МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Физический факультет
Кафедра высшей математики

С. В. Смирнов

ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ФИЗИКИ

Часть I

Учебное пособие

Новосибирск
2015

УДК 519.6
ББК 22.19я73
С50

Рецензент
д-р. физ.-мат. наук, проф. М. П. Федорук

Смирнов, С. В.
С50 Основы вычислительной физики : учеб. пособие /
С. В. Смирнов. Новосибирск : РИЦ НГУ, 2015. – Ч. I. – 113 с.

ISBN 978-5-4437-0429-6

Настоящее учебное пособие соответствует материалу первых шести лекций по дисциплине «Основы вычислительной физики», читаемых студентам 4 курса физического факультета НГУ, и содержит рассмотрение ряда базовых вопросов методов вычислений, используемых в физике. Пособие знакомит читателей с машинной арифметикой, решением уравнений с одним неизвестным, квадратурными формулами, интерполяцией, интегрированием обыкновенных дифференциальных уравнений. Отбор материала и уровень строгости изложения адаптированы для студентов-физиков. Для студентов 4 курса физического факультета НГУ, студентов старших курсов и аспирантов физических и технических специальностей вузов.

УДК 519.6
ББК 22.19я73

ISBN 978-5-4437-0429-6

Оглавление

Предисловие

Настоящее учебное пособие соответствует материалу первых шести лекций по дисциплине «Основы вычислительной физики», читаемых студентам 4 курса физического факультета НГУ, и содержит рассмотрение ряда базовых вопросов методов вычислений, используемых в физике. Пособие знакомит читателей с машинной арифметикой, решением уравнений с одним неизвестным, квадратурными формулами, интерполяцией, интегрированием обыкновенных дифференциальных уравнений. Отбор материала и уровень строгости изложения адаптированы для студентов-физиков.

Автор выражает глубокую признательность Александру Ивановичу Черных за большой вклад в отбор материала курса лекций и целый ряд исключительно полезных советов; Максиму Александровичу Никулину за выполненное им большинство креативных иллюстраций, а также обсуждения, идеи и многочисленные критические замечания, позволившие существенно улучшить текст пособия.

Введение

Развитие современной науки и технологий немыслимо без численных расчётов и компьютерного моделирования. Значительная часть актуальных научных и инженерных задач не могут быть решены аналитически без достаточно сильных допущений и упрощений, существенно ограничивающих точность и применимость получаемых результатов. С другой стороны, стремительно растущие в последнее время вычислительные мощности современных ЭВМ позволяют получать численные решения многих сложных задач с точностью, значительно превосходящей возможности аналитических решений. Термоядерный синтез, создание и эксплуатация атомного реактора, взрыв бомбы, расчёт и оптимизация авиационных двигателей и турбин электростанций, проектирование мостов и зданий, вывод на орбиту искусственных спутников и полёты к другим планетам, репликация заданных участков ДНК *in vitro* для медицинской диагностики, создание метаматериалов, разработка новых микропроцессоров — лишь малая часть наиболее очевидных примеров впечатляющих задач, для решения которых используется моделирование.

Высокая производительность современных компьютеров и открываемые ими возможности, однако, не должны вводить в заблуждение и создавать иллюзию исключительности численных методов решения

задач. В действительности, наибольших успехов можно достичь, грамотно сочетая возможности аналитических (символьных) вычислений и численных расчётов. Важно помнить, что численное моделирование не может и не должно заменять собой аналитических расчётов и оценок.

Одним из ключевых преимуществ аналитических решений является возможность увидеть и проанализировать зависимость результатов от произвольного числа параметров задачи. В противоположность этому, численные расчёты всегда дают значение искомой величины в одной точке. Чтобы исследовать зависимость от нескольких параметров, нужно выполнить большое число однотипных расчётов, что может быть сопряжено с большими затратами времени и вычислительных ресурсов.

С другой стороны, аналитическое решение может быть получено лишь для относительно узкого круга простых задач (при наличии симметрии, малых параметров) и почти всегда является весьма приближённым ввиду значительных допущений. В противоположность этому, численные решения, как правило, могут быть получены с меньшим числом приближений — для несимметричных физических систем, при отсутствии малых и больших параметров и т. п. При этом удаётся достичь более высокой точности решения и расширить область его применимости на представляющие наибольший интерес случаи, реализуемые на практике.

Важным достоинством аналитических решений является относительная простота их верификации. Аналитические решения, как правило, публикуются в научной литературе полностью вместе с ключевыми выкладками, которые привели к ответу. Открытость решения позволяет выполнить его проверку любому специалисту в данной области исследования, обладающему достаточным уровнем квалификации. Кроме того, полученный ответ, как правило, допускает ряд способов верификации, включая проверку по размерности, а также исследование предельных случаев входящих в него параметров задачи. Численное решение гораздо сложнее поддаётся верификации сторонними специалистами, не принимавшими непосредственного участия в работе. Действительно, исходные коды программ обычно не публикуются в научных работах ввиду своей громоздкости и сложности их анализа. Наиболее распространённые в численных расчётах языки программирования не имеют средств для проверки размерностей, а исследование предельных случаев и соответствия аналитическим решениям в тех областях пространства параметров, где они могут быть получены, может быть произведено только самими авторами исследования.

Численное решение задач, как правило, легче и требует меньшей квалификации исследователя по сравнению с получением аналитических решений. Вычислительная физика обладает меньшим «барьером вхождения», который необходимо преодолеть для начала работы. Более того, в последнее время появился набор готовых программных продуктов (таких как Matlab, Mathematica и др.), ещё больше сокращающих время и количество усилий, которые необходимо затратить для подготовки вычислений и даже полного решения некоторого набора задач. В этой связи может создаться впечатление, что для получения численного решения задачи достаточно иметь современный компьютер с необходимым набором программ и начальные познания в области высшей математики. Следует предостеречь читателя, что такое впечатление верно лишь отчасти. Подобно тому, как обладание современным цифровым фотоаппаратом не делает фотографом любого желающего щёлкать затвором, так же и выполнение численных расчётов требует определённой квалификации, необходимость которой может быть сразу не очевидна. В отличие от фотоискусства, успехи в компьютерном моделировании могут быть поверены критериями точности и эффективности, поддающимися строгим математическим определениям. В рамках данного курса мы познакомимся с целым рядом примеров, когда результаты численных расчётов, выдаваемые «написанной без ошибок» программой, идут вразрез с неподготовленной интуицией начинающего исследователя — надеемся, что эти примеры уберегут читателя от многих ошибок в его будущей работе.

Пожалуй, самым важным правилом, которое необходимо освоить для успешного применения численных методов решения физических задач, является *критическое отношение к получаемым результатам моделирования.* Сложность грамотного использования численных методов является оборотной стороной их кажущейся простоты: компьютер почти всегда (за редким исключением зациклившейся программы или выхода за пределы разрядной сетки) выдаёт какое-то число. Он не знает сомнений и всегда даст ответ — вопрос лишь в *правильности* и *точности* этого ответа. Современный компьютер, несмотря на огромную по человеческим меркам скорость арифметических вычислений, не более «умён», чем калькулятор, арифмометр или логарифмическая линейка. Вряд ли кто-то из читателей будет всерьёз возлагать на калькулятор ответственность за правильность полученного результата — очевидно, что если в вычислениях возникла ошибка, то практически наверняка это произошло по вине человека, неверно нажавшего клавишу или использовавшего неточные формулы и/или исходные данные для вычислений. Аргументированный и убедительный ответ на вопрос

о правильности выдаваемых программой результатов всегда остаётся за человеком, тогда как компьютер в расчётах — не более чем удобный инструмент, пришедший на смену арифмометрам и логарифмическим линейкам и существенно расширивший возможности численного решения задач благодаря высокой скорости вычислений.

