ядром

Пусть $u_1(x), u_2(x), \ldots, u_n(x)$ и $v_1(x), v_2(x), \ldots, v_n(x)$ — две совокупности по n функций, определенных на $[a, b]$. Функции каждой из них считаем линейно независимыми.

Ядро $K(x,t)$ называется *вырожденным*, если оно представимо в виде

$$K(x,t) = \sum_{j=1}^{n} u_j(x) v_j(t). \tag{14.3.1}$$

Вырожденным, например, является ядро $K(x,t) = \sin(x+t) = \sin x \cos t + \cos x \sin t$: здесь $n = 2$, $u_1(x) = \sin x$, $u_2(x) = \cos x$, $v_1(t) = \cos t$, $v_2(t) = \sin t$.

В случае вырожденного ядра для любой функции $y(t)$ имеем

$$\begin{aligned} \int_a^b K(x,t) y(t)\, dt &= \int_a^b [u_1(x)v_1(t) + \ldots + u_n(x)v_n(t)] y(t)\, dt \\ &= u_1(x) \int_a^b v_1(t) y(t)\, dt + \ldots + u_n(x) \int_a^b v_n(t) y(t)\, dt. \end{aligned}$$

Таким образом, для вырожденного ядра

$$\int_a^b K(x,t) y(t)\, dt = c_1 u_1(x) + c_2 u_2(x) + \ldots + c_n u_n(x), \tag{14.3.2}$$

где $c_1, c_2, \ldots, c_n$ удовлетворяют системе n линейных уравнений

$$\begin{aligned} c_k &= \int_a^b y(x) v_k(x)\, dx = c_1 \int_a^b u_1(x) v_k(x)\, dx + \ldots \\ &+ c_n \int_a^b u_n(x) v_k(x)\, dx + \int_a^b f(x) v_k(x)\, dx, \qquad k = \overline{1, n}. \end{aligned}$$

Вводя обозначения

$$f_k \stackrel{\text{def}}{=} \int_a^b f(x)v_k(x)\,dx, \qquad d_{kj} \stackrel{\text{def}}{=} \int_a^b u_j(x)v_k(x)\,dx, \tag{14.3.3}$$

перепишем предыдущее уравнение в виде

$$c_k = d_{k1}c_1 + d_{k2}c_2 + \ldots d_{kn}c_n + f_k, \qquad k = \overline{1,n}.$$

Полагая $a_{kj} \stackrel{\text{def}}{=} -d_{kj}, \quad k \neq j$, и $a_{kk} \stackrel{\text{def}}{=} 1 - d_{kk}$, получаем систему уравнений

$$\begin{array}{ccccccccc} a_{11}c_1 & + & a_{12}c_2 & + & \ldots & + & a_{1n}c_n & = & f_1, \\ a_{21}c_1 & + & a_{22}c_2 & + & \ldots & + & a_{2n}c_n & = & f_2, \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1}c_1 & + & a_{n2}c_2 & + & \ldots & + & a_{nn}c_n & = & f_n. \end{array} \tag{14.3.4}$$

Несколько заключительных замечаний:

1. Если определитель системы (14.3.4) отличен от нуля, то интегральное уравнение имеет (14.1.2) единственное решение. В противном случае система либо несовместна, либо имеет бесчисленное множество решений — в зависимости от набора чисел $\{f_k\}$, определяемых свободным членом $f(x)$ исходного интегрального уравнения.

2. Простота нахождения решения уравнения с вырожденным ядром естественно приводит к мысли о приближенной замене данного произвольного ядра $K(x,t)$ на вырожденное $L(x,t)$. В качестве последнего применительно к ядру $K(x,t)$ можно брать отрезок ряда Тейлора или ряда Фурье по любой полной ортонормированной системе функций — см. [49, гл. 5]. Например, невырожденное ядро $K(x,t) = e^{xt}$ можно представить степенным рядом с конечным числом членов

$$\tilde{K}_n(x,t) \stackrel{\text{def}}{=} 1 + \frac{xt}{1!} + \frac{(xt)^2}{2!} + \ldots + \frac{(xt)^{n-1}}{(n-1)!}.$$

Теперь его можно считать вырожденным с порождающими наборами функций

$$u_k(x) = x^{k-1}, \quad v_k(t) = \frac{t^{k-1}}{(k-1)!}, \qquad k = \overline{1,n}.$$

14.4. Разложение по координатным функциям

14.4.1. Постановка задачи

Представим решение $Y_n(x)$ интегрального уравнения

$$y(x) = f(x) + \lambda \int_a^b K(x,t)y(t)\,dt$$

в виде комбинации известных линейно независимых координатных функций:

$$Y_n(x) = y_0(x) + \sum_{i=1}^{n} c_i y_i(x) \tag{14.4.1}$$

(без потери общности можно считать $y_0 \equiv 0$).

Невязка предлагаемого решения

$$R[Y_n] = y_0(x) + \sum_{i=1}^{n} c_i y_i(x) - f(x) - \lambda \int_a^b K(x,t)\Big[y_0(t) + \sum_{i=1}^{n} c_i y_i(t)\Big]\,dt,$$

или

$$R[Y_n(x)] = \psi_0(x,\lambda) + \sum_{i=1}^{n} c_i \psi_i(x,\lambda), \tag{14.4.2}$$

где

$$\begin{aligned} \psi_0(x,\lambda) &= y_0(x) - f(x) - \lambda \int_a^b K(x,t)y_0(t)\,dt, \\ \psi_i(x,\lambda) &= y_i(x) - \lambda \int_a^b K(x,t)y_i(t)\,dt, \qquad i = \overline{1,n}. \end{aligned} \tag{14.4.3}$$

Параметры $c_1, c_2, \ldots, c_n$ подбирают так, чтобы эта невязка была в определенном смысле возможно малой. Ниже рассматриваются различные критерии и, соответственно, методы минимизации невязки.

Заметим, однако, что из малости невязки *не следует близость функций* $Y_n(x)$ и $y(x)$ в смысле равномерной или квадратичной нормы. Успех применения методов этой группы определяется тем, насколько удачно выбраны координатные функции $y_1(x), y_2(x), \ldots, y_n(x)$.

14.4.2. Метод коллокации

Метод коллокации предполагает обращение невязки в нуль в заданной системе точек $\{x_j\}$, $j=\overline{1,n}$, из отрезка коллокации $[a,b]$. Соответственно на основании формулы (14.4.2) получаем систему линейных алгебраических уравнений

$$\sum_{i=1}^{n} c_i \psi_i(x_j,\lambda) = -\psi_0(x_j,\lambda), \qquad j=\overline{1,n}. \tag{14.4.4}$$

14.4.3. Метод наименьших квадратов

Здесь весовые коэффициенты $\{c_i\}$ отыскиваются из условия минимума интеграла

$$\int\limits_a^b \{R[Y_n]\}^2\,dx = \int\limits_a^b \left[\psi_0(x,\lambda) + \sum_{i=1}^{n} c_i\psi_i(x,\lambda)\right]^2 dx. \tag{14.4.5}$$

Дифференцируя по параметрам под знаком интеграла и приравнивая производные нулю, получаем уравнения

$$\frac{1}{2}\frac{\partial I}{\partial c_j} = \int\limits_a^b \psi_j(x,\lambda)\left[\psi_0(x,\lambda) + \sum_{i=1}^{n} c_i\psi_i(x,\lambda)\right] dx = 0, \qquad j=\overline{1,n}. \tag{14.4.6}$$

Вводя сокращенные обозначения

$$(\psi_i,\psi_j) = \int\limits_a^b \psi_i(x,\lambda)\psi_j(x,\lambda)\,dx, \tag{14.4.7}$$

можно переписать систему (14.4.6) в виде нормальной системы метода наименьших квадратов:

$$\begin{array}{ccccccccc}
c_1(\psi_1,\psi_1) & + & c_2(\psi_1,\psi_2) & + & \ldots & + & c_n(\psi_1,\psi_n) & = & -(\psi_1,\psi_0);\\
c_1(\psi_2,\psi_1) & + & c_2(\psi_2,\psi_2) & + & \ldots & + & c_n(\psi_2,\psi_n) & = & -(\psi_2,\psi_0);\\
\multicolumn{9}{c}{\dotfill}\\
c_1(\psi_n,\psi_1) & + & c_2(\psi_n,\psi_2) & + & \ldots & + & c_n(\psi_n,\psi_n) & = & -(\psi_n,\psi_0).
\end{array} \tag{14.4.8}$$

Вместо интегрального варианта метода наименьших квадратов можно рассмотреть его дискретную версию.

14.4.4. Метод моментов

В данном методе приближенное решение задачи представляется конечной суммой

$$Y_n(x) = f(x) + \sum_{i=1}^{n} c_i y_i(x). \tag{14.4.9}$$

Тогда невязка

$$R[Y_n] = \sum_{j=1}^{n} c_j \Big[y_j(x) - \lambda \int_a^b K(x,t) y_j(t)\, dt \Big] - \lambda \int_a^b K(x,t) f(t)\, dt. \tag{14.4.10}$$

Коэффициенты $\{c_j\}$ здесь определяются из условия ортогональности невязки на отрезке $[a,b]$ ко всем координатным функциям:

$$\int_a^b R[Y_n] y_i(x)\, dx = 0, \qquad i = \overline{1,n}.$$

Полагая

$$\begin{aligned} \alpha_{ij} &= \int_a^b y_i(x) y_j(x)\, dx, \\ \beta_{ij} &= \int_a^b dx \int_a^b K(x,t) y_i(x) y_j(t)\, dt, \\ \gamma_i &= \int_a^b dx \int_a^b K(x,t) y_i(x) f(t)\, dt, \end{aligned}$$

получаем окончательный вариант уравнений для весовых коэффициентов

$$\sum_{j=1}^{n} c_j (\alpha_{ij} - \lambda \beta_{ij}) = \lambda \gamma_i, \qquad i = \overline{1,n}. \tag{14.4.11}$$

14.4.5. Метод Бубнова—Галеркина

В этом методе выбирают систему линейно независимых функций $\{y_i(x)\}$, полную в $L_2[a,b]$, и ищут приближенное решение уравнения

$$y(x) = f(x) + \lambda \int_a^b K(x,t)y(t)\,dt \tag{14.4.12}$$

в виде усеченного ряда

$$Y_n(x) = \sum_{i=1}^{n} c_i y_i(x). \tag{14.4.13}$$

Коэффициенты $\{c_i\}$ определяются из системы линейных уравнений

$$(Y_n(x), y_i(x)) = (f(x), y_i(x)) + \lambda\Big[\int_a^b K(x,t)Y_n(t)\,dt, y_i(x)\Big], \qquad i = \overline{1,n}, \tag{14.4.14}$$

где (f,g) означает скалярное произведение функций, а вместо $Y_n(x)$ подставляется степенной ряд (14.4.13). Если значение λ в уравнении (14.4.13) не является характеристическим, то при достаточно больших n (14.4.12) разрешима, и при $n \to \infty$ ряд (14.4.13) стремится к точному решению уравнения.

Для вырожденных ядер метод Бубнова—Галеркина дает точное решение; для общего случая он эквивалентен замене ядра на вырожденное.

14.5. Метод итерируемых ядер

Будем рассматривать вместо (14.1.2) интегральное уравнение

$$y(x) = \lambda \int_a^b K(x,t)y(t)\,dt + f(x). \tag{14.5.1}$$

Уравнение (14.1.2) получается из него как частный случай при $\lambda = 1$. При $\lambda = 0$ уравнение (14.5.1) имеет очевидное решение $y(x) = f(x)$.