Заметим, что вопросы о правильности и точности полученного решения порой оказываются весьма непростыми. В этой связи показателен пример из воспоминаний академика С. К. Годунова[1] («О пользе ложных гипотез», Эксперт Сибирь, № 32 (174), 3 сентября 2007 г.):

> Однажды в Институт математики им. В. А. Стеклова приехал Лев Ландау (через 11 лет — нобелевский лауреат по физике). Собрал математиков, вычислителей, раздал привезённые с собой графики и расчёты с вычислениями решений обыкновенных уравнений и попросил решить задачу: «Мне надо знать, сколько тут знаков. Говорят, что шесть, а я не знаю, есть они здесь или нет». Началось обсуждение. Среди дискутирующих — Израиль Гельфанд, Константин Семендяев. Кто говорит — семь знаков, кто — восемь, и абсолютно все утверждают, что за шесть можно ручаться. Лев Давидович спрашивает почему — ему отвечают, потом он вынимает из своего кармана скомканный лист и объявляет: «А у меня совсем другое, но точное аналитическое решение!». У публики, рассказывает Сергей Константинович, был шок. Никто не заметил то «тонкое, узенькое место», которое очень сильно влияло на решение задачи. «По существу, все были опозорены», — заключает Годунов.

Таким образом, написание и запуск программы являются лишь первым шагом на пути решения задачи. Исследование полученных результатов, сравнение с аналитическими решениями в тех областях значений параметров, где они могут быть получены, поверка качественного поведения результатов физическими соображениями и здравым смыслом, апостериорная оценка точности полученного численного решения также являются совершенно необходимыми этапами численного решения задачи. Зачастую оставаясь в тени или находя своё отражение лишь в

[1] Годунов Сергей Константинович — советский и российский математик и механик, академик РАН. Внёс значительный вклад в развитие общей теории разностных схем, применяемых при решении дифференциальных уравнений. Кавалер ордена «Знак почёта» и двух Орденов Трудового Красного Знамени, Ленинской премии, премий АН СССР им. А. Н. Крылова и РАН им. М. А. Лаврентьева. С 1969 по 1997 гг. — профессор кафедры дифференциальных уравнений ММФ НГУ, с 1977 по 1989 гг. — зав. кафедрой.

предельно кратких ремарках в научных статьях и монографиях, именно эти этапы решения сильнее всего отличают профессионала от начинающего заниматься численным моделированием.

Другой аспект профессионализма физика-вычислителя связан с оптимальным выбором численных алгоритмов для обеспечения высокой скорости расчётов при требуемом уровне точности. Хотя производительность аппаратных вычислительных средств демонстрирует поистинне ошеломляющий рост начиная с середины прошлого столетия, методы решения численных задач также непрерывно совершенствуются. Так, например, в своей монографии [A1, с. 343] Джон Райс отмечает, что для трёхмерных эллиптических уравнений повышение быстродействия за счёт разработки новых методов вычислений в период с 1945 до 1978 гг. превышает прогресс быстродействия электроники! Однако даже безотносительно последних достижений в области вычислительной математики, которые далеко выходят за рамки данного курса, следует помнить, что использование неэффективных методов способно на много порядков величины уменьшить быстродействие расчётов и даже привести начинающего исследователя к ошибочному выводу о «нерешаемости» стоящей перед ним задачи.

Ключевая роль в приобретении необходимых компетенций в области вычислительной физики, безусловно, принадлежит практическим занятиям в терминальном классе. На лекциях в первую очередь будет рассмотрен круг базовых вопросов, мимо которых вряд ли удастся пройти любому, кто вынужден будет прибегнуть в своей работе к использованию численных методов решения задач: поиск корней, квадратурные формулы, конечные разности, интегрирование обыкновенных дифференциальных уравнений, ряд базовых задач линейной алгебры. Кроме того, программа курса вкючает ряд относительно простых примеров более специфических задач и методов их решения, которые могут не иметь непосредственного отношения к задачам дипломной практики и последующей работе в тех или иных областях. Тем не менее, мы надеемся, что решение предложенных задач окажется полезным, позволив приобрести навыки отладки программ и верификации результатов, на практике познакомиться с рядом базовых понятий, таких как аппроксимация, сходимость, порядок точности и устойчивость численных схем.

Для численного решения задач в компьютерном классе рекомендуется использовать язык Си или Си++. Выбор Си в качестве инструмента обусловлен его высокой эффективностью, переносимостью на разные платформы, наличием большого числа библиотек и компиляторов, в том числе и бесплатных. Разумеется, существует значительное

число альтернатив, включая как различные языки программирования (Fortran, Java, JavaScript, Python и другие), число которых продолжает расти, так и специализированные среды для численного моделирования и вычислений (MatLab, MathCad, Mathematica, COMSOL и т. п.) Нет однозначного ответа на вопрос, что лучше использовать для вычислений — оптимальное решение зависит от задачи, личных предпочтений и опыта использования тех или иных инструментов. Делая свой выбор в пользу языка Си, мы надеемся, что читатель при желании сможет впоследствии познакомиться с различными средами вычислений и языками программирования самостоятельно либо на соответствующих спецкурсах. Поскольку учебный план включает изучение основ Си на младших курсах, в данном пособии мы не будем касаться вопросов программирования (за исключением ряда полезных советов — см. с. 109), отсылая читателей к руководствам [1] и [C2]. Для углубленного изучения Си++ рекомендуем Интернет-ресурсы [C3, C4], официальный стандарт языка [C5] и книгу [C6].

1. Построение графиков в gnuplot

Как уже отмечалось выше, в отличие от аналитического решения, позволяющего непосредственно увидеть зависимость исследуемых величин от параметров задачи, использование численных методов предполагает многократное повторение решения с разными наборами параметров для исследования таких зависимостей. Помимо прочего, такой подход сопряжён с необходимостью анализа большого количества данных, выдаваемых компьютерной программой, что обуславливает важность грамотного представления результатов. Человеческий мозг гораздо быстрее и легче воспринимает графическую информацию — для понимания качественных свойств полученного численного решения целесообразно использовать графики вместо таблиц значений в качестве результата работы программы.

К настоящему времени разработано огромное количество программ для построения графиков, анализа и обработки численных зависимостей и таблиц. Опять-таки среди них нет одной «лучшей», которую можно было бы рекомендовать на все случаи жизни — оптимальный выбор зависит от задачи и предпочтений конкретного пользователя.

Для использования в компьютерном практикуме мы рекомендуем кросс-платформенную программу gnuplot. Её можно бесплатно скачать с официального сайта [A2]. Относительно небольшой размер программы и возможность работы без установки под Windows позволяют всегда

иметь gnuplot под рукой на съёмном USB-носителе. Также к достоинствам gnuplot можно отнести относительную простоту использования, возможность сохранять графики в различных форматах, построение двумерных и трёхмерных графиков как с использованием таблиц значений из текстового файла или потока, так и по заданному символьному выражению. Полезной особенностью gnuplot является возможность работы в качестве калькулятора, поддерживающего пользовательские переменные и функции. Освоение базового функционала программы для решения задач учебного курса вряд ли отнимет у читателя много времени и сил. Однако возможности gnuplot несравненно шире — при наличии опыта с помощью этой небольшой бесплатной программы можно обрабатывать данные и строить высококлассные графики для публикаций в научных журналах и книгах. В частности, все графики для иллюстрации данного учебного пособия выполнены в gnuplot.

Многие читатели могут усомниться в необходимости знакомства с gnuplot, предпочтя продолжить использование для построения графиков уже знакомых им программ, таких как Micfosoft Excel, Microcal Origin и т. п. В этой связи нужно обратить внимание на ещё одно важное достоинство gnuplot — возможность *пакетного* (программируемого) построения графиков. Анализ большого количества результатов численного моделирования зачастую требует построения значительного числа однотипных графиков, полученных при разных наборах параметров задачи, — десятков, а иногда и нескольких сотен. Построение столь большого количества графиков в Excel или Origin потребует значительных затрат времени и рутинной работы, выполнение которой разумно поручить компьютерной программе, сохранив свои силы для решения более творческих задач.

Таким образом, если читатель знаком с программами, поддерживающими пакетный режим построения графиков, мы оставляем ему свободу выбора инструментов из числа таких программ. В противном случае настоятельно рекомендуем потратить час на освоение основных приёмов работы с gnuplot как с одной из наиболее простых программ с достаточно богатым функционалом.

Для начала работы в gnuplot под Windows найдите и запустите файл wgnuplot.exe (под Linux наберите `gnuplot` в консоли). Программа запустится, выдав в текстовом окне приглашение для ввода команд:
`gnuplot>`

1.1. Команды

Gnuplot управляется текстовыми командами. Для выполнения любого действия в gnuplot — будь то построение графика, вывод на экран значения переменной или математического выражения, изменение настроек программы — нужно отдать соответствующую команду. Например, если набрать в терминале `print pi/2` и нажать клавишу Enter, gnuplot напечатает на экране приближённое значение $\pi/2$. Таким образом, gnuplot можно использовать в качестве достаточно удобного программируемого калькулятора, поддерживающего именованые переменные, пользовательские функции и историю команд.

Для построения графиков используется команда `plot`. Например, результатом выполнения `plot sin(x)` станет график синуса. Очевидно, вместо `sin(x)` можно написать любую функцию, например `plot cos(x)*exp(-x*x)`. Можно предварительно определить новую функцию, отдав команду наподобие[2] `gauss(x)=exp(-x**2)` или `sign(x)=(x==0)?0:((x<0)?-1:1)`[3], и использовать её далее наравне со стандартными математическими функциями.

Часто бывает, что новая команда очень похожа (или даже в точности совпадает) с использованной ранее. Например, после графика гауссовской функции нам потребовалось построить `plot sin(x)/x`. В таком случае нет необходимости набирать текст заново: gnuplot «помнит» историю команд, и, нажимая стрелки «вверх» / «вниз» на клавиатуре, можно перемещаться по ней, вызывая предыдущую / следующую команду соответственно.

Из числа других полезных и часто используемых команд следует отметить `set` — установить (изменить) значение какого-либо параметра настройки gnuplot (например, `set title "sample graph"` изменит название графика на `sample graph`). Чтобы новое название появилось на графике, его нужно построить заново. Для этого нужно либо повторить последнюю команду `plot` со всеми её параметрами, либо скомандовать `replot`. Для любителей мыши кнопка replot есть в панели инструментов окна графиков — нажатие на неё также приведёт к перерисовке графика с новыми параметрами.

Команда `cd`[4] изменяет текущую рабочую директорию, в которой gnuplot будет искать файлы с таблицами данных. Опять-таки предусмотрена альтернатива в виде пункта меню File — Change Directory,

---

[2] Выражение `x**y` в gnuplot обозначает x^y, возведение в степень; напомним, что в языке Си для этого используется функция `pow(x,y)`.

[3] Синтаксис тернарного оператора в gnuplot аналогичен принятому в языке Си: *<условие>* `?` *<выражение-если-истина>* `:` *<выражение-если-ложь>*.

[4] Сокращение от *change directory*.

позволяющая достичь того же результата с помощью мыши или «горячих» клавиш. Узнать текущую рабочую директорию можно с помощью команды `pwd`[5] или через меню File — show Current Directory.

Для получения справки предусмотрена команда `help`. Например, `help plot` показывает информацию об использовании команды `plot` и т. п. Полный список поддерживаемых команд можно посмотреть, набрав `help commands` в gnuplot. Для удобства ниже мы приводим краткий список наиболее важных для наших целей команд gnuplot:

`plot`	построение графиков
`replot`	обновление графиков; добавление нового графика
`splot`	построение трёхмерных графиков
`set`	изменение параметра
`unset`	сброс параметра
`print`	печать текстового сообщения
`fit`	аппроксимация сеточной функции заданным выражением
`load`	выполнение набора команд из файла

1.2. Построение графиков сеточных функций

Для простоты мы начали знакомство с gnuplot с построения графиков на примере элементарных библиотечных функций, однако при численном решении задач нам понадобится прежде всего отображение графиков функций, заданных с помощью таблиц значений: именно в таком виде обычно получаются результаты численных расчётов. Чтобы рассмотреть на примере построение графика сеточной функции, сначала мы должны создать текстовый файл с числовыми данными. Сделать это можно либо вручную в текстовом редакторе, либо в самом gnuplot[6] или же написав программу, создающую для нас такой файл. Выберем последний вариант, что позволит нам дополнительно освежить в памяти синтаксис языка Си.

```
#include <stdio.h>
#include <stdlib.h>
#define _USE_MATH_DEFINES      //для использования M_PI в MSVC
#include <math.h>
```

[5] Сокращение от *print working directory*.

[6] Для этого нужно перенаправить печать текстовых сообщений в файл, скомандовав, например, `set print "out.txt"`, после чего напечатать набор чисел с помощью команды `print` и вернуть вывод в стандартный поток stderr, набрав `set print` без аргументов.

```
//функция, значения которой мы будем сохранять в файл:
double f(double x)
{
    return exp(-3.*pow(0.1*x,2.));
}

//выполнение программы на языке Си начинается с вызова main:
int main(void)
{
    int i, N=100;                //N - число точек в таблице
    //независимая переменная x и верхний предел её перебора:
    double x, x_max=10.;
    //создаем файл out.txt для сохранения таблицы данных:
    FILE *fd = fopen("out.txt", "wb");
    if(fd == NULL) { //если не удалось открыть файл...
        //сообщаем об ошибке и выходим из программы:
        fprintf(stderr, "Could not open file.\n");
        exit(1);
    }

    //записываем в файл строку с названиями столбцов:
    fputs("x    sin(x)    f(x)\n", fd);

    //в цикле перебираем точки при записи таблицы с данными:
    for(i = 0;  i < N;  ++i) {
        x = i*x_max/N;
        //записываем в каждую строку по три значения:
        fprintf(fd, "%f %.8f %15.9e\n", x, sin(x), f(x));
    }

    //закрываем файл:
    fclose(fd);

    //выход из функции main завершает выполнение программы:
    return 0;               //0 - код успешного выполнения
}
```

Мы не будем останавливаться на разборе этой простой программы, посоветовав читателю в случае возникновения вопросов обратиться за помощью к Google, Интернет-ресурсам [C3, C4] либо книге [1]. Обратим

лишь внимание на способ печати чисел с плавающей точкой: функциям семейства `printf` необходимо передать в символьной строке указание `"%f"` для каждого значения (либо `"%e"` для вывода числа в «научном» виде). Попутно можно указать опции наподобие количества разрядов после запятой и общего количества символов в записи числа (напр., `"%15.9e"`), что регулярно бывает необходимо на практике.

Скомпилировав эту программу и запустив её, мы получим файл `out.txt` с тремя столбцами чисел. Полагая, что этот файл лежит в текущей рабочей директории gnuplot (см. команды `cd` и `pwd`), построим графики, скомандовав:

```
plot "out.txt" using 1:2 with lines title 'sin', \
     ""        u     1:3 w    l     t     'gauss'
```

Ключевое слово `using` (и его лаконичная форма `u`) позволяет выбрать столбцы, используемые для построения графика. В частности, запись `u 1:3` означает, что мы хотим построить график зависимости третьего столбца от первого. Ключевое слово `title` (его также можно сократить до одной буквы `t`), позволяет давать графикам названия, которые будут отображаться на легенде (см. `help key`). Можно подписывать графики на легенде выборочно, для чего служит опция `notitle`, например: `plot sin(x) t 'sin', cos(x) notitle`. Команда `set key noautotitle` позволяет отображать в легенде только те графики, для которых была явно указана опция `title`. Обратная наклонная черта (slash) в конце строки позволяет разбить слишком длинную команду на несколько строк, «экранируя» символ перевода строки. Пустые кавычки вместо имени файла во второй строке позволяют не повторять ввод одного и того же имени при построении нескольких графиков из одного файла (в данном случае — `out.txt`).

Часто возникает необходимость при построении графика выполнить некоторые преобразования исходных данных, записанных в файл. Например, для преобразования значений в столбце 1 из сантиметров в километры в gnuplot следует набрать команду: `plot "out.txt" using ($1*1e-5):($2)`. Можно выполнять операции сразу над значениями из нескольких столбцов — перемножать, складывать, использовать прочитанные из файла значения в качестве аргумента произвольных функций. Например, если мы хотим построить график зависимости мощности тока от времени имея файл, в столбцах которого записаны время, сила тока и напряжение в каждый момент времени, нам потребуется записать что-нибудь наподобие `plot "file.dat" using ($1):($2*$3) with lines`.

При построении графиков физических величин, изменяющихся на несколько порядков, используется логарифмический масштаб осей. Для отображения графиков на логарифмической оси y нужно дать команду `set logscale y`, в двойных логарифмических осях — `set logscale xy`. Для возврата к линейному масштабу осей следует использовать команду `unset logscale`.

Чтобы изменить способ подписи значений на осях, используется команда `set tics format "`*формат*`"`, где *формат* кодируется текстовой строкой аналогично функциям Си семейства `printf`. Подробную информацию и примеры использования можно посмотреть в разделе справки gnuplot, набрав `help format specifiers`.