Попытаемся получить его решение в виде *ряда Неймана* — разложения по степеням λ

$$y(x) = y_0(x) + \lambda y_1(x) + \ldots + \lambda^n y_n(x) + \ldots, \qquad (14.5.2)$$

причем $y_0(x) = f(x)$. Для сходимости ряда Неймана достаточно, чтобы

$$|\lambda| < \frac{1}{B_K}\Big[\int\int K^2(x,t)\,dxdt\Big]^{-1/2}.$$

Для квадратной области и ограниченного ядра $|K(x,t)| \le M$ это условие сводится к

$$|\lambda| < 1/[M(b-a)].$$

Подставляя (14.5.2) в левую и правую части (14.5.1), получим

$$\begin{aligned} y_0(x) + \lambda y_1(x) + \ldots + \lambda^n y_n(x) + \ldots \;&=\; f(x) + \lambda\Big[\int_a^b K(x,t)y_0(t)\,dt + \\ +\lambda\int_a^b K(x,t)y_1(t)\,dt + \ldots \;&+\; \lambda^{n-1}\int_a^b K(x,t)y_{n-1}(t)\,dt + \ldots\Big]. \end{aligned}$$

Сравнивая коэффициенты при одинаковых степенях λ, получим бесконечную систему рекуррентных соотношений

$$\begin{aligned} y_0(x) &= f(x), \\ y_1(x) &= \int_a^b K(x,t)y_0(t)\,dt, \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ y_n(x) &= \int_a^b K(x,t)y_{n-1}(t)\,dt, \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots \end{aligned} \qquad (14.5.3)$$

Пример 14.1 [49, с. 155]. Пусть необходимо решить уравнение

$$y(x) = 1 + \int_0^x y(t)\,dt.$$

Положим $y_0 \equiv 0$. Тогда $y_1 \equiv 1$. Выполняя последовательные подстановки, получаем

$$y_2(x) \;=\; 1 + \int_0^x 1 \cdot dt = 1 + x,$$

$$y_3(x) = 1 + \int_0^x (1+t)\,dt = 1 + x + x^2/2,$$
$$y_4(x) = 1 + \int_0^x (1+t+t^2/2)\,dt = 1 + x + x^2/2! + x^3/3!$$

и т. д. Очевидно,

$$y_n(x) = 1 + x + x^2/2! + \ldots + x^{n-1}/(n-1)!.$$

Отсюда следует, что

$$\lim_{n\to\infty} y_n(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x.$$

Легко проверить, что e^x действительно является решением исходного интегрального уравнения.

14.6. Замена интеграла квадратурной суммой

Численные методы решения интегральных уравнений опираются на замену интеграла конечной суммой, реализующей какую-либо квадратурную формулу. В результате задача сводится к алгебраической системе относительно дискретных значений (*каркаса*) искомого решения, соответствующих узлам этой формулы.

Если нам нужны значения искомой функции только в этих точках, то задача решена. Если нужны еще и другие, то по каркасу можно построить интерполяционный многочлен или аппроксимацию какого-либо иного вида. Хорошие результаты будут получены лишь в случае, когда алгебраический порядок точности квадратурной формулы хорошо согласуется со степенями гладкости ядра и свободного члена интегрального уравнения.

Точность численного решения интегрального уравнения квадратурным методом зависит от применяемой квадратурной формулы, количества узлов и, разумеется, свойств функций, определяющих уравнение. Известные оценки погрешностей чрезвычайно сложны. Непосредственное использование принципа Рунге (с дроблением шага) возможно только на равномерной сетке (следовательно, лишь для квадратур Ньютона—Котеса). Для других квадратурных формул необходим пересчет «новых» результатов к «старым» узлам.

14.6.1. Уравнение Вольтерра

Будем предполагать, что ядро $K(x,t)$ и свободный член $f(x)$ уравнения (14.1.3) типа Вольтерра непрерывны в $a \le t \le x \le b$ и в $a \le x \le b$ соответственно и решение $y(x)$ разыскивается в конечном промежутке $[a,b]$.

Выберем на этом промежутке равноотстоящие узлы $\{x_k\}$ с шагом $h = (b-a)/n$, включая границы промежутка. Будем искать решение в виде таблицы $\{y_k\}$ для упомянутых точек. Введем обозначения

$$f_k \stackrel{\text{def}}{=} f(x_k), \quad K_{ij} \stackrel{\text{def}}{=} K(x_i, y_j), \qquad i = \overline{0,n}, \quad j = \overline{0,i}.$$

Из уравнения (14.1.3) при $x = a$ получаем $y(a) = 0 + f(a)$, т. е. $y(a) = f(a)$. Поэтому полагаем $y_0 = f(a) = f_0$.

Допустим, что уже найдены $y_0, y_1, y_2, \ldots, y_{k-1}$ для $i < n$. Полагая в (14.1.3) $x = x_i$, получим

$$y_i = y(x_i) = \int\limits_a^{x_i} K(x_i,t)y(t)\,dt + f_i. \tag{14.6.1}$$

Заменяя интеграл в правой части (14.6.1) квадратурной суммой по формуле трапеций и $y(x_j)$ на y_j, имеем

$$y_i = h\Big[\frac{1}{2}K_{i0}y_0 + K_{i1}y_1 + \ldots + K_{i,i-1}y_{i-1} + \frac{1}{2}K_{ii}y_i\Big] + f_i,$$

или

$$\Big(1 - \frac{h}{2}\Big)y_i = h\Big[\frac{1}{2}K_{i0}y_0 + K_{i1}y_1 + \ldots + K_{i,i-1}y_{i-1} + K_{ii}y_i\Big] + f_i.$$

Отсюда

$$y_i = \frac{h}{1 - hK_{ii}/2}\Big[\frac{1}{2}K_{i0}y_0 + K_{i1}y_1 + \ldots + K_{i,i-1}y_{i-1} + f_i\Big]. \tag{14.6.2}$$

Первый раз эта формула применяется при $i = 1$:

$$y_i = \frac{f_1 - K_{i0}f_0 h/2}{1 - K_{11}h/2},$$

а затем при $i = \overline{2,n}$. Шаг предполагается достаточно малым:

$$h \ll 1/\max_{a \le t \le x \le b} |K(x,t)|.$$

14.6.2. Уравнение Фредгольма

Пусть для вычисления определенного интеграла используется конкретная квадратурная формула

$$\int_a^b \varphi(t)\,dt \approx \sum_{j=1}^{n} A_j \varphi(t_j) \tag{14.6.3}$$

с n узлами $t_j \in [a,b]$ и соответствующими весовыми коэффициентами $\{A_j\}$. Подстановка ее в неоднородное интегральное уравнение Фредгольма *второго* рода дает

$$y(x) \approx \lambda \sum_{j=1}^{n} A_j K(x,t_j) x(s_j) + f(x). \tag{14.6.4}$$

Применение этого равенства к точкам $\{x_i\}$, совпадающим с узлами квадратурной формулы, приводит к n равенствам

$$y(x_i) \approx \lambda \sum_{j=1}^{n} A_j K(x_i,x_j) y(x_j) + f(x_i). \tag{14.6.5}$$

Обозначим для краткости

$$K_{ij} = K(x_i,x_j), \quad f_i = f(x_i), \quad y_i = y(x_i).$$

Получаем систему n линейных алгебраических уравнений с n неизвестными $\{x_j\}$:

$$\begin{array}{llll}
(1-\lambda A_1 K_{11})x_1 & -\lambda A_2 K_{12} x_2 & -\ldots & -\lambda A_n K_{1n} x_n = f_1, \\
-\lambda A_1 K_{21} x_1 & +(1-\lambda A_2 K_{22})x_2 & -\ldots & -\lambda A_n K_{2n} x_n = f_2, \\
\ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\
-\lambda A_1 K_{n1} x_1 & -\lambda A_2 K_{n2} x_2 & -\ldots & +(1-\lambda A_n K_{nn})x_n = f_n.
\end{array} \tag{14.6.6}$$

При любых фиксированных $\{A_j, Q_{ij}\}$ за счет параметра λ можно добиться преобладания диагональных элементов, что гарантирует сходимость метода простых итераций.

14.7. Интегро-дифференциальные уравнения

В заключение данной главы покажем применение аппарата интегральных уравнений (и его обобщения) к анализу систем массового обслуживания (СМО).

14.7.1. Идея метода и диаграмма переходов

Все методы расчета систем обслуживания в конечном счете применимы только к не зависящим от предыстории *марковским* процессам в них. Соответствующим расширением понятия «состояние» — иначе говоря, увеличением числа компонент отображающего процесс вектора — его можно свести к марковскому. В частности, в системе $M/G/1$ (одноканальной с марковским потоком заявок и произвольным распределением обслуживания) для вероятностного прогнозирования ее будущего приходится ввести в вектор состояния истекшее время очередного обслуживания.

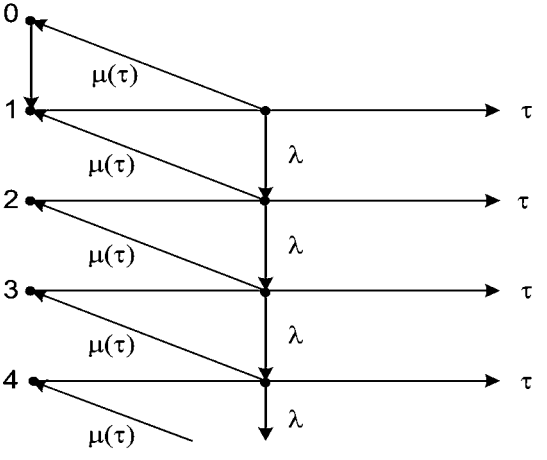

Рис. 14.1. Линейчатая диаграмма переходов для системы $M/G/1$

Полученный марковский процесс в связи с графическим методом его представления — рис. 14.1 — называется *линейчатым*. Здесь номер каждой полупрямой указывает число заявок в системе. Завершение обслуживания уменьшает число заявок на единицу и возобновляет отсчет времени обслуживания τ с нуля. Прибытие заявки приводит к мгновенному переходу вниз на следующую полупрямую при сохранении τ.

14.7.2. Законы сохранения для линейчатых процессов

Согласно [80], в стационарных немарковских задачах средние интенсивности противоположных переходов через разрез равны. Специально для линейчатых процессов имеют место дифференциальные и интегральные варианты закона сохранения заявок:

- для скользящей точки оси производная от плотности вероятностей равна разности интенсивностей переходов по входящим и выходящим стрелкам;

- для начальных точек осей плотность равна интегралу от взвешенных исходными плотностями интенсивностей переходов по входящим стрелкам.

14.7.3. Постановка и решение задачи

Для системы с диаграммой переходов рис. 14.1 интенсивность потока переходов вниз $\lambda = \text{const}$, а мгновенная интенсивность обслуживания $\mu(\tau)$ есть известная плотность завершения длительности обслуживания при условии, что оно не завершено до момента τ.