Приведённой выше информации о gnuplot вполне хватит для начала работы (возможно, и для успешного выполнения всех задач учебного курса). Ниже будет рассмотрен также ряд «дополнительных» приёмов работы с gnuplot, позволяющих строить более красивые и информативные графики (далеко не исчерпывающий, однако, всех возможностей данной программы). Вместе с тем мы настоятельно рекомендуем освоить вначале «базовые» команды и опции, поэкспериментировав самостоятельно с gnuplot и выполнив упражнения, приведённые в конце главы, и только после этого двигаться дальше.

1.3. Дополнительные возможности команды plot

Команду `plot` можно использовать для построения нескольких графиков в одних осях, для этого нужно указать функции через запятую, как в примере в предыдущем параграфе. Альтернативный вариант — использование команды `replot`:

```
plot 0.5*exp(-0.25*x**2)  #строим новый график взамен старого;
replot exp(-x**2)}  #добавляем график на существующий рисунок.
```

При построении нескольких графиков любым из указанных способов gnuplot автоматически изменяет стиль отрисовки каждой следующей функции. Возможна также «ручная» настройка, для чего в конце записи команды `plot` можно указывать `with points` для отрисовки графика с помощью точек (маркеров), `with lines` — для отрисовки линиями, `with linespoints` — линиями с точками и т. п. Полный перечень возможностей gnuplot см. в разделе справки `help plotting styles`. Лаконичный вариант тех же опций: `w p`, `w l` и `w lp`. Например, команда `plot sin(x) w lp` нарисует график $\sin x$ линиями с маркерами. При желании можно управлять толщиной (`linewidth`), типом

(`linetype`, `dashtype`[7]) и цветом (`linecolor`) линий, стилем и размером маркеров (подробности см. в разделе справки `help with`, где перечислены всевозможные способы построения графиков и доступные для них опции). Список названий цветов и их RGB представление можно посмотреть, набрав команду `show colornames`. Также можно указывать цвета в виде текстовой строки вида `#rrggbb`, где `rr`, `gg` и `bb` — шестнадцатиречные коды красной, зелёной и синей компонент. Например, строка `"#ff0000"` кодирует красный цвет, а `"#ffff00"` — жёлтый. Чтобы каждый раз не указывать в команде `plot` потенциально длинный список стилей отображения линий и маркеров, их можно определить командой `set style line` <*номер-линии*> <*список-стилей*>[7]. Параметрами отображения легенды на графике также можно управлять (подробности см. в справке `help key` либо набрав `show key` для отображения текущих настроек).

Заметим, что даже если график функции представляет собой гладкую линию, gnuplot вычисляет значения функции при построении графика в ограниченном числе точек (`samples`), соединяя их линиями. При построении графиков быстро осциллирующих функций заданного по умолчанию числа точек (100) может оказаться недостаточно, в этом случае следует использовать команду наподобие `set samples 500`.

Обязательным требованием редакций научных журналов и просто правилом хорошего тона является наличие подписей осей на графиках. Для этого в gnuplot предусмотрены команды `set xlabel`, `set ylabel`. Например, команда `set xlabel "Время, с"; set ylabel "Мощность, Вт"` будет уместна при построении графика зависимости мощности от времени. Точка с запятой позволяет записывать несколько команд подряд в одной строке, что иногда делает особенно удобным использование истории команд, доступной при нажатии стрелок «вверх» и «вниз» на клавиатуре. Указать название для графика в целом можно с помощью команды `set title`. Напомним, что внесённые командами `set` изменения станут заметны лишь после следующего вызова `plot` или `replot`.

Чтобы изменить пределы, в которых gnuplot построит график, можно указать требуемые значения в квадратных скобках после команды `plot`. Так, команда `plot [0:2*pi] sin(x)` построит график синуса на промежутке от 0 до 2π; команда `plot [0:2*pi][-0.1:1.1] sin(x)` дополнительно ограничит отображаемый интервал по вертикальной оси от -0,1 до 1,1. Другая возможность решения той же задачи — команды `set xrange [0:2*pi]` и `set yrange [-0.1:1.1]`. Если необходимо, чтобы gnuplot выбирал то или иное значение предела самостоятельно

[7] Начиная с версии 5 gnuplot.

в зависимости от отображаемых данных, следует использовать символ «звёздочка». Например, команда `set yrange [0:*]` заставит gnuplot рисовать графики на положительной полуоси y, выбирая верхний предел отображаемых значений автоматически.

Для отображения графиков в полярных координатах $r(t)$ предусмотрен режим `polar`:

```
set polar     #построение графиков в полярных координатах r(t)
plot t        #строим график r(t)=t - спираль
unset polar   #возвращаемся к декартовым координатам x,y
```

Gnuplot позволяет строить графики функций, заданных параметрически $(x(t), y(t))$. Например, следующий код нарисует две фигуры Лиссажу (рис. 1 (*а*)):

```
set parametric  #активируем режим построения параметр. функций
set xrange [-1.3:1.3]; set yrange [-1.3:1.3]  #диапазон по x,y
set xlabel 'X' ; set ylabel 'Y'            #подписи осей (X,Y)
set samples 500           #будем строить графики по 500 точкам
set key off          #отключаем отображение легенды на графике
set xtics 1.0 ; set ytics 1.0  #определяем шаг штрихов по осям
set mxtics 5 ; set mytics 5  #количество промежуточных делений
set style line 1 linecolor "gray70" lw 3  #стиль1 линий граф.
set style line 2 lc "black" linewidth 1 #стиль2 линий графиков
plot [0:2*pi] sin(t)/2,cos(3*t)/2 ls 1, \
              sin(5*t),cos(3*t) ls 2
unset parametric  #возвращаемся в режим построения явных ф-ций
```

Часто в физике возникает необходимость отобразить на одном графике величины разных порядков. Для этого удобно использовать парные оси y и y_2 слева и справа от графика. В качестве примера рассмотрим построение графика функции Бесселя $J_0(x)$, её асимптотики $\sqrt{2/(\pi x)} \cdot \cos(x - \pi/4)$ и модуля их разности (рис. 1 (*б*)):

```
asj0(x)=sqrt(2./pi/x)*cos(x-pi/4)    #определяем новую функцию
set samples 500           #будем строить графики по 500 точкам
set xlabel 'X' ; unset ylabel      #подписи нижней и левой оси
set y2label 'Невязка' textcolor "gray50"   #подпись правой оси
set xrange [0.01:24]    #диапазон по X при построении графиков
set yrange [-2:1.3]     #диапазон, отображаемый на левой оси Y
set y2range [3e-4:3]    #диапазон изменения на правой оси (Y2)
set xtics 5 ; set ytics nomirror (-0.5,0,0.5,1,1.5)    #штрихи
set my2tics 1          #промежуточные деления на правой оси Y2
```

```
set key right top font "Arial,11"           #параметры легенды
#показываем штрихи на оси Y2, выбираем формат и цвет чисел:
set y2tics format "10^{%T}" textcolor "gray50"
set logscale y2  #используем логарифмич. масштаб на правой оси
set style line 1 linecolor "black" linewidth 2.5 #определяем>>
set style line 2 lc "gray50" lw 1        #стили линий графиков
plot besj0(x) with lines linestyle 1 title 'J_0(x)', \
     abs(besj0(x)-asj0(x)) ls 2 axes x1y2 t 'невязка'
```

Другим примером использования двойных осей является отображение величин в разных размерностях: в калориях и Джоулях, МэВ и атомных единицах, ГГц и см$^{-1}$ и т. п. В оптике часто бывает необходимо строить спектры на связанных осях частот и длин волн как показано в следующем примере и на рис. 1 (*в*):

```
set xlabel 'Частота, ТГц'              #подпись нижней оси (X)
set ylabel 'Дисперсия {/Symbol b}_2, пс^2/км'   #подпись оси Y
set x2label 'Длина волны, мкм'       #подпись верхней оси (X2)
set xrange [190:610]    #диапазон по X при построении графиков
set xtics nomirror 100 #отменяем зеркальные штрихи, задаём шаг
set ytics 100                      #задаём шаг штрихов по оси Y
set x2tics (0.5,0.6,0.8,1,1.4)    #положение штрихов на оси X2
#Определяем связь верхней оси X2 с нижней X (поддерживается >>
set link x2 via 300/x inverse 300/x    #начиная с Gnuplot v.5)
set key off          #отключаем отображение легенды на графике
set grid                         #показываем координатную сетку
set label 1 "(в)" at screen 0.1, screen 0.9 font ",14"  #метка
#показываем штрихпунктирной линией ноль дисперсии на графике:
set arrow 1 from 420,graph 0 to 420,graph 1 \
    nohead lc "gray40" dashtype ".-._"
set label 2 "{/Symbol l}_0" at 410,0 offset 0.55,-0.4
set terminal pdfcairo enhanced color lw 0.75 \
    font "Arial,12" size 5cm,5cm
set output 'DispersionTF.pdf'     #перенаправляем вывод в файл
plot 'beta2_d=2000nm.dat' u ($1*500/pi):($2) w l lc "black"
```

По умолчанию, каждый следующий вызов команды `plot` приводит к очистке построенного ранее графика. В случае, когда необходимо совместить несколько графиков в виде панелей (подобно тому, как организован рис. 1), либо сделать врезку на графике, можно использовать режим `multiplot`, активируемый и отключаемый командами `set` и `unset`.

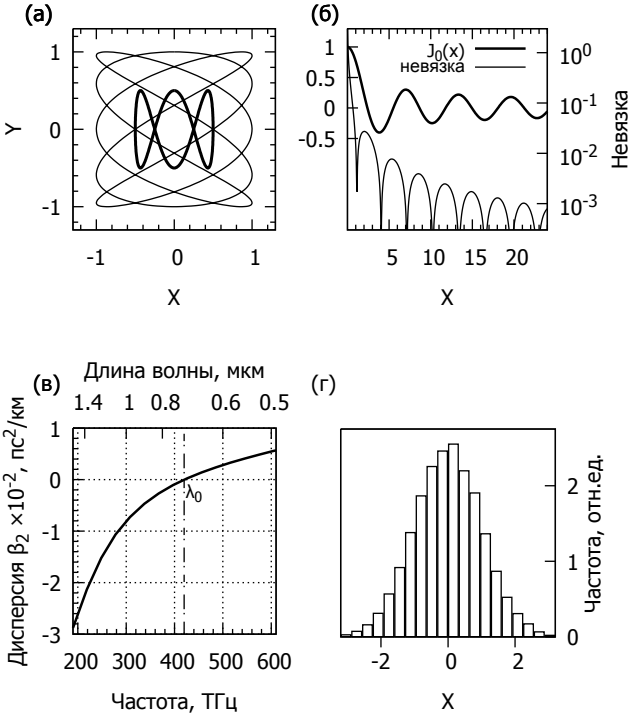


Рис. 1. Построение графиков в gnuplot: (а) параметрические функции, (б) отображение величин разных порядков, (в) связанные оси, (г) гистограмма

Повысить информативность графика и сделать более понятным его описание можно, добавив на график дополнительные подписи и метки (`label`), а также линии и стрелки (`arrow`). Пример использования меток и линий показан на рис. 1 (*в*) — вертикальной штрихпунктирной линией показана длина волны нулевой дисперсии λ_0. Заметим, что gnuplot позволяет использовать пять способов задания координат для указания положения произвольных линий, стрелок, текста и легенды. Их называют «первая» (`first`), «вторая» (`second`), «графическая» (`graph`), «экранная» (`screen`) и «символьная» (`character`). «Первая» координата отсчитывается по нижней оси X и левой оси Y, «вторая» — по верхней оси X2 и правой оси Y2. «Графическая» координата опре-

деляет относительную позицию точки в рамке графика[8]. «Экранная» координата отсчитывается относительно всего экрана (бумаги), при этом начало отсчёта расположено в левом нижнем углу, точка (1,1) — в правом верхнем. Если тип координаты не указан явно, по умолчанию предполагается, что задана «первая» координата. Иногда удобно бывает комбинировать координаты разных типов — в приведённом выше примере было использовано:

```
set arrow 1 from 420, graph 0   to   420, graph 1
```

Такой способ позволяет провести вертикальную линию через заданную точку $\nu_0 = 420$ ТГц от низа (координата `graph 0`) до верха графика (`graph 1`) независимо от выбранных диапазонов отображения `xrange`, `yrange` по осям. Более подробную информацию можно получить, набрав `help coordinates`.

С помощью gnuplot можно строить гистограммы и функции распределения по выборке реализаций случайной величины, сохранённой в текстовом файле. Например, на рис. 1 (*г*) представлена гистограмма плотности стандартного нормального распределения, построенная[9] по выборке размером 10 000 с помощью команд:

```
n=20; min=-3.2; max=3.2    #количество каналов и диапазон по x
cw=(max-min)/n                #ширина одного канала гистограммы
hist(x,width)=width*floor(x/width)+width/2.0
plot "rnd.dat" u (hist($1,cw)):(1e-3) smooth freq w boxes
```

Для построения гистограммы в примере выше была определена функция `hist`, задающая «ступенчатое» отображение своего первого аргумента. Ширина и высота «ступенек» равна значению переменной `cw`. Входные данные разбиваются функцией `hist` по каналам гистограммы, после чего вызывается команда построения графиков `plot` с опцией дополнительной обработки данных `smooth` в режиме `frequency`. В этом режиме функция `plot` отображает одинаковые значения x из входного потока в виде одной точки, y-координата которой равна сумме значений y соответствующих точек из входного потока. Для построения плотности вероятности по выборке y-координаты всех точек должны быть одинаковы (в данном примере они равны 10^{-3}), а x-координаты точек

[8] При этом положение самой рамки может меняться в зависимости от наличия подписей осей, штрихов, а также под действием команд `set size` и `set origin`.

[9] Для генерации случайной величины $\sim \Phi_{0,1}$ в gnuplot использовалось преобразование Бокса — Мюллера; равномерно распределенная на отрезке $[0,1]$ случайная величина генерировалась с помощью функции `rand(0)`.

должны принимать относительно небольшое число возможных значений за счёт группировки близких точек в каналы гистограммы, что и достигается в примере выше указанием `using (hist($1,cw)):(1e-3)`.

1.4. Графики функций двух переменных

Для построения графиков функций двух переменных в gnuplot предусмотрена команда `splot`[10]. Её синтаксис в целом аналогичен `plot`, основное отличие состоит в формате файлов с данными. Наиболее простое решение — формировать строки вида $x \quad y \quad z_1 \quad \ldots \quad z_n$, вставляя по одной пустой строке между данными, соответствующими различным значениям x. В качестве примера приведём код программы на языке Си для записи таблицы аналитичекого решения одномерного уравнения теплопроводности $(\partial_t - \partial_x^2)G(x,t) = 0$ с начальным условием $G(x,0) = \delta(x)$:

```
#include <stdio.h>
#include <stdlib.h>
#define _USE_MATH_DEFINES //для использов. M_PI в Microsoft VC
#include <math.h>

//функция Грина 2 рода одномерного уравнения теплопроводности:
double G(double x, double t)
{
   return exp(-x*x/(4*t))/sqrt(4*M_PI*t);
}

int main(void)
{
   int i, j, N=50; //i,j - счётчики, N - число точек в таблице
   double t, x, t0=0.5, dt=0.05, dx=0.2;  //шаг сетки по t и x
   FILE *fd = fopen("solution.dat", "wb");
   if(fd == NULL) //если не удалось открыть файл, завершаем >>
      exit(1);    //    выполнение программы с кодом ошибки 1
   for(i = 0;  i < N;  ++i) {  //цикл перебора узлов сетки t_i
      //вложенный цикл - перебор узлов x_j, запись данных:
      for(j=0, t=t0+i*dt;  j<N;  ++j) {
         x = (j-N/2.)*dx;
         fprintf(fd, "%12.5e %12.5e %12.5e\n",  t, x, G(x,t));
      }
```

[10] Сокращение от *surface plot*.

```
      //пустая строка, разделяющая различные сечения t:
      fputs("\n", fd);
   }
   fclose(fd);   //закрываем файл
   return 0;     //возвращаем код успешного завершения
}
```

Скомпилируем и запустим программу, наберём в gnuplot команду `splot "solution.dat" using 1:2:3 with pm3d`, в результате чего получим график функции $G(x,t)$ в виде цветной поверхности[11]. Отображение множества значений математической функции на цвета называется *палитрой* (`palette`) и может быть изменено с помощью команды `set palette` с соответствующими опциями. Например, для выбора серой градиентной палитры можно скомандовать:

```
set palette defined ( 0 "black", 1 "white" )
```

Палитра, приблизительно соответствующая палитре MATLAB по умолчанию:

```
set pal defined (1 '#00008f',  8 '#0000ff', 24 '#00ffff', \
                40 '#ffff00', 56 '#ff0000', 64 '#800000')
```

Для построения поверхности $G(x,t)$ одним цветом вместо `pm3d` можно использовать построение с помощью линий (`with lines`). Чтобы сделать поверхность непрозрачной, необходимо скомандовать `set hidden3d`. По умолчанию поверхность графика будет приподнята над плоскостью xy; чтобы это исправить, воспользуемся командой `set xyplane relative 0`.