Обозначим $\rho_k(\tau)$ стационарную плотность вероятности нахождения системы в состоянии $E_k(\tau)$. Применение к диаграмме указанных выше законов сохранения приводит к системе дифференциальных уравнений

$$\begin{aligned} \frac{d\rho_1(\tau)}{d\tau} &= -[\mu(\tau)+\lambda]\rho_1(\tau), \\ \frac{d\rho_k(\tau)}{d\tau} &= -[\mu(\tau)+\lambda]\rho_k(\tau) + \lambda\rho_{k-1}(\tau), \qquad k = 2,3,\ldots \end{aligned} \tag{14.7.1}$$

с начальными условиями

$$\begin{aligned} \rho_1(0) &= \lambda p_0 + \int_0^\infty \mu(\tau)\rho_2(\tau)\,d\tau, \\ \rho_k(0) &= \int_0^\infty \mu(\tau)\rho_{k+1}(\tau)\,d\tau, \qquad k = 2,3,\ldots \end{aligned} \tag{14.7.2}$$

Проведение разреза между состояниями E_0 и E_1 позволяет установить, что

$$\lambda p_0 = \int_0^\infty \mu(\tau)\rho_1(\tau)\,d\tau. \tag{14.7.3}$$

Введем условные плотности вероятности

$$\tilde{\rho}_k(\tau) = \frac{\rho_k(\tau)}{\bar{B}(\tau)}, \qquad k = 1,2,\ldots$$

Сразу же заметим, что при естественном допущении $B(+0)=0$ оказывается $\bar{B}(+0)=1$ и $\tilde{\rho}_k(0)=\rho_k(0)$. Производная

$$\tilde{\rho}_k'(\tau)=\frac{\rho_k'(\tau)\bar{B}(\tau)-\rho_k(\tau)\bar{B}'(\tau)}{\bar{B}^2(\tau)}=\frac{\rho_k'(\tau)}{\bar{B}(\tau)}-\frac{\bar{B}'(\tau)}{\bar{B}(\tau)}\tilde{\rho}_k(\tau)=\frac{\rho_k'(\tau)}{\bar{B}(\tau)}+\mu(\tau)\tilde{\rho}_k(\tau),$$

откуда

$$\frac{\rho_k'(\tau)}{\bar{B}(\tau)}=\rho_k'(\tau)-\mu(\tau)\tilde{\rho}_k(\tau). \tag{14.7.4}$$

Разделив левые и правые части дифференциальных уравнений системы (14.7.1) на $\bar{B}(\tau)$ и используя (14.7.4), убеждаемся, что система (14.7.1) может быть переписана в виде

$$\begin{aligned}\frac{d\tilde{\rho}_1(\tau)}{d\tau} &= -\lambda\tilde{\rho}_1(\tau),\\ \frac{d\tilde{\rho}_k(\tau)}{d\tau} &= -\lambda\tilde{\rho}_k(\tau)\quad+\lambda\tilde{\rho}_{k-1}(\tau),\qquad k=2,3,\ldots\end{aligned} \tag{14.7.5}$$

Поскольку $\mu(\tau)\rho_k(\tau)=\frac{B'(\tau)}{\bar{B}(\tau)}\rho_k(\tau)=\tilde{\rho}_k(\tau)B'(\tau)$, начальные условия (14.7.2) преобразуются в

$$\begin{aligned}\tilde{\rho}_1(0) &= \lambda p_0+\int_0^\infty \tilde{\rho}_2(\tau)\,dB(\tau),\\ \tilde{\rho}_k(0) &= \int_0^\infty \tilde{\rho}_{k+1}(\tau)\,dB(\tau),\qquad k=2,3,\ldots\end{aligned} \tag{14.7.6}$$

Полученная система решается последовательным интегрированием систем дифференциальных уравнений для плотностей с точностью до постоянных. Найденные выражения подставляются в выражения для начальных плотностей, что дает систему линейных уравнений для упомянутых постоянных. Ход решения этой и ряда других аналогичных задач описан в [80].

Глава 15. Математические пакеты и библиотеки

15.1. Стандартные подпрограммы

В предшествующих главах обсуждались *теоретические основы* вычислительной математики и параллельно — элементы программирования ее типовых алгоритмов. На наш взгляд, указанные вопросы при подготовке инженеров должны изучаться обязательно *совместно*, что и имело место в некогда существовавших дисциплинах «Методы вычислений», «Математические методы в инженерном деле» и т. п.

Это, разумеется, не означает необходимости самостоятельно программировать упомянутые методы при решении каждой прикладной задачи. За 50 лет существования вычислительной техники и программирования были созданы богатейшие библиотеки численных подпрограмм. Алгоритмы численных методов вместе с контрольными примерами (иногда — и с теоретическим обоснованием) регулярно публикуются в журналах (назовем «Communications of ACM») и отдельных сборниках («Библиотеки алгоритмов», издававшейся ВЦ АН СССР). Сошлемся также на [89, 100, 26]. История создания, стратегии разработки и функциональные возможности получивших мировую известность пакетов обсуждаются в сборнике [28].

Однако упомянутым богатством надо суметь воспользоваться, и знание математики является ключом к нему, поскольку:

1. Задачи из каждого раздела вычислительной математики настолько разнообразны, что не могут быть перекрыты никакими стандартными процедурами.

2. Постоянно будут возникать вопросы о выборе метода, шага, требуемой точности, начальных приближений и т. п.

3. Приходится балансировать между крайностями: например, в случае квадратур — гарантией получения требуемой точности для любой подынтегральной функции и быстрым вычислением интеграла с нужной точностью для большинства предъявленных задач.

4. Каждый вычислительный алгоритм может иметь свои плюсы:

   – простота программирования,
   – малая трудоемкость (быстрая сходимость),
   – малая дополнительная память,
   – простота оценки погрешности,
   – применимость к широкому кругу задач.

   Выигрывая сравнение по одним показателям, он обязательно уступает конкурентам по другим.

5. Ситуация усугубляется проблемой *переносимости* программ: различием языков и диалектов языков (отклонениями от стандартов); особенностями отдельных трансляторов (например, ограничением на длину массивов — до 64 Кбайт); аппаратными особенностями (различием в машинных ε, условиях переполнения и потери значимости).

6. Эффективное решение достаточно содержательной задачи всегда нуждается в особом подходе, что требует как развития и специализации методов ВМ (вплоть до создания новых), так и самостоятельного программирования.

Выбор пользователя должен определяться ситуацией: количеством задач, их структурой, имеемыми наработками, перспективами, квалификацией сотрудников, вычислительными средствами (бортовая ЭВМ с ограниченными возможностями, персональный компьютер или «mainframe»), ответственностью применений.

Профессиональные разработчики библиотек обычно отдают приоритет «эффективным алгоритмам, ясной документации и точным результатам» [37]. Если в описании правил использования стандартной подпрограммы написано слишком много, то пользователь может не понять того, что написано, и:

- обратиться к худшей ПП, имеющей лучшее описание или более простое обращение;
- неверно обратиться к ПП и получить авост;
- еще хуже — получить неверное решение и принять его за истинное.

По этой причине в справочных системах математических пакетов подпрограммы представляются как «черные» или «серые» ящики.

15.2. Понятие о математических пакетах

Еще недавно для решения своей математической задачи пользователь должен был не только разбираться в математике, но и освоить работу на компьютере, знать один (а то и несколько) языков программирования, освоить сложные и подчас капризные численные методы, провести трудоемкое и ответственное тестирование. Сейчас к услугам тех, кто не может или не хочет программировать, имеются готовые (но не всегда доступные как в прямом, так и в экономическом смысле) библиотеки научных программ, электронные справочники и т. д., а также *пакеты прикладных программ* (ППП) — важнейший инструмент выполнения типовых расчетов в достаточно формализованных областях знания. Излагаемые ниже сведения полезны сами по себе и имеют дополнительной целью побудить читателя к созданию *собственных* пакетов, как это произошло с автором ([73]).

Каждый такой пакет является комплексом взаимосвязанных прикладных программ, специальных и общих средств системного обеспечения. Хороший пакет позволяет проделать, не выходя из него, всю нужную работу или весьма значительную ее часть:

- исследование проблемы (в том числе в символьной форме),
- анализ данных,
- проверку существования решения,
- моделирование,
- тестирование,
- оптимизацию,

- построение графиков,
- документирование и оформление результатов,
- подготовку презентации.

Пакет позволяет сосредоточить основное внимание на *существе* проблемы, оставляя в стороне технику классической математики, тонкие детали вычислительных методов, «маленькие хитрости» языков программирования и команд операционной системы. Однако всегда следует считаться с возможностью напороться именно на них.

Большинство ППП перед использованием требуют проведения *инсталляции*: настройки пакета на конкретные условия эксплуатации (конфигурация аппаратных средств, размещение системных файлов, определение инсталлируемого подмножества пакета, выбираемые по умолчанию опции).

Функциональная часть пакета — это совокупность базисных процедур, комбинированием которых может быть получено решение достаточно широкого круга задач предметной области. *Модульное* построение функциональной части обеспечивает:

- обозримость, структуризацию и систематизацию ее;
- возможность параллельной разработки модулей;
- сокращение общего объема программного текста;
- гибкость применения;
- уменьшение потребности в оперативной памяти избирательной загрузкой только необходимых в частном случае модулей;
- уменьшение затрат на трансляцию.

Общие требования к функциональной части пакета:

- членами библиотеки должна быть перекрыта вся область возможных запросов;
- алгоритмы должны не конкурировать, а *дополнять* друг друга;
- библиотечная реализация должна сочетать предпринимаемые по умолчанию стандартные действия с разумным выбором опций пользователем;

- библиотека должна содержать общий указатель, который классифицирует категории задач. Необходимы схемы выбора, которые разветвляются в зависимости от данных пользователем ответов на простые вопросы, касающиеся его задачи. Конечный элемент дерева выбора — имя рекомендуемой процедуры. При необходимости пользователь должен иметь возможность составить и применить вспомогательные процедуры.

Специальные средства системного обеспечения — это общий для функциональных модулей специфический механизм приведения их в действие. К таким средствам можно отнести:

- входной язык пакета;
- транслятор (обычно — интерпретирующего типа) написанных на нем заданий;
- архивы программ и текстовых вставок;
- банк исходных данных и результатов счета;
- монитор интерактивного общения;
- справочную систему (`Help`) .

При работе с ППП непосредственно и/или через специальные системные средства используется и *общее программное обеспечение* вычислительной установки:

- компиляторы, макрогенераторы, редакторы связей, загрузчики;
- текстовые редакторы;
- наборы шрифтов;
- файловые системы.

15.3. Пакеты и пользователь

Освоение машинной математики может создать у пользователя иллюзию освоения самой математики. Однако следует понимать, что *инструмент не заменяет компетентность*. Никакие красочные меню

не освобождают пользователя от понимания сути математических команд и методов, реализованных в таких системах. Этим математические системы принципиально отличаются от текстовых и графических редакторов. Если пользователь не знает, что такое матрица, то пакет программ матричной алгебры ему не поможет. Если он пытается применить численный метод к вычислению несобственного интеграла, то рискует получить решение, сколь угодно далекое от истинного, или вообще ничего не получить. Любой пакет широкого применения по необходимости реализует универсальный (хочется сказать «тупой») подход. В частности, все математические пакеты строят графики с постоянным шагом аргумента, тогда как для полноценного исследования кривой нужны характерные точки и свойства (корни, экстремумы, точки перегиба, асимптоты).

К сожалению, популярные компьютерные журналы изобилуют теоретически несостоятельными статьями самонадеянной молодежи, возомнившей, что она «ухватила господа бога за бороду». Как отмечается в [28, с. 20], «многие рекомендации "Byte"и "Personal Computing" из области численных расчетов носят дилетантский характер». Это относится и к другим областям компьютерной *науки* (не путать с технологией!). К примеру, молодые специалисты по компьютерным сетям убеждены в том, что информационные технологии вполне заменяют знание основ теории очередей. Автор статьи [56] М. Кульгин полагает, что «аналитик может провести анализ очередей в заданной сетевой структуре, используя уже готовые таблицы очередей или простые компьютерные программы, которые занимают несколько строк кода», и далее демонстрирует математическую безграмотность и полное непонимание проблемы. Формула $\Pr[\Theta \geq \vartheta] = 100e^{-\mu\vartheta}$ из-за невесть откуда появившейся сотни может давать значения вероятностей до 100 включительно. Далее утверждается, что «μ — уровень обслуживания, в данном случае равный коэффициенту загрузки ρ». Поскольку ρ — величина безразмерная, показатель степени приобретает размерность времени — совершенно вопиющий абсурд. На самом деле μ есть *интенсивность обслуживания* (величина, обратная его средней длительности).

Утверждается, далее, что «если существует среда с разделяемыми каналами связи, то производительность такой системы обычно изменяется по экспоненциальному закону при увеличении нагрузки . . . При дальнейшей загрузке системы ее производительность будет резко снижаться». На самом деле производительность системы измеряется не временем ответа, а числом заявок, обслуженных в единицу времени. Она равна интенсивности входящего потока — следовательно, с ростом коэффициента за-

грузки ρ будет *возрастать*, пока он не достигнет единицы. После этого производительность системы будет постоянна и равна ее максимальной производительности. Экспоненциальному же закону при докритическом режиме подчиняется не производительность системы, а распределение *времени пребывания* заявки в ней.