Угловые координаты наблюдателя и масштабные коэффициенты при построении поверхностей можно изменить с помощью команды `set view`. Команда `set view map` переводит gnuplot в режим отображения графиков в виде плоской двумерной карты, на которой значения функции отображаются цветом. Чтобы показать линии уровня на графике, используется команда `set contour`.

Если добавлять в файл не по одной, а по две пустых строки между сечениями по времени, можно построить зависимости от координаты x, нарисованные отдельными линиями. Для этого нужно скомандовать `splot "solution.dat" using 1:2:3 with lines`. При этом можно строить сечения не для всех моментов времени t, сохранённых в файле, а указать требуемые с помощью ключевого слова `index` либо его лаконичной формы `i`.

[11] `pm3d` — сокращение от *palette-mapped 3d*.

1.5. Сохранение графических файлов

Для сохранения построенных с помощью gnuplot графиков в файл можно использовать кнопку слева на панели инструментов. Полученное таким способом изображение можно быстро отправить по электронной почте или вставить на веб-страницу, тогда как для подготовки печатных документов его использовать не следует ввиду низкого разрешения — в этом случае более подходящими являются векторные форматы графических файлов, в которых рисунок сохраняется в виде набора линий и других графических объектов, что позволяет получать чёткие высококачественные изображения любого размера.

Рассмотрим сохранение графиков на примере формата EMF, подходящего для последующей вставки рисунков в приложения MS Office:

```
#изменяем терминал вывода:
set terminal emf enhanced lw 4 size 1024,960 font "Arial,24"
set output 'figure.emf'   #перенаправляем вывод gnuplot в файл
plot sin(x)               #строим график
set output                #восстанавливаем вывод в STDOUT
set term wxt              #восстанавливаем исходный терминал
```

Команда `set terminal` *имя-терминала* (краткая форма `set term` *имя-терминала*) изменяет терминал, используемый для построения графиков. Терминалы gnuplot, подобно физическим устройствам, отличаются друг от друга набором возможностей для построения графиков (палитрами, возможностями использования различных шрифтов и ширин линий). В частности, gnuplot поддерживает терминалы для построения графиков в форматах png, emf, pdf, LaTeXи даже для формирования интерактивных иллюстраций для html-страниц (`canvas`). Полный список поддерживаемых терминалов можно посмотреть в справке gnuplot (`help terminal`), либо набрав `set terminal` без параметров. При выборе терминала можно указать опции (в примере выше мы увеличили базовую ширину линий до 4, установили размер изображения 1024×960 отн.ед. и выбрали шрифт семейства Arial, кегль 24).

Команда `set output '`*имя файла*`'` открывает графический файл и перенаправляет туда вывод из терминала. Чтобы посмотреть построенный график, может потребоваться вначале закрыть файл, для чего нужно отдать ещё одну команду `set output`, указав в качестве параметра имя нового файла для построения следующего графика либо вызвав `set output` без параметра как в приведённом выше примере. В этом случае вывод будет направлен в стандартный поток (stdout).

1.6. Пакетное построение графиков. Циклы

Очень часто при решении научных и инженерных задач приходится выполнять большое количество однотипных численных расчётов для исследования зависимости результатов от параметров исследуемых физических систем, что сопряжено с построением большого количества однотипных графиков. Gnuplot позволяет автоматизировать этот процесс — помимо рассмотренного выше интерактивного режима работы, когда отдельные команды вручную вводятся пользователем в консоли gnuplot, предусмотрен также пакетный режим работы. Для этого нужно создать текстовый файл, содержащий набор команд gnuplot, назовём его `sample.txt`:

```
# построение графиков в пакетном режиме
set terminal png size 320,240
set output "sample.png"
set xlabel "x" ; set ylabel "y" ; set title "sample plot"
set xrange [-pi:pi] ; set yrange [*:1.1]
plot sin(x), cos(x)*exp(-x*x)
set output        #восстанавливаем вывод в STDOUT
set term pop      #восстанавливаем исходный терминал
```

Для того, чтобы «запустить» написанную для gnuplot программу, нужно либо передать gnuplot командный файл при запуске[12], либо набрать в консоли gnuplot команду `load "sample.txt"`. Первый способ позволяет быстро строить большое количество графиков с использованием скриптов оболочки операционной системы. Вместо созданного вручную командного файла часто бывает удобно использовать выходной поток такого скрипта, перенаправленный на вход gnuplot. Однако прежде чем переходить к автоматизации, вначале рекомендуем хорошо освоить работу в «ручном» режиме.

В наиболее простых случаях автоматизацию можно выполнить средствами gnuplot, не прибегая к скриптам оболочки. При этом оказываются полезными условный оператор `if { ... } else { ... }` и циклы `for`, которые могут быть использованы совместно с командами `plot`, `splot`, `set` и `unset`. Цикл `for` позволяет вести перебор по строкам `for [filename in "A B C D"]` либо по целым числам:

```
set logscale xy  ; set key left bottom
set xrange [1:3] ; set ytics format "10^{%T}"
plot for[n=2:10:2] x**(-n) t sprintf("x^{%d}",n)
```

[12] Под Windows: `wgnuplot sample.txt` или `pgnuplot < sample.txt`; под Linux: `cat sample.txt | gnuplot` или `gnuplot < sample.txt`.

Функция `sprintf` позволяет сформировать текстовую строку, содержащую значения переменных, и использовать её в качестве имени файла, подписей на легенде, осях, метках и т. п. Синтаксис `sprintf` в gnuplot очень близок к синтаксису одноимённой функции в языке Си.

Выполнение блока произвольных команд в цикле может быть реализовано с помощью `do for [`*область*`] {` *команды* `}` либо конструкции `while(`*выражение*`) {` *команды* `}`. Циклы могут быть вложенными:

```
set for [i=1:9] for [j=1:9] label i*10+j \
    sprintf("%d",i*10+j) at i,j
```

В некоторых случаях для организации циклов удобно использовать команду `reread` внутри командного файла для повторного выполнения команд с начала файла. Для суммирования конечного числа членов последовательности или ряда предусмотрена специальная команда `sum` (см. `help summation`).

Таким образом, в данной главе были кратко рассмотрены основные возможности программы gnuplot, и даны сведения, необходимые для начала работы. Более подробную информацию легко найти в документации и многочисленных блогах, форумах, списках F.A.Q. и HowTo, посвящённых gnuplot. Большое количество полезных примеров от базовых до нетривиальных можно найти в замечательно иллюстрированной книге [A3]. Желающим более полно и систематически изучить возможности программы можно рекомендовать [A4].

Упражнения

1. Напишите программу, сохраняющую в текстовый файл с двумя столбцами таблицу значений гауссовской функции. Запустите программу, постройте график функции в gnuplot.

2. Выполните упражнение 1, сформировав файл с данными средствами gnuplot.

3. Постройте график функций Бесселя $J_0(x)$ и $J_1(x)$ нулевого и первого порядков (`besj0` и `besj1`) на интервале от -20 до $+20$.

4. Постройте график зависимости разности функции Бесселя J_1 первого порядка (`besj1`) и её асимптотики $J_1(x) \sim \sqrt{2/(\pi x)} \times \times \cos\left(x - \frac{3}{4}\pi\right)$ на интервале от 1 до $+100$ в логарифмическом масштабе. Как убывает исследуемая разность с ростом x?

5. Исследуйте с помощью gnuplot абсолютную и относительную погрешность аппроксимации Гамма-функции Эйлера $\Gamma(x+1) = \int_0^\infty t^x e^{-t} dt$ частичными суммами ряда Стирлинга

$$\Gamma(x+1) = \sqrt{2\pi x}\left(\frac{x}{e}\right)^x \left(1 + \frac{x^{-1}}{12} + \frac{x^{-2}}{288} - \frac{139x^{-3}}{51840} - \frac{571x^{-4}}{2488320} + \dots\right).$$

6. Исследуйте с помощью gnuplot абсолютную и относительную погрешность аппроксимации дополнительной функции ошибок $\operatorname{erfc} x = 1 - \operatorname{erf} x$ частичными суммами асимптотического разложения

$$\operatorname{erfc} x = \frac{2}{\sqrt{\pi}} \int\limits_x^\infty e^{-t^2}\, dt \sim \frac{e^{-x^2}}{x\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{\Gamma(n+\frac{1}{2})}{\Gamma(\frac{1}{2})} \left(-x^2\right)^{-n}.$$

7. Исследуйте с помощью gnuplot абсолютную и относительную погрешность аппроксимации функции Эйри (`airy`) при $x \to -\infty$ главным членом асимптотики

$$\operatorname{Ai} x \sim \frac{1}{|x|^{1/4}\sqrt{\pi}} \cos\left(\frac{2}{3}|x|^{3/2} - \frac{\pi}{4}\right).$$

8. Выполните команду `plot [0:99][-1.5:1.5] cos(2*x*pi)`, объясните полученный результат. Что нужно сделать, чтобы график соответствовал интуитивным ожиданиям?

9. Используя данные из файла, полученного в упражнении 1, в качестве начальных условий $u(x, 0)$, постройте в gnuplot графики точного решения уравнения Хопфа $u_t - uu_x = 0$ при $t =$ 0, 0,5, 1,0, 1,5[13].

[13] Напомним, что точное решение квазилинейного уравнения Хопфа может быть получено методом характеристик в неявном виде $u(x,t) = g(x - ut)$.

10. Используя скрипты gnuplot вместо соответствующих программ на языке Си, выполните упражнение № 1 на с. 40; № 5 на с. 55; № 3 на с. 68.

2. Машинные числа

В повседневной практике мы настолько привыкли иметь дело с числами, что кажется совершенно невероятным встретить в этой области что-то нетривиальное. И тем не менее, близкое знакомство с вычислениями на ЭВМ способно преподнести немало сюрпризов. Законы машинной арифметики, как будет показано ниже, неизбежно отличаются от тех, к которым мы привыкли с первых классов школы. Погрешности округления, которые на первый взгляд кажутся ничтожными по сравнению с привычными масштабами точности данных, могут приводить к катастрофическим последствиям — сбоям в программе и ответам без единой верной цифры. Чтобы избежать подобных ошибок и суметь исправить их в случае, если они всё-таки возникли, необходимо иметь представление о том, как комьютер работает с числами. Этому и посвящена данная глава.

Наиболее фундаментальное отличие чисел, используемых компьютером, от привычных математических абстракций заключается в конечности множества машинных чисел. По сути, всё остальное является прямым или косвенным следствием этого достаточно очевидного факта. Действительно, независимо от способа представления чисел в ЭВМ, в силу ограниченности объёма компьютерной памяти, множество используемых чисел будет конечным. Прямыми следствиями данного факта являются наличие минимального и максимального чисел в ЭВМ и обусловленный этим эффект переполнения разрядной сетки, а также ограниченная точность вычислений. Вопрос о конкретных пределах и ограничениях не имеет однозначного ответа, поскольку зависит от способа представления чисел, используемого в программе. Мы ограничимся использованием стандартных типов данных, поддерживаемых как языком Си, так и современными процессорами. Однако возможно применение дополнительных библиотек для работы с числами более высокой разрядности. Такой подход используется на практике для решения задач, где ограничения, накладываемые стандартными типами данных, оказываются неприемлимыми. Вычисления с более высокой точностью реализованы в системе Wolfram Mathematica, а также в ряде библиотек C/C++, таких как GNU GMP (GNU multiple precision arithmetic library), GNU MPFR (GNU Multiple Precision Floating-Point Reliably)

и boost/multiprecision. Предельная точность вычислений определяется при этом объёмом доступной памяти и требуемой скоростью расчётов.

2.1. Целые числа

Все данные хранятся в памяти компьютеров в двоичном виде, представляя собой последовательность *битов*, каждый из которых может принимать значения 0 или 1. Количество целых чисел, которое может кодировать последовательность $\{a_j\}$ из p бит, есть, очевидно, 2^p. Интерпретация последовательности из p бит как беззнакового числа $\sum 2^j a_j$ позволяет кодировать числа от 0 до 2^p-1, для чисел со знаком — от -2^{p-1} до $2^{p-1}-1$. Кодирование отрицательных чисел реализовано с использованием так называемого *дополнительного кода*: отрицательное число $-M$ записывается как положительное $2^p - M$, дополняющее его до последовательности нулей длиной p. На первый взгляд такой способ кодирования отрицательных чисел может показаться несколько странным. Заметим, однако, что при этом арифметические операции проводятся над группой остатков от деления $\bmod 2^p$. При этом сложение и вычитание выполняются одинаково для знаковых и беззнаковых чисел, отличия в умножении и делении также незначительны.

Например, число -1_{10} кодируется в ЭВМ последовательностью $11\ldots1_2$ (здесь и далее в этой главе нижним индексом после числа мы будем обозначать основание системы счисления). Та же самая последовательность битов соответствует наибольшему беззнаковому p-битовому целому числу. Результат сложения $11\ldots1_2 + 00\ldots01_2 = 00\ldots00_2 = 0$. Для чисел со знаком это соответствует $-1_{10} + 1_{10} = 0$. При сложении беззнаковых чисел должно получиться $(p+1)$-значное двоичное число $100\ldots00_2$, однако первый единичный бит выходит за пределы p-значной разрядной сетки — происходит *переполнение* целочисленной переменной, и в качестве результата в целочисленной переменной сохраняются последние p нулевых битов.

Аналогично, число -2_{10} кодируется последовательностью $1\ldots110_2$, -3_{10} соответствует $11\ldots1100_2$ и т. д. Самое большое по модулю отрицательное число равно -2^{p-1} и кодируется последовательностью битов $100\ldots000_2$. Признаком отрицательного числа при использовании дополнительного кода является 1 в старшем двоичном разряде (при вычислении в десятичных числах старший разряд отрицательных чисел лежит в диапазоне от 5 до 9 включительно).

Существует целый ряд типов данных, предусмотренных стандартом языка Си и поддерживаемых современными процессорами: `short int`, `int`, `long int`, `long long int`, а также их беззнаковые (`unsigned`)

модификации[14]. Количество байтов[15], используемое для хранения переменной каждого типа, можно узнать с помощью ключевого слова `sizeof` языка Си. Разрядность целочисленных типов не регламентируется жёстко стандартом Си и может меняться в зависимости от компилятора и системы. Стандартом предписывается лишь, что каждый следующий тип из перечисленных выше имеет по крайней мере не меньшую разрядность, чем предшествующие[16].

2.2. Дробные числа

Для кодирования дробных чисел можно использовать различные подходы. Например, можно представлять дробное число в виде отношения двух целых a/b, сохраняя в памяти ЭВМ пару (a, b). Данный способ неудобен ввиду большой избыточности кода (так, числа $(1, 1)$, $(2, 2)$ и (n, n) при $n \neq 0$ соответствуют одному и тому же числу 1,0), а также относительной сложности выполнения арифметических операций над дробными числами. Избавиться от указанных недостатков можно, используя в качестве делителя b предопределённую константу. В этом случае любое дробное число (a/b) полностью и однозначно определяется делимым a; сложение и вычитание ничем не отличается от операций над целыми числами, умножение и деление лишь немногим сложнее (содержат одну дополнительную операцию по сравнению с целочисленной арифметикой). Данный формат записи дробных чисел называется записью *с фиксированной точкой* (или *фиксированной запятой*) ввиду того, что запись дробной части числа всегда имеет фиксированное число знаков. Эта особенность и определяет применимость такого формата: его следует использовать, когда требуемая *абсолютная* точность вычисления фиксирована и не зависит от результата. Характерным примером являются финансовые расчёты, которые всегда выполняются до копеек, идёт ли речь о покупке коробка спичек или бюджете страны. В финансовых расчётах невозможно зафиксировать *относительную* точность расчётов: при увеличении суммы это приведёт к появлению значительных *абсолютных* погрешностей, что неизбежно вызовет недовольство как минимум одной из сторон финансовых отношений.