Возвращаясь к проблеме использования математических библиотек и пакетов, отметим, что только специалист может сделать разумные допущения; провести линеаризацию; отсеять второстепенные факторы; ввести хорошие начальные приближения; выбрать из альтернативных подпрограмм наиболее подходящую в конкретном случае; при необходимости — модифицировать подпрограмму в направлении расширения области применения, лучшего приспособления к конкретной ситуации, снятия ранее наложенных ограничений, уменьшения объема вычислений и т. п.

В настоящее время относительно доступны пакеты компьютерной алгебры Mathematica, Maple, MatLab, MathCAD, Derive и Scientific WorkPlace. Из них два первых ориентированы на профессиональных математиков: подавляют обилием предоставляемых возможностей (вспомним любимое В. И. Лениным выражение «embarras de richesse»), относительно сложны в использовании. Предельная общность постановки задачи (все параметры — комплексные) ограничивает возможности получения решения и затрудняет его интерпретацию.

MatLab ориентирован на работу с матрицами и решение задач теории автоматического управления и обработки сигналов.

MathCAD и Derive наиболее просты в применении и вполне обеспечивают типовые потребности студента. К этому же классу пакетов относится Eureka.

Scientific WorkPlace является интегрированной средой подготовки математических рукописей в системе рукописей LaTeX с возможностью одновременного выполнения символьных и числовых выкладок.

Относительно подробный обзор этих пакетов и их сопоставление приведены в [82]. Литература по каждому из них исчисляется десятками наименований. Исключением является Scientific WorkPlace — см. [22].

Знамение эпохи — появление *учебников по вычислительной математике*, ориентированных на упомянутые пакеты. В частности, все примеры «курса лекций» [76] записаны на языке MatLab, а пять глав руководства по интерполяции [75] посвящены решению соответствующих

задач в разных пакетах. На наш взгляд, работа с пакетами целесообразна прежде всего там, где числовые результаты нужны безотносительно к способу их получения (выполнение курсовых работ по электротехнике, сопромату, теории управления и пр.). Сама по себе — при работе с функциями пакета как «черными ящиками» — она *не способствует усвоению вычислительной математики*. Тем не менее постановка практических занятий типа

- исследования областей сходимости итераций,
- определения скорости сходимости,
- сравнения простых итераций с зейделевыми и сверхрелаксацией,
- сравнения квадратурных формул при одинаковом числе узлов,
- сопоставления методов вычисления несобственных интегралов

и т. п. представляется полезной. Проблемой здесь является не применение пакетов, хорошо обеспеченное литературой, а продуманный подбор заданий вышеуказанного содержания.

Учитывая изложенное, ниже дается лишь краткий *обзор возможностей* типичного математического пакета — Maple.

15.4. Введение в Maple

Maple — это среда для выполнения символьных, численных и графических вычислений профессиональными математиками, разработанная фирмой Waterloo Maple Software (University of Waterloo, Канада) и Высшей технической школой в Цюрихе. Она воплотила колоссальный математический потенциал, включает широчайший арсенал средств («от элементарной арифметики до общей теории относительности») и активно используется научной общественностью, о чем свидетельствуют многочисленные ссылки в научных статьях. Выпускается журнал *Maple Technical Newsletter* и даже газета. Подмножества символьного инструментария Maple являются существенными элементами интегрированной системы *Scientific WorkPlace*, а также популярных пакетов MathCAD и MatLab. На 2-й Всероссийской конференции по имитационному моделированию (Санкт-Петербург, октябрь 2005 г.) докладывалась выполненная

в ВЦ РАН средствами Maple аналитическая модель российской теневой экономики.

Приводимое здесь краткое описание этого пакета опирается на счастливую возможность описания программных средств сразу после изложения в предшествующих главах их математических основ.

15.4.1. Входной язык

В строках рабочего листа после приглашения $>$ набираются команды Maple, состоящие главным образом в вызове ее процедур. Всего таких процедур около трех тысяч, и даже их перечисление далеко выходит за рамки данной книги. Наиболее употребительные из них будут прокомментированы ниже.

Для обращения к контекстно зависимому `help`'у необходимо выделить в рабочем листе ключевое слово (имя процедуры или оператор) и нажать [F1]. В ответ высветится соответствующая страница `help`'а и ее место в иерархии справочных сведений (узлы пути от корня дерева). Это позволяет при необходимости ознакомиться и со смежными проблемами.

Maple не предусматривает обязательного объявления типов переменных. Рациональные числа в общем случае имеют вид дробей с целыми компонентами (при выводе — несократимых). Вещественное число записывается в форме `<мантисса>*10^(<порядок>)`. Все переменные по умолчанию считаются *комплексными*. Имена (идентификаторы) записываются как во всех алгоритмических языках.

Из элементарных объектов могут быть сформированы составные: множества, списки, массивы, матрицы и векторы. Решения уравнений обычно представляются в виде множеств.

Для формирования *выражений* используются знаки операций `+, -, *, /, ^,!`. Средством запоминания результатов вычисления является оператор присваивания, записываемый через `:=`.

Функциями Maple называются процедуры, выполняющие системные и вычислительные обязанности. Они делятся на внутренние (встроенные, т. е. входящие в ядро Maple), внешние (автоматически считываемые из библиотеки после команды `readlib <имя>`) и «пакетные», для применения которых нужно предварительно указать в команде `with` имя содержащего их пакета.

Перечислим наиболее употребительные математические функции:

BesselI	BesselJ	BesselK	BesselY	Beta	GAMMA
Si	arccos	arccosh	arcsec	arcsech	arcsin
arcsinh	arctan	arctanh	bernoulli	chebyshev	collect
combine	cos	cosh	dsolve	erf	evalc
evalm	exp	factor	frac	fsolve	gcd
isprime	ithprime	laplace	latex	limit	linalg
ln	log	msolve	plot	plot3d	product
rand	round	signum	simplify	sin	sinh
solve	sqrt	sum	tan	tanh	trace

Бóльшая часть этих функций легко узнаваема.

15.4.2. Ввод в стандартной символике

В новых версиях Maple появилась возможность «кнопочного ввода» в привычной математической символике. Для ее реализации нужно через меню `View/Palettes` вызвать на экран панели `Symbol` с греческими буквами и специальными символами, `Expression` с шаблонами операторов и при необходимости `Matrix` — с шаблонами векторно-матричных объектов.

При щелчке мышью по панели соответствующий символ появляется в позиции вставки на рабочем листе. Подлежащие заполнению слоты оператора (`placeholders`) указываются вопросительными знаками. Слот заполняется с помощью клавиатуры или панели символов. Необходимые навыки приобретаются достаточно быстро; однако пользователь с навыками процедурного программирования, вероятно, предпочтет работу в нотации Maple.

Набранная команда выполняется после нажатия [Enter] или «восклицательной» кнопки панели.

15.4.3. Решение уравнений и систем уравнений

Данная задача решается оператором `solve(<eqns>,<vars>)`. При нескольких неизвестных уравнения и неизвестные должны представляться как множества, т. е. в фигурных скобках. Соответственно будут организованы и результаты. Частные виды уравнений или требований к

решениям обрабатываются специальными процедурами решения уравнений:

`dsolve` — дифференциальных,

`fsolve` — с плавающей точкой,

`isolve` — в целых числах,

`msolve` — по модулю,

`rsolve` — рекуррентных,

`linalg[linsolve]` — матричных.

В последнем случае используется «пакетная» процедура, которая должна иметь префиксом имя пакета.

При вызове процедур решения уравнений могут указываться дополнительные опции. Для `fsolve` ими могут быть `complex` (искать комплексные корни), `x=a..b` (искать корень в заданном интервале).

В Maple различаются массивы, матрицы и векторы. Наиболее общей из структур этого класса является массив. Он создается командой вида

```
> array(indf,<границы>,<список>)
```

Массив может иметь произвольные размерность и границы (списки граничных пар). Заключительный (необязательный) список — это список начальных значений в одномерном случае и список таких списков — в двумерном. Означивание можно выполнить и последующими операторами.

Указатель `indf` определяет специальную структуру массива (`symmetric`, `antisymmetric`, `sparse`, `diagonal`). Учет структуры уменьшает трудоемкость обработки массива и позволяет доопределять его системными средствами.

В матричном выражении допустимы суммирование, перемножение, вычисление целых степеней и покомпонентные операции. Единичную матрицу следует записывать как `&*()` или использовать собственное обозначение — например, `E` — введенное оператором вида `>alias(E=&*()):` (буква `I` занята под мнимую единицу). Матричные произведения должны записываться в форме `A&*B&*C` или `&*(A,B,C)`.

15.4.4. Символические вычисления

Допущения. В ряде случаев (например, при вычислении определенных интегралов или бесконечных сумм) вид результата и даже возможность его получения могут зависеть от области значений аргументов или параметров. Область задается командой вида

```
assume(p,<свойство>)
```

где свойством могут быть `integer, real, RealRange(a,b), positive, negative`. В роли «свойства» может выступать заключенное в скобки отношение или перечень отношений (подразумевается логическое произведение отношений). Имеется возможность введения пользовательских типов свойств и иерархий типов с наследованием свойств «предков».

Эквивалентные преобразования.

`subs` выполняет замену переменных во втором аргументе согласно первому;

`expand` в простейшем случае разворачивает произведения и функции сложных аргументов в суммы;

`factor` выполняет противоположное преобразование;

`normal` использует основные алгебраические формулы для приведения выражений к форме полиномов или дробей, числитель и знаменатель которых — взаимно простые многочлены с целыми коэффициентами;

`simplify` позволяет упрощать выражения — в особенности при указании области поиска решения. В качестве таких указателей назовем `GAMMA, exp, ln, sqrt, trig, power`, а также спецификаторы класса результирующих значений;

`collect` группирует члены выражения по степеням подвыражения — в частности, многочлена от многих переменных по степеням одной из переменных;

`combine` пытается объединить показатели степенных функций-сомножителей и понизить степени тригонометрических функций переходом к кратным углам.

Суммы и произведения. Команда `sum(f,'k'=m..n);` предназначена для «символического» суммирования. Предполагается, что f

зависит от k. Аналогично используется и команда вычисления произведения `product`.

Аппроксимации. В пакете `numapprox` предлагаются 13 видов аппроксимации функций. В качестве иллюстрации мы рассмотрим примеры на два класса аппроксимаций.

Функция `minimax(f,x=a..b,[m,n],w,'maxerror')` обеспечивает дробно-рациональную аппроксимацию $f(x)$ на $[a,b]$ с весом $w(x)$, дающую минимум максимальной ошибки (оценка последней присваивается переменной с именем `maxerror`). Целые m,n суть максимальные степени числителя и знаменателя соответственно.

Функция `pade(f,x=a,[m,n])` служит для подбора аппроксимации Паде функции $f(x)$. Эта аппроксимация дает дробь, разложение которой в ряд Тейлора имеет минимальное отличие от непосредственного разложения $f(x)$.

Операции математического анализа. Пределы вычисляются командой `limit(f,x=a[,dir])`. Предельная точка может находиться в бесконечности. Третий (необязательный) параметр может принимать значения `left`, `right`, `real`, `complex`.

Функция `diff(f,x1$k_1,...,xn$k_n)` обеспечивает дифференцирование f с заданными для каждой из перечисленных переменных после `$` кратностями (единичную кратность можно не указывать).

Функция `series(f,x=a,n)` дает разложение $f(x)$ в степенной ряд в окрестности $x=a$ с учетом членов до n-го порядка включительно.

Оператор вида `int(f,x)` вычисляет неопределенный интеграл, а `int(f,x=a..b)` — определенный. Численное интегрирование организуется также оператором

```
evalf(int(f,x=a..b,digits,flag))
```

Здесь сначала выполняется символьное интегрирование, а затем получение численного эквивалента. Третий и четвертый параметры не обязательны: `digits` определяет число верных знаков результата, `flag` — метод интегрирования. Если внутри интервала подынтегральная функция имеет особенности, то делается попытка их устранения применением аналитических преобразований (замена переменных, «вычитание особенности», разложение в ряд).

Решение дифференциальных уравнений.

Оператор вида

```
dsolve(<deqns>,<vars>,<opns>)
```

служит для решения обыкновенных дифференциальных уравнений. Последние должны содержать производные в явной форме (запись в

дифференциалах не допускается). Укажем смысл и основные варианты его параметров.

deqns содержит исходные уравнения (системы уравнений) и — при необходимости — начальные условия к ним. Производные записываются в форме diff(y(x),x). После независимой переменной через $ может быть указана кратность высших производных. Начальные условия для производных записываются применением к функции оператора D (например, первая и вторая производные z в нуле — как $D(z)(0)$, $D(D(z))(0)$ соответственно).

vars — это список искомых переменных. Аргументы функций обязательно указывать только здесь и при записи производных.

opns задает опции в формате <имя>=<значение>. Важнейшей из них является тип решения type $\in$ {exact, series, numeric}.

Возможности исследования и решения обыкновенных дифференциальных уравнений расширяются пакетом DEtools. В частности, в него входит классификатор уравнений odeadvisor.

Численное решение дифференциальных уравнений организуется указанием type=numeric. По умолчанию оно проводится методом Рунге—Кутты—Фельберга (rkf45). Предлагаются еще 7 методов, среди них разработанный в Ливерморской лаборатории метод решения «жестких» уравнений. Решение по умолчанию выводится в виде таблицы, каждая строка которой содержит текущие значения независимой и зависимых переменных. Таблице предшествуют заголовки колонок. Для графического представления результатов решения дифференциальных уравнений существует специальный подпакет.

15.4.5. Maple и Фортран

Система Maple не приспособлена для массовых трудоемких расчетов — их лучше выполнять в обычных системах программирования. Имеется возможность автоматического переноса апробированных средствами Maple подходов в Фортран-программы. Для этого служит команда

```
> fortran(expr,<список опций>)
```

где expr — выражение или список выражений. Все опции (имя выходного файла, необходимость экономии подвыражений, разрядность счета) не обязательны и могут приниматься по умолчанию.

15.4.6. Графические средства

Набор графических возможностей Maple поистине уникален. Достаточно сказать, что при построении двумерных графиков Maple поддерживает 15 систем координат, а в трехмерном случае — 31 (с возможностями преобразования из одной системы в другую). Maple гарантирует отсутствие наложения цифр и букв на кривые и сетку. Smart-графика позволяет быстро строить пробные графики с их последующей «доводкой».

15.4.7. Дополнительные пакеты

В состав Maple входят дополнительные пакеты:

DEtools	Domains	GF	GaussInt
LREtools	combinat	combstruct	difforms
finance	genfunc	geometry	Groebner
group	inttrans	liesymm	linalg
logic	networks	numapprox	numtheory
orthopoly	padic	plots	plottools
powseries	process	simplex	stats
student	sumtools	tensor	totorder

Некоторые из подпакетов, как это видно уже из их названий, нужны только профессиональным математикам. Отметим, в частности, наличие средств работы с абстрактными операторами — назначение и проверку свойств линейности, коммутативности, ассоциативности, определения обратного оператора, неподвижной точки и т. п. Еще раз напомним, что для вызова процедур какого-либо пакета следует открыть к нему доступ командой

```
> with(<имя_пакета>):
```

(при завершении команды точкой с запятой выводится перечень процедур пакета).

Приведем обзор содержания тех пакетов, которые могут быть интересны (будущим) инженерам.

Student. Этот пакет содержит «набор юного джентльмена» и обеспечивает редкую возможность получения необходимого для предъявления преподавателю *пошагового* решения типовых математических задач — расчет сумм, произведений, пределов, интегралов и т. д. Имеется инструментарий для отделения корней уравнений, исследования кривых,

нахождения экстремумов и реализации простейших методов интегрирования (подстановкой, по частям, численных).

Linalg. Пакет включает свыше ста функций формирования разнообразных матриц специального вида и операций над ними. В дополнение к ранее отмеченным возможностям он может:

- формировать случайные, блочные, ленточные, гильбертовы, эрмитовы матрицы;
- вычислять нормы, собственные значения и векторы, обратные матрицы, якобианы вектор-функций, гессианы;
- проводить ортогонализацию;
- выполнять скалярное и матричное умножение;
- находить базис;
- решать недоопределенные и переопределенные системы линейных алгебраических уравнений и т. д.

Пакет включает в себя ядро, состоящее из базисных подпрограмм нижнего уровня — в частности, выполняющих LU-разложение.

Simplex. Сюда входят основанные на симплекс-методе процедуры решения задачи линейного программирования — как в целом, так и отдельных элементов метода (переход к стандартной постановке, нахождение базиса, его одновекторное изменение и т. д.).

Inttrans. В пакет входят процедуры вычисления интегральных преобразований Фурье (общего, $\sin$- и $\cos$-), Ганкеля, Гильберта, Лапласа и Меллина, а также соответствующих обратных преобразований.

Stats. Пакет ориентирован на типовые задачи статистической обработки данных. В него включены семь субпакетов. Укажем назначение первых шести:

`anova` — проверка гипотез об однородности двух статистических выборок;

`fit` — построение регрессии (выбор параметров заданного типа зависимости $y(x)$ методом наименьших квадратов);

`random` — генерация серий случайных чисел с требуемыми законами распределения (равномерный, бета-, гамма-, нормальный, χ^2, Стъюдента);

`statevalf` — построение плотности, кумулянты и обратной ей функции для непрерывного распределения и их аналогов — для дискретного;

`statplots` — построение графиков и гистограмм;

`transform` — предварительная обработка данных (упорядочение, сортировка по классам, вычисление заданной функции от каждого элемента ряда и т. п.).

Седьмой субпакет — `describe` — используется для получения числовых характеристик статистического ряда и заслуживает более подробного рассмотрения. Он позволяет вычислять:

- моду и размах распределения;
- обычное, геометрическое и гармоническое средние значения;
- сечения статистического распределения (медиану, квартили и децили, а также квантили для произвольного аргумента);
- высшие моменты относительно произвольной точки;
- производные от них характеристики: дисперсию D, среднеквадратическое отклонение, асимметрию и эксцесс.

15.5. Математические библиотеки IMSL

Снова напомним, что лучшим способом изучать вычислительную математику является программирование, а лучший язык программирования численных методов — это Фортран.

В состав профессиональной версии Фортрана 90 входят библиотеки IMSL, содержащие около тысячи программ на Фортране 77 в объектной форме. Большая часть процедур имеется в вариантах с одинарной и двойной точностью. Каталоги библиотек можно просмотреть в соответствующем разделе систематической части `help`'а IMSL. Библиотек имеется две: 1) прикладная математика и специальные функции; 2) статистика. Для подготовки их использования в вызывающую программу включается оператор

```
use numerical_libraries
```

обеспечивающий интерфейс с членами библиотеки и контроль синтаксической корректности вызовов.

Выполнение файла dfvars.bat (см. директорию ...DF98\BIN) устанавливает переменные окружения, указывающие, в частности, путь к этим библиотекам. Для контроля следует просмотреть include- и library-пути (цепочка Tools\Options \Directory\Show Directories).

Упоминаемые в операторах вызова члены библиотеки подключаются к программе на этапе линкования.

Описания раздела библиотеки начинаются со структурированного перечня входящих в него процедур. Для имен процедур используются стандартные соглашения. «Корень» имени имеет мнемоническое (применительно к английскому языку) значение: например, GQRUL — гауссово квадратурное правило. Префикс S означает одинарную точность, D — двойную, C — комплексные одинарной точности, Z — двойной. Таким образом получают имена целые *семейства* процедур.

За оглавлением следуют общие рекомендации по разделу. Например, по «Линейным решателям»:

- сообщается, что учет специальных свойств матриц позволяет радикально уменьшить время счета и потребность в памяти;
- определяются «коренные» обозначения процедур (RG — общий случай вещественной, CG — то же для комплексной, TR — трехдиагональная вещественная, RB — вещественная ленточная, XG — разреженная вещественная);
- приводятся структурные схемы факторизации и решения линейных систем;
- на тот же счет даются текстовые рекомендации;
- дополнительно обсуждаются частные проблемы (несколько правых частей, расчет определителя, итеративное решение и итеративное уточнение решения, обращение матрицы, проблема сингулярности в ее компьютерном аспекте, матрицы специального вида, QR-декомпозиция).

Далее даются общие описания семейств процедур. По каждому семейству указываются: корневое имя или набор имен; назначение; обращение (форма вызова и список аргументов); спецификация аргументов (смысл и статус — входной, выходной, изменяемый); рекомендации по использованию и выделению дополнительной памяти; идея алгоритма

со ссылками на источники более детальной информации[1]; пример применения; результаты расчета примера. В предисловии к описанию IMSL отмечается, что простейший способ ее применения — копирование подходящего примера.

Приведем каталоги основных библиотек IMSL:

Математическая. Линейные системы. Задачи на собственные значения. Интерполяция и аппроксимация. Интегрирование и дифференцирование. Дифференциальные уравнения. Интегральные преобразования. Нелинейные уравнения. Оптимизация. Основные матрично-векторные операции.

Специальные функции. Элементарные функции. Тригонометрические и гиперболические функции. Гамма-функция и связанные с ней. Функция ошибок (нормальное распределение) и связанные с ней. Функции Бесселя. Функции Кельвина. Функции Эйри. Эллиптические интегралы и смежные воппросы. Эллиптические и связанные с ними функции. Функции распределения вероятностей и им обратные. Функции Матье. Разные функции.

Статистика. Основы статистики. Регрессия. Корреляция. Анализ дисперсии. Категорийный и дискретный анализ данных. Непараметрическая статистика. Критерии согласия и случайности. Анализ временных рядов и прогнозирование. Ковариации и факторный анализ. Дискриминантный анализ. Кластерный анализ. Выборки. Выживание и надежность. Многомерное шкалирование. Оценка плотностей и вероятностей. Построчная принтерная графика. Функции распределения вероятностей и им обратные. Генерация случайных чисел. Утилиты. Вспомогательные математические средства.

Для лучшего представления о содержательности *разделов* приведем характеристику процедур раздела «Интегрирование и дифференцирование»:

Одномерное интегрирование. Общая адаптивная, особенность на границе. Общая адаптивная. Общая адаптивная, разрывы. Общая, бесконечный интервал. Адаптивная осциллирующая. Адаптивная (Фурье). Адаптивная с алгебраическим весом, разрывы. Адаптивная взвешенная, главное значение по Коши. Общая неадаптивная.

Многомерные квадратуры. Двумерная квадратура (повторный интеграл). Адаптивная N-мерная квадратура по гиперпрямоугольнику.

[1] Описания алгоритмов рассчитаны на профессионалов и страницы учебников по вычислительной математике не повторяют.

Правила Гаусса и трехчленные рекуррентные соотношения. Правило Гаусса для классических весов. Правило Гаусса для рекуррентных коэффициентов. Рекуррентные коэффициенты для классических весов. Рекуррентные коэффициенты из квадратурного правила. Квадратурное правило Фейера.

Дифференцирование. Получение первой, второй или третьей производной.

Несомненно, что названные библиотеки являются ценнейшим инструментом, использовать который может и должен каждый работающий на Фортране программист.

В заключение приведем программу вычисления определенного интеграла из разд. 12.8.2 с применением библиотеки IMSL.

```
program intsiml
   use numerical_libraries
   real*8         a,b,errabs,errel,errest,f,result,z
   external       f
   z=exp(1d0)-1d0 ! Точное значение интеграла
   a=0d0; b=1d0
   errabs=0d0; errel=1d-9
   call dqdags &
        (f,a,b,errabs,errel,result,errest)
   print *, 'result = ',result ! точный: 1.718281828459045
   print *, 'errest = ',errest !
   print *, 'dres = ',result-z !
end program intsiml

real*8 function f(x)
   real*8 x
   real*8, intrinsic :: dexp
   f=dexp(x)
end function f
```

Вызываемая процедура реализует адаптивную схему вычисления интеграла по 21-точечному правилу Гаусса—Кронрода (см. разд. 12.9) на каждом подынтервале разбиения исходного участка и допускает наличие сингулярностей на границах участка интегрирования. Имена фактических параметров совпадают с именами формальных. Из них `errest` есть

оценка абсолютной погрешности результата, а смысл остальных очевиден. Получены следующие результаты:

```
result= 1.71828182845905
errest= 1.907676048750246e-014
dres  = 2.220446049250313e-016
```

В данной программе поучительна работа с данными двойной точности (в частности, использование специфического имени функции вычисления экспоненты).

Специально работе с IMSL посвящена книга [9].

15.6. Библиотека Fortran 90 MP

Названная библиотека, в отличие от IMSL, написана уже на Ф90 и обеспечивает более простой и надежный вызов процедур — при сокращенном списке аргументов. В ее состав входят следующие разделы:

Линейные решатели.
Задачи на собственные значения.
Преобразование Фурье.
Сглаживание сплайнами кривых и поверхностей.
Утилиты.
Операторы и семейства функций.
Утилиты LAPACK для параллельных решателей.
Дифференциальные уравнения в частных производных.
Обработка ошибок.

Описания подпрограмм даются применительно к их *семействам* (generics); соответственно и вызывать их можно по мнемоническому родовому имени с суффиксом _gen. Например, для решения системы линейных уравнений $Ax = b$ независимо от формата и вещественности исходных данных можно воспользоваться оператором

```
call lin_sol_gen(A,x,b)
```

Предварительно следует сослаться на модуль, содержащий *интерфейс к семейству* (не к пакету в целом):

```
use lin_sol_gen_int
```

Особого внимания заслуживает глава «Операторы и семейства функций» (Operators and Generic Functions). Ее содержание поз-

воляет писать программы для задач линейной алгебры в стиле, близком к математической нотации. При этом получаются короткие и легко читаемые специалистами программы. Разработчики старались избегать вызовов и имен процедур, зависящих от типа параметров.

Модуль `linear_operators` определяет необходимые операторы и семейства функций, причем для многих подзадач предложена параллельная реализация.

Рассмотрим фрагмент Ф90-кода, который решает систему линейных алгебраических уравнений $Ay = b$ со случайными коэффициентами, а затем для контроля вычисляет вектор невязок $r = b - Ay$. Первая задача $y = A^{-1}b$ решается с помощью оператора «`.ix.`», невязка вычисляется посредством «`.x.`». Пример:

```
program
   use linear_operators
   integer, parameter :: n=3
   real A(n,n),y(n),b(n),r(n)
   A=rand(A); b=rand(b)
   y=A.ix.b
   r=b-A.x.y
end
```

При работе с комплексными объектами все ссылки на транспонированную матрицу A^T заменяются на присоединенную (комплексно сопряженную) $\bar{A}^T \equiv A^* \equiv A^H$. Аналогичные замены выполняются в связи с понятием унитарных матриц. Приведем таблицу операторов библиотеки Ф90:

Таблица 15.1. Линейные операторы

Операторная запись	Программный код
A^{-1}	`.i.A`
A^T, A^*	`.t.A`, `.h.A`
$A^{-1}B$	`A.ix.B`
BA^{-1}	`B.xi.A`
A^TB, A^HB	`A.tx.B`, `(.t.A).x.B`
A^TB, A^HB	`A.hx.B`, `(.h.A).x.B`

Для сравнения запишем альтернативу для A^HB непосредственно на Ф90: `matmul(conjg(transpose(A)),B)`.

В заключение приведем таблицу 15.2 матричных и вспомогательных функций из этого раздела библиотеки Ф90. Необязательные операнды в таблице опущены.

Работа с этой библиотекой требует строгого соответствия типов аргументов типу формальных параметров; для облегчения такого контроля настоятельно рекомендуется использовать в вызывающей программной единице `IMPLICIT NONE`.

Таблица 15.2. Реализация матричных функций

Операция	Вызов
$A = USV^T$	`S=SVD(A)`
$AV = VE$, $AVD = BVE$ $(AW = WE)$, $AWD = BWE$	`E=EIG(A)`
$A = R^T R$	`R=CHOL(A)`
$A = QR$, $Q^T Q = I$	`Q=ORTH(A)`
Унитарная матрица	`U=UNIT(A)`
Определитель	`F=DET(A)`
Ранг	`K=RANK(A)`
$p_1 = \max\limits_j \left(\sum\limits_{i=1}^{m} \lvert a_{i,j} \rvert \right)$ $p_2 = s_1$ — наибольшее сингулярное значение $p_3 = \max\limits_i \left(\sum\limits_{j=1}^{n} \lvert a_{i,j} \rvert \right)$	`P=NORM(A,i)`
$s_1/s_{\mathrm{rank}A}$ — число обусловленности	`C=COND(A)`
$Z = I_N$ — единичная матрица	`Z=EYE(A)`
Диагональ	`P=DIAG(A)`
Прямое преобразование Фурье	`Y=FFT(X)`
Обратное преобразование Фурье	`X=IFFT(Y)`
Случайное заполнение	`A=RAND(A)`

15.7. Compaq Extended Mathematical Library

Эта библиотека (CXML) может быть затребована при `Custom`–инсталляции Compac Visual Fortran'a. Библиотека разработана консорциумом правительственных учреждений и университетов США. В нее входят следующие разделы:

- Базовые программы линейной алгебры, разделенные на три уровня (вектор — вектор, матрица — вектор, матрица — матрица).
- Векторные операции (для покомпонентных операций матрицу выгоднее рассматривать как вектор).
- Прямые и итеративные решатели систем линейных алгебраических уравнений с матрицами специальной структуры (симметричными, кососимметричными, эрмитовыми, ленточными, треугольными и т. п.).
- Стандартный пакет LAPACK решения систем линейных уравнений и задач на собственные значения, по своим возможностям и эффективности превосходящий известные LINPACK и EISPACK.
- Подпрограммы обработки сигналов: 1-, 2- и 3-мерное быстрое преобразование Фурье, свертки, корреляционный анализ, цифровые фильтры.
- Утилиты: генераторы случайных чисел, векторные и матричные функции, программы сортировки.

Повышенную эффективность процедур CXML разработчики аргументируют рациональным использованием всей иерархии запоминающих устройств:

- повторным обращением к данным в *регистровой* памяти — без промежуточных обменов со следующим уровнем;
- локализацией вычислений в кэш-памяти: алгоритмы работают с подматрицами исходных объектов, что сводит к минимуму страничный обмен.

Большую роль играют также многоступенчатый выбор оптимальных подалгоритмов и реорганизация циклов для преимущественного использования единичных приращений адресов (например, через «векторизацию»).

В PDF-документации к процедурам по каждому разделу имеются: обстоятельное введение в структуры данных; соглашения о мнемонике имен процедур и порядке следования параметров; сводный каталог процедур и порядок использования каждой из них с примерами вызова; список литературы. Имена процедур начинаются с префикса, задающего тип данных аналогично IMSL (S, D, C, Z). Две последующих буквы

определяют тип матрицы (например, GE — общий случай, GB — общая ленточная, GT — общая трехдиагональная, SB — симметричная ленточная, ST — симметричная трехдиагональная, HB — эрмитова ленточная, PO — положительно определенная). Наконец, две последних буквы определяют решаемую задачу: SV — решение системы линейных уравнений, EV — задача на собственные вектора и собственные значения, SVD — сингулярное разложение.

Тексты процедур, однако, приводятся лишь в немногих случаях: для генераторов равномерно распределенных случайных чисел и сортировок.

Для работы с процедурами CXML из среды MDS следует в меню Settings/Link выбрать категорию Input и добавить имена используемых CXML-библиотек, разделенные пробелами. Подключение их при работе с консоли обеспечивается командой вида

```
DF main.f90 %link_f90%
```

15.8. Numerical Recipes

В профессиональной работе с математическими библиотеками часто возникает желание заглянуть «внутрь» используемых процедур в целях:

- их адаптации к отдельным классам задач;
- улучшения их характеристик (расширения области применения, уменьшения трудоемкости, повышения точности);
- выяснения причин отклонений результатов от ожидаемых;

наконец, из педагогических соображений и просто здоровой любознательности. Такую возможность дают входящие в состав четвертой версии Фортрана PowerStation «Численные рецепты» (Numerical Recipes). В гипертекстовом справочнике, имеющем алфавитный и систематический разделы, содержатся тексты свыше 600 процедур и примеров их применения на Фортране 77 (напомним, что составленные на нем программы совместимы снизу вверх с Ф90). Все процедуры написаны в расчете на стандартную точность; в пакет входит процедура NR2DP, формаль-

но преобразующая их к двойной точности[2]. В сопровождающей подборке рецензий содержащая «Рецепты» книга (издания Кембриджского университета) аттестуется как «воистину золотая жила» и «классический источник», доставляющий «пользу, удовольствие и радость» всем программирующим на Фортране и содержащий массу ценных советов и деталей. Другой рецензент заявляет, что всякий вычислитель получит пользу, литератор — удовлетворение, а человек с юмором — радость. К сожалению, упомянутая книга поставляется на отдельной дискете, а в электронной документации собственно FPS4 ни к процедурам, ни к вызывающим их программам нет *никаких* комментариев. Это заметно ограничивает «удовольствие и радость» от «Рецептов».

Работать с «Рецептами» удобно через файл `Numerical Recipes Subroutines and Functions by Chapter`. Приведем соответствующее

Оглавление. Разное (вплоть до астрологии). Решение линейных алгебраических уравнений. Интерполяция и экстраполяция. Интегрирование функций. Вычисление функций (аппроксимации). Специальные функции. Случайные числа (различные законы распределения). Сортировки. Решение уравнений и систем уравнений. Минимизация и максимизация функций (одномерный поиск, линейное и нелинейное программирование). Задачи на собственные значения. Быстрое преобразование Фурье. Спектральный анализ. Обработка статистики. Выравнивание экспериментальных данных. Обыкновенные дифференциальные уравнения. Двухточечные краевые задачи. Интегральные уравнения и обратные преобразования. Дифференциальные уравнения в частных производных. Нечисловые задачи (главным образом помехоустойчивые коды).

Выбрав в каталоге раздела нужный файл, его можно скопировать в нужную директорию (в панели инструментов `Recipes` имеется кнопка `Set Path`), включить в проект и откомпилировать. Запасать продукты такой компиляции в библиотеке нет смысла, поскольку многие процедуры в связи с отсутствием в Фортране 77 динамических массивов перед использованием нуждаются в настройке параметров.

Сопоставление содержимого разделов обсуждаемых библиотек показывает, что в ряде случаев они взаимно дополняют друг друга. Это определяет целесообразность использования их всех.

[2]Следует считаться с возможной недостаточностью этого средства, когда для достижения высокой точности потребуется принципиально иной алгоритм.

15.9. Работа с личными библиотеками

В главе 12 была описана работа с профессиональными библиотеками, интегрированными в систему программирования Ф90. Однако серьезно работающий программист-прикладник быстро обнаруживает, что занимающие его специальные задачи и их фрагменты начинают повторяться. В связи с этим он приходит к необходимости создания и использования *личных* библиотек. Применение их для реализации однотипных функций ускоряет работу, уменьшает общий объем программного кода, упрощает структуру программных систем, облегчает их доработку и обновление, обеспечивает работу разных частей таковых с тождественными версиями подпрограмм.

Опыт программиста может накапливаться просто в виде архивов исходных текстов программ и сопутствующей документации. Автор с грустью наблюдал многолетнюю работу специалиста по численным методам газовой динамики и хорошего программиста, который долгие годы игнорировал системные технологии и в каждую новую программу непосредственно вставлял фрагменты исходных текстов старых. Накопленный им архив превышал миллион перфокарт и доставлял владельцу немало хлопот при вводе очередной задачи.

Гораздо более эффективной формой накопления опыта служат *личные библиотеки объектных подпрограмм*, которые не нуждаются в перекомпиляции для каждого их использования. Технология их создания описана в [85]. Для обновления приложений с измененной библиотекой нужно лишь заново выполнить линкование. Достаточно содержательные библиотеки могут стать основой успешного коммерческого проекта.

При работе со статической библиотекой в выполняемый EXE-модуль включаются только реально необходимые процедуры. Их автоматически отбирает линкер для удовлетворения вызовов внешних процедур.

DLL (dynamic link libraries, библиотеки динамической компоновки) служат для хранения процедур, которые могут одновременно понадобиться одной или нескольким программам. Для этого члены библиотеки должны быть реентерабельными. Сфера их преимущественного использования — операционные системы и управляющая часть сложных приложений.

Заключение

В настоящем учебном пособии изложены далеко не все алгоритмы приближенного решения основных задач анализа. Некоторые чрезвычайно важные для приложений задачи (дифференциальные уравнения в частных производных, вариационные проблемы) не упоминались вовсе. Заметим, однако, что решение этих задач часто расчленяется на более простые, часть из которых рассмотрена в данном учебном пособии; поэтому оно существенно облегчит овладение более сложными разделами численного анализа.

Появление электронных цифровых вычислительных машин и включение в состав математического обеспечения ЭЦВМ стандартных подпрограмм, реализующих типовые алгоритмы численного анализа, способствовали резкому расширению областей его применения и круга лиц, активно использующих его методы.

Применение ЭВМ не заменяет плодотворные математические методы, но должно сочетаться с ними. Недостаточная математическая подготовка основной части пользователей привела к распространению слепой веры во всемогущество ЭЦВМ и неправильной реализации их возможностей. Рассмотрим наиболее характерные ошибки этого рода.

Недоиспользование классической математики. Эта ошибка прежде всего проявляется в стремлении возможно скорее выйти с задачей на машину, не исчерпав возможностей аналитических методов применительно к проблеме в целом и ее отдельным частям. Часто программируют последовательное накопление непосредственно суммируемых рядов; для каждого члена степенного ряда заново считают степени независимой переменной и факториалы; интерполяционные квадратурные формулы применяют к табличным или сводящимся к ним интегралам; при использовании составных формул интегрирования перевычисляют ранее найденные ординаты; пользуются методом Рунге—Кутты для дифференциального уравнения с известным аналитическим выражением решения. При использовании итерационных методов недостаточное внимание уделяется получению хороших начальных приближений.

Шаблонное применение вычислительных методов. Ошибки этого вида встречаются наиболее часто. Их проявления весьма многообразны: попытка применения итерационных методов без предварительной проверки условии сходимости; численное интегрирование быстро колеблющейся функции без учета расположения ее экстремумов; численное дифференцирование в условиях, когда погрешности вычисления функции соизмеримы с ее приращениями. К ошибкам данной группы можно отнести и упущенные возможности преобразования исходной задачи к форме, обеспечивающей более эффективные вычисления (см. раздел о вычислении несобственных интегралов).

По-видимому, самым общим советом по профилактике ошибок данного вида будет рекомендация тщательного анализа свойств каждой конкретной математической модели, которые могут повлиять на ход вычислений.

Заметим, однако, что довольно часто приходится применять методы, допустимость которых (например, их сходимость) теоретически не доказана или теоретические оценки непригодны для практической работы. Поэтому при выборе метода решения задачи и анализа результатов приходится полагаться на опыт предшествующего решения задач, интуицию, сравнение с экспериментом. Для успеха в проведении расчетов на ЭЦВМ, помимо безусловного знания основных методов вычислений и техники программирования, необходимы знание существа прикладной проблемы, развитое неформальное мышление и умение находить аналогии, дающие основания надеяться на достоверность результата при невозможности строгих доказательств.

Неправильное использование возможностей ЦВМ. Возможности современных ЦВМ велики, но не безграничны. Еще недавно машинное время было дефицитно и стоило десятки и даже сотни рублей в час (в *старом* масштабе цен — умножьте эти цифры на 50!). Однако даже сейчас не следует компенсировать собственную лень и нежелание поразмыслить над задачей бездумной тратой ресурсов ЦВМ. К сожалению, слишком часто:

- расчет выполняется для гораздо большего диапазона параметров, чем это необходимо на самом деле;
- непосредственно вычисляются излишне подробные таблицы (если они все же необходимы, их обычно выгоднее получать интерполяцией с помощью сравнительно небольшого числа опорных точек).

К бессмысленной трате машинного времени приводят и такие уже от-

меченные ошибки, как низкое качество начальных приближений, плохая сходимость итерационных процессов и т. п.

Весьма распространенной ошибкой является вывод на печать избыточного объема результатов. Убедительный и вполне правдоподобный пример такого рода обсуждается в [29, с. 576–577], который мы дополним расчетом трудоемкости и стоимости $печати$.

Пусть мы хотим составить таблицу решений полного кубического уравнения

$$ax^3 + bx^2 + cx + d = 0.$$

Если допустить, что каждый из параметров может принимать 50 значений, то всего получится $50^4 \approx 6 \cdot 10^6$ комбинаций. Расчет на современной ПЭВМ по формулам Кардано займет немного времени и «ничего не будет стоить». Совсем иначе обстоит дело с выводом и использованием результатов. При печати по 6 чисел в строке на странице можно разместить около 300 чисел; всего потребуется 20 тыс. страниц, или 40 томов по 500 страниц в каждой. Вывод на печать с помощью лазерного принтера по 5 страниц в минуту займет 4 тыс. минут, т. е. около трех суток $непрерывной$ работы принтера и оператора при нем. На бумагу уйдет около 5 тыс. рублей; печать потребует 8 заправок картриджа, за что придется заплатить еще 2 тыс. (если удастся обойтись без покупки нового).

К сожалению, такого рода заявки отнюдь не являются околонаучной фантастикой. В дни молодости отечественной вычислительной техники (конец 1950-х гг.) один из старших коллег автора некритически воспользовался возможностью заказать расчет нужных ему таблиц на «Урале-1». Через полгода был доставлен объемистый мешок, доверху набитый исписанными девичьим почерком блокнотиками. Мешок развязали, ахнули, завязали снова и через месяц... сожгли.

Вернемся, однако, к нашей задаче и попытаемся уменьшить число ее параметров. Разделим обе части уравнения на a:

$$x^3 + \frac{b}{a}x^2 + \frac{c}{a}x + \frac{d}{a} = 0.$$

Затем сделаем подстановку $x = y + \alpha$, где $\alpha = -b/3a$ выбирается из условия уничтожения члена с неизвестным во второй степени. В результате приходим к уравнению

$$y^3 + py + q = 0,$$

где

$$p = \frac{c}{a} - \frac{b^2}{3a^2}; \quad q = \frac{d}{a} - \frac{bc}{3a^2} + \frac{2b^3}{27a^2}.$$

Сделаем, наконец, подстановку $y = \beta z$; это даст

$$\beta^3 z^3 + p\beta z + q = 0, \quad \text{или} \quad z^3 + \frac{p}{\beta^2} z + \frac{q}{\beta^3} = 0.$$

Выбрав $\beta = \sqrt[3]{q}$, приходим к уравнению

$$z^3 + rz + 1 = 0$$

с *единственным* параметром $r = pq^{-2/3}$. По найденному z легко находится

$$x = \beta z + \alpha = [\sqrt[3]{27a^3 d - 9abc + 2b^3} z]/(3a).$$

Новые (комбинированные) параметры часто имеют глубокий физический смысл типа критериев подобия и наталкивают на весьма конструктивные идеи.

Если у вычислителя имеется ясное представление о способе обработки результатов, то эту обработку следует также возложить на ЦВМ. Если такого представления нет, то от решения задачи вообще следует воздержаться.

Встречаются ошибки и другого рода: на печать выдается жесткий минимум результатов. В этом случае вычислитель затрудняет себе контроль правильности счета и упускает возможность попутно исследовать важные свойства математической модели: крутизну критериальной функции в задачах оптимизации, степень влияния различных факторов, скорость сходимости итерационных процессов и т. д.

Изложенные соображения и анализ типовых ошибок, допускамых при машинной реализации вычислительных схем, убедительно свидетельствуют о необходимости исследовательского подхода к каждой конкретной задаче и творческого применения методов вычислительной математики всеми лицами, непосредственно использующими ЦВМ. Прежде чем решать задачу, подумайте, что делать с ее решением!

Литература

1. *Акритас А.* Основы компьютерной алгебры с приложениями /Пер. с англ. — М.: Мир, 1994. — 544 с.

2. Актуальные проблемы вычислительной математики и математического моделирования /Отв. редактор А. А. Алексеев. — Новосибирск: Наука, 1985. — 264 с.

3. *Алабужев П. М., Геронимус В. Б., Минкевич Л. М., Шеховцев Б. А.* Теория подобия и размерностей, моделирование. — М.: Высшая школа, 1968. — 208 с.

4. *Амосов А. А. и др.* Вычислительные методы для инженеров: Учеб. пособие. — М.: Высшая школа, 1994. — 544 с.

5. *Бабенко К. И.* Основы численного анализа. — М.: Наука, 1986. — 43 с.

6. *Бакушинский А. Б., Гончарский А. В.* Итерационные методы решения некорректных задач. — М.: Изд-во МГУ, 1989. — 197 с.

7. *Банди Б.* Методы оптимизации. Вводный курс /Пер. с англ. — М.: Радио и связь, 1988. — 128 с.

8. *Бартеньев О. В.* Современный Фортран. — М.: Диалог–МИФИ, 1998. — 397 с.

9. *Бартеньев О. В.* Фортран для профессионалов: математическая библиотека IMSL, тт. 1-2. — М.: Диалог–МИФИ, 2000.

10. *Бахвалов И. В., Жидков Н. П., Кобельков Г. М.*. Численные методы. — М.: Лаборатория Базовых Знаний, 2000. — 624 с.

11. *Блехман И. И.* Прикладная математика: предмет, логика, особенности подходов. Киев: Наукова думка, 1976. — 269 с.

12. *Боглаев Ю. П.* Вычислительная математика и программирование: Учеб. пособие. М.: Высшая школа, 1990. — 543 с.

13. *Вейль Г.* Математическое мышление: Сб. статей. — М.: Наука, 1989. — 400 с.

14. *Веников В. А.* О моделировании. М.: Знание, 1974. — 63 с.

15. *Вержбицкий В. М.* Основы численных методов: Учебник. — М.: Высшая школа, 2002. — 848 с.

16. *Воеводин В. В.* Математические модели и методы в параллельных процессах. — М.: Наука, 1986. — 296 с.

17. *Волков Е. А.* Численные методы: Учеб. пособие. — М.: Наука, 1987. — 348 с.

18. Вычислительная математика: Учеб. пособие. — СПб.: ВИКУ им. А. Ф. Можайского, 1997. — 354 с.

19. *Ганшин Г. С.* Методы оптимизации и решения уравнений. — М.: Наука, 1987. — 125 с.

20. *Гнеденко Б. В.* Математика и научное познание. — М.: Знание, 1983. — 64 с.

21. *Городецкий В. И., Рыжиков Ю. И.* Математическое программирование и массовое обслуживание: Учеб. пособие. Л.: ВИКИ им. А. Ф. Можайского, 1975. — 182 с.

22. *Давыдов Е. Г.* Интегрированная система Scientific WorkPlace 4.0. — М.: Финансы и статистика, 2003. — 208 с.

23. *Демидович Б. П., Марон И. А.* Основы вычислительной математики. — М.: Физматгиз, 1963. — 660 с.

24. *Демидович Б. П., Марон И. А., Шувалова Э. З.* Численные методы анализа. Приближение функций, дифференциальные и интегральные уравнения. — М.: Физматгиз, 1963. — 400 с.

25. *Дэвенпорт Дж. и др.* Компьютерная алгебра: системы и алгоритмы алгебраических вычислений /Пер. с франц. — М.: Мир, 1991. — 350 с.

26. *Дьяконов В. П.* Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. — М.: Наука, Физмат, 1987. — 240 с.

27. *Забрейко П. П. и др.* Интегральные уравнения (Справочная математическая библиотека). — М.: Наука, Физмат, 1968. — 448 с.

28. Зарубежные библиотеки и пакеты программ по вычислительной математике /Пер. с англ. — М.: Наука, 1993. — 343 с.

29. *Зельдович Я. Б., Мышкис А. Д.* Элементы прикладной математики. — М.: Наука, 1972. — 592 с.

30. *Иванов В. В.* Методы вычислений на ЭВМ: Справ. пособие. — Киев: Наукова думка, 1986. — 582 с.

31. *Икрамов Х. Д.* Численное решение матричных уравнений. — М.: Наука, Физмат, 1984. — 192 с.

32. *Ильин В. П.* Вычислительная информатика: открытие науки. — Новосибирск: Наука, 1991. — 194 с.

33. *Исаков В. Н.* Элементы численных методов. — М.: Академия, 2003. — 188 с.

34. История и методология естественных наук: Сб. статей. — Вып. 5, Математика. — М.: МГУ, 1966. — 275 с.

35. *Калиткин Н. Н.* Численные методы: Учеб. пособие. — М.: Наука, 1978. — 512 с.

36. *Канторович Л. В., Крылов В. И.* Приближенные методы высшего анализа. — М.–Л.: Физмат, 1962. — 708 с.

37. *Каханер Д., Моулер К., Нэш С.* Численные методы и программное обеспечение /Пер. с англ. — М.: Мир, 1998. — 575 с.

38. *Клейнен Дж.* Статистические методы в имитационном моделировании, вып. 1–2 /Пер. с англ. — М.: Статистика, 1978.

39. *Коваленко Е. С., Киселев О. Н., Шарыгин Г. С.* Основы научных исследований: Учеб. пособие. — Томск: ТГУ, 1989. — 192 с.

40. *Козлов В. Н.* Математика и информатика: Учеб. пособие. — СПб.: ГТУ, 2001. — 265 с.

41. *Колесников А. П.* Математическое моделирование и информатика: Учеб. пособие. — М.: РУДН, 2003. — 108 с.

42. *Колмогоров А. Н.* Математика — наука и профессия. — М.: Наука, 1988. — 287 с.

43. *Копченова Н. В., Марон И. А.* Вычислительная математика в примерах и задачах. — М.: Наука, Физмат, 1972. — 368 с.

44. *Коробицын В. И., Фролова Ю. В.* Численные методы и программирование. — Омск: Наследие, Диалог-Сибирь. — 2004. — 183 с.

45. *Косарев В. И.* 12 лекций по вычислительной математике. — М.: МФТИ, 1995. — 175 с.

46. *Костомаров Д. П., Фаворский А. П.* Программирование и численные методы. М.: МГУ, 2001. — 223 с.

47. *Костомаров Д. П., Фаворский А. П.* Вводные лекции по численным методам. М.: Логос, 2004. — 183 с.

48. *Краскевич В. Е. и др.* Численные методы в инженерных исследованиях: Учеб. пособие. — Киев: Вища школа, 1986. — 263 с.

49. *Краснов М. Л., Киселев А. И., Макаренко Г. И.* Интегральные уравнения. — М.: УРСС, 2003. — 192 с.

50. *Крылов А. Н.* Мои воспоминания. — Л.: Судостроение, 1984. — 480 с.

51. *Крылов В. И., Шульгина Л. Т.* Справочная книга по численному интегрированию. — М.:, Наука, Физмат, 1966. — 372 с.

52. *Крылов В. И., Бобков В. В., Монастырный П. И.* Вычислительные методы, т. 1. — М.: Наука, Физмат, 1976. — 304 с.

53. *Крылов В. И., Бобков В. В., Монастырный П. И.* Линейная алгебра и нелинейные уравнения. — Минск: Наука и техника, 1985. — 280 с.

54. *Кудрявцев Л. Д.* Мысли о современной математике и ее изучении. — М.: Наука, 1977. — 111 с.

55. *Кук Д., Бейз Г.* Компьютерная математика /Пер. с англ. — М.: Наука, 1990. — 383 с.

56. *Кульгин М.* Теория очередей и расчет параметров сети // BYTE — Россия, ноябрь 1999. — С. 26–33.

57. *Куницман Ж.* Численные методы /Пер. с франц. — М.: Наука, 1979. — 159 с.

58. *Курант Р., Роббинс Г.* Что такое математика? /Пер. с англ. — М.: Просвещение, 1967. — 558 с.

59. *Ланс Дж. Н.* Численные методы для быстродействующих вычислительных машин /Пер. с англ. — М.: ИЛ, 1962. — 208 с.

60. *Лесин В. В., Лисовец Ю. П.* Основы методов оптимизации: Учеб. пособие. — М.: МАИ, 1995. — 344 с.

61. Математика в афоризмах, цитатах, высказываниях /сост. Н. А. Вирченко. — Киев: Вища школа, 1983. — 278 с.

62. Математика сегодня'90: Научно-метод. сб. — Киев: Вища школа, 1990. — 183 с.

63. Математики о математике: Сб. статей. — М.: Знание, 1982. — 64 с.

64. *Меткалф М., Рид Дж.* Описание языка программирования Фортран-90 /Пер. с англ. — М.: Мир, 1995. — 302 с.

65. *Мину М.* Математическое программирование. Теория и алгоритмы /Пер. с франц. — М.: Физматгиз, 1990. — 488 с.

66. *Михлин С. Г., Смолицкий Х. Л.* Приближенные методы решения дифференциальных и интегральных уравнений (Справочная математическая библиотека). — М.: Наука, Физматгиз, 1965. — 384 с.

67. Моделирование систем и информатика: Сб. трудов. — М.: Ун-т дружбы народов, 1989. — 102 с.

68. *Молчанов И. Н.* Машинные методы решения прикладных задач. Алгебра, приближение функций. — Киев: Наукова думка, 1987. — 287 с.

69. *Мысовских И. П.* Интерполяционные кубатурные формулы. — М.: Наука,1981. — 336 с.

70. Некорректные задачи естествознания. — М.: изд-во МГУ, 1987. — 303 с.

71. Основы современных компьютерных технологий: Учебник. — СПб: КОРОНА Принт, 2005. — 672 с.

72. Пакет научных программ на Фортране. — Харьков, 1983. — 116 с.

73. Пакет прикладных программ МОСТ для расчета стационарных режимов в системах массового обслуживания. — Эстонское НПИ ВТИ. — 1988.
Вып. 3. Описание применения. — 36 с.
Вып. 5. Руководство программиста. — 92 с.
Вып. 6. Теоретические основы, ч. 1. — 116 с.
Вып. 7. Теоретические основы, ч. 2. — 142 с.

74. *Пирумов У. Г.* Численные методы. — М.: Дрофа, 2003. — 223 с.

75. *Половко А. М., Бутусов П. Н.* Интерполяция. Методы и компьютерные технологии их реализации. — СПб.: БХВ-Петербург, 2004. — 320 с.

76. *Поршнев С. В.* Вычислительная математика. Курс лекций. — СПб.: БХВ-Петербург, 2004. — 320 с.

77. Практическое руководство по программированию /под ред. Б. Мика, П. Хит, Н. Рашби, пер. с англ. — М.: Радио и связь, 1986. — 168 с.

78. *Ракитин В. И., Первушин В. Е.* Практическое руководство по методам вычислений с приложением программ для персональных компьютеров. — М.: Высшая школа, 1998. — 383 с.

79. *Рузавин Г. И.* О природе математического знания. — М.: Мысль, 1968. — 302 с.

80. *Рыжиков Ю. И.* Машинные методы расчета систем массового обслуживания: Учеб. пособие. — Л.: ВИКИ им. А. Ф. Можайского, 1979. — 177 с.

81. *Рыжиков Ю. И.* Эффективность и эксплуатация программного обеспечения ЭЦВМ: Учеб. пособие. — МО СССР, 1985. — 263 с.

82. *Рыжиков Ю. И.* Решение научно-технических задач на персональном компьютере. — СПб.: КОРОНА Принт, 2000. — 271 с.

83. *Рыжиков Ю. И.* Информатика. Лекции и практикум. — СПб.: КОРОНА Принт, 2000. — 256 с.

84. *Рыжиков Ю. И.* Теория очередей и управление запасами. — СПб.: Питер, 2001. 376 с.

85. *Рыжиков Ю. И.* Современный Фортран. — СПб.: КОРОНА принт, 2004. — 288 с.

86. *Рыжиков Ю. И.* Работа над диссертацией по техническим наукам. — СПб.: БХВ-Петербург, 2005. — 496 с.

87. *Рябенький В. С.* Введение в вычислительную математику. — М.: Наука, 2000. — 296 с.

88. *Самарский А. А.* Введение в численные методы: Учеб. пособие. — СПб.: Лань, 2005. — 288 с.

89. Сборник научных программ на Фортране /Пер. с англ. — М.: Статистика, 1974.

90. *Сергеев И. Н.* Примени математику. — М.: Наука, 1989. — 240 с.

91. *Сигорский В. П.* Математический аппарат инженера. — Киев: Техника, 1977. — 766 с.

92. *Смолицкий Х. Л., Рыжиков Ю. И.* Вычислительная математика. — Л.: ВИКИ им. А. Ф. Можайского, 1976. — 144 с.

93. *Сойер У. У.* Прелюдия к математике. Рассказ о некоторых... /Пер. с англ. — М.: Просвещение, 1972. — 192 с.

94. *Сойер У. У.* Путь в современную математику /Пер. с англ. — М.: Мир, 1972. — 259 с.

95. *Стренг Г.* Линейная алгебра и ее применения /Пер. с англ. — М.: Мир, 1980. — 454 с.

96. *Стронгин Р. Г.* Поиск глобального оптимума. — М.: Знание, 1990. — 48 с.

97. *Тихонов А. Н., Арсенин В. Я.* Методы решения некорректных задач. — М.: Наука, Физмат, 1979. — 288 с.

98. *Тихонов А. Н., Костомаров Д. П.* Вводные лекции по прикладной математике. М.: Наука, 1984. — 190 с.

99. *Турчак Л. И., Плоткин П. В.* Основы численных методов: Учебное пособие. — М.: Физматгиз, 2003. — 304 с.

100. *Уилкинсон, Райнш.* Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра /Пер. с англ. — М.: Машиностроение, 1976 г. — 389 с.

101. *Федоренко Р. П.* Введение в вычислительную физику: Учеб. пособие. — М., 1994. — 528 с.

102. *Формалев В. Ф., Ревизников Д. Л.* Численные методы. — М.: Физматгиз, 2004. — 400 с.

103. *Фрейденталь Х.* Математика в науке и вокруг нас /Пер. с нем. — М.: Мир, 1977. — 261 с.

104. *Хемминг Р. В.* Численные методы для научных работников и инженеров /Пер. с англ. — М.: Наука, Физмат, 1968. — 400 с.

105. *Черноруцкий И. Г.* Методы принятия решений. — СПб.: БХВ–Петербург, 2005. — 416 с.

106. *Чечкин А. В.* Математическая информатика. — М.: Наука, 1991. — 411 с.

107. Что такое прикладная математика. — М.: Знание, 1980. — 63 с.

108. *Шипачев В. С.* Высшая математика: Учебник. — М.: Высшая школа, 2003. — 480 с.

109. *Шиханович Ю. А.* Введение в современную математику. Начальные понятия. — М.: Наука, 1965. — 376 с.

110. *Шуп Т.* Решение инженерных задач на ЭВМ. Практическое руководство /Пер. с англ. — М.: Мир, 1982. — 238 с.