[14] Строго говоря, `signed char` и `unsigned char` также являются целочисленными типами [C5], однако для численных расчётов они вряд ли понадобятся.

[15] 1 байт = 8 бит.

[16] Также стандартом предусмотрены опциональные целочисленные типы данных `std::int8_t`, `std::int16_t`, `std::int32_t` и `std::int64_t`, см. заголовочный файл `<cstdint>`.

В противоположность этому, в научных и инженерных расчётах, как правило, фиксирована именно относительная погрешность. Не так важно, измеряем ли мы длину лабораторного стола или расстояние между городами, — допустимые погрешности зачастую будут одного порядка величины. Если попытаться использовать в научных и инженерных расчётах числа с фиксированной точкой (запятой), возникнет проблема с записью «слишком больших» чисел (наподобие числа Авогадро $N_A = 6{,}02 \cdot 10^{23}$ моль$^{-1}$) и неотличимостью от нуля «слишком маленьких» чисел (наподобие постоянной Планка $\hbar = 1{,}05 \cdot 10^{-27}$ эрг·с). Выходом из ситуации служит использование в научных расчётах *чисел с плавающей точкой* (с *плавающей запятой*). Данный формат подразумевает отдельную запись *мантиссы* и *порядка числа*. Например, для записи числа $1{,}05 \cdot 10^{-27}$ необходимо сохранять мантиссу 1,05 и показатель -27. Основание системы счисления (в данном примере равное десяти, для современных ЭВМ — двум) можно не сохранять, т.к. оно одинаково для всех записываемых чисел.

Одна из проблем, которую необходимо было решить для эффективного использования чисел с плавающей точкой в ЭВМ, связана с неоднозначностью их записи. Действительно, одно и то же число $1{,}05 \cdot 10^{-27}$ допускает различные способы записи: $1{,}05 \cdot 10^{-27} = 10{,}5 \cdot 10^{-28} = 0{,}105 \cdot 10^{-26} = \ldots$. Избыточность кода неизбежно приводит к неэффективному использованию памяти ЭВМ и снижению быстродействия. Решением проблемы является использование *нормализованной* формы записи чисел с плавающей точкой, в которой целая часть числа записывается ровно одной цифрой, отличной от нуля: $\hbar = 1{,}05 \cdot 10^{-27}$ эрг·с, $c = 2{,}998 \cdot 10^{10}$ см/с, $\pi = 3{,}14 \cdot 10^{0}$ и т. д.

Для двоичных чисел указанное ограничение означает, что запись мантиссы любого нормализованного числа начинается с 1, так что любое такое число может быть представлено в виде $x = \pm 1{,}M \cdot 2^E$, где M — дробная часть мантиссы числа x, E — двоичный порядок числа (exponent). Поскольку целая часть мантиссы всегда равна 1, в целях оптимизации её не сохраняют в памяти ЭВМ.

Например, число 1 в нормализованном двоичном виде записывается как $+1{,}000 \cdot 2^0$, десятичное число $16_{10} = +1{,}000 \cdot 2^4$, аналогично $-0{,}75_{10} = -1{,}1_2 \cdot 2^{-1}$ (обратите внимание на мантиссу $-1{,}1_2$ — *минус одна целая одна вторая*). Как будет записываться число $\pi \approx 3{,}1416_{10}$ в нормализованном виде с использованием четырёхразрядной мантиссы? Ближайшая к π степень 2 есть 2^1, так что $\pi \approx 1{,}5708_{10} \times 2^1$. Если бы мантисса была десятичной, она бы кодировала десятитысячные доли. Аналогично, в двоичной записи четырёхразрядная мантисса соответствует числителю дроби со знаменателем $2^4 = 16$. Поскольку

$\pi/2 \approx 1{,}5708 \approx 1\frac{9}{16}$, а $9_{10} = 1001_2$, имеем окончательно $\pi \approx +1{,}1001_2 \cdot 2^1$.

В памяти ЭВМ числа записываются в виде последовательности битов (двоичных цифр): самый первый (старший) бит — знак числа S, далее w битов — *смещённый порядок* $E_S = E - (2^{w-1} - 1)$, последние n бит — дробная часть мантиссы числа, $M \cdot 2^n$ (рис. 2). Нормализованное число с плавающей точкой может быть представлено выражением:

$$x = (-1)^S \cdot (1 + M/2^n) \cdot 2^{E_S - (2^{w-1}-1)}. \qquad (1)$$

Знак S числа x может быть равен либо 0 ($x > 0$), либо -1 ($x < 0$). Дробная часть мантиссы может принимать любое из 2^n значений, смещённый порядок E_S меняется от 1 до $2^w - 2$ включительно (ещё два возможных значения $E_S = 0$ и $E_S = 2^w - 1$ зарезервированы, о чем мы поговорим чуть позже). Таким образом, порядок машинных чисел меняется от $E_{\min} = -2^{w-1} + 2$ до $E_{\max} = 2^{w-1} - 1$. Для чисел с плавающей точкой с одинарной точностью $w = 8$, смещение (bias) $2^{w-1} - 1 = 127$, $E_{\min} = -126$, $2^{-E_{\min}} = 1{,}175 \cdot 10^{-38}$, $E_{\max} = 127$, $2^{E_{\max}+1} = 3{,}403 \cdot 10^{38}$.

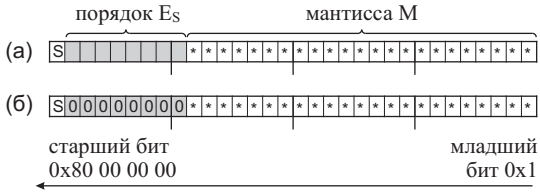


Рис. 2. Запись (а) нормализованных и (б) денормализованных чисел одинарной точности в соответствии со стандартом IEEE 754

Очевидным ограничением нормализованной формы записи чисел с плавающей точкой является невозможность представления нуля и близких к нулю чисел. Действительно, самое маленькое по модулю нормализованное число в ЭВМ равно $f_1 = 1{,}00\ldots00_2 \cdot 2^{E_{\min}}$, тогда как следующее за ним число f_2 будет отличаться от f_1 в последнем знаке мантиссы $f_2 = 1{,}00\ldots01_2 \cdot 2^{E_{\min}}$. Таким образом, даже если кодировать число 0 с помощью некоторой специально зарезервированной последовательности битов, вблизи нуля получится огромный «провал» — область чисел, непредставимых в нормализованном виде. Размер этой «околонулевой ямы», отнесенный к расстоянию между соседними числами, равен $f_1/(f_2 - f_1) = 2^n$, где n — разрядность мантиссы. Для стандартного типа данных с одинарной точностью $n = 23$, $2^{23} = 8{,}4 \cdot 10^6$, для чисел с двойной точностью $n = 52$, $2^{52} = 4{,}5 \cdot 10^{15}$. Буквальное

отсутствие $4 \cdot 10^{15}$ машинных чисел вблизи нуля означает, в частности, существование 2^n машинных чисел вида $1{,}M_k \cdot 2^{E_{\min}}$, разность любых двух из которых будет равна нулю $x_i - x_j = 0 \quad \forall i, j = 1, \ldots, 2^n$. Это, в свою очередь, будет приводить к «неожиданным» делениям на ноль, дополнительным сложностям в написании и отладке программ (проверка условий $x = y$ и $x - y \neq 0$ может давать разные результаты!) Согласно приблизительным оценкам [A6], в 70-х годах каждый компьютер попадал в «околонулевую яму» в среднем один раз в месяц — очевидно, указанная частота должна была расти пропорционально производительности ЭВМ.

Для решения данной проблемы в начале 1980-х были предложены (и в 1985 г. закреплены стандартом IEEE 754) *денормализованные* (subnormal) числа, запись мантиссы которых начинается с нуля целых: $\pm 0{,}M \cdot 2^{E_{\min}-1}$. Денормализованные числа кодируются в памяти ЭВМ двумя целыми числами: знаком $S = 0, 1$ и дробной частью мантиссы M. Смещённый двоичный порядок E_S денормализованных чисел равен нулю, что позволяет процессору отличать их от нормализованных чисел, для которых $E_S \geq 1$. При этом фактическое значение двоичного порядка, очевидно, совпадает с наименьшим значением $E_{\min} = -2^{w-1} + 2$ для нормализованных чисел:

$$x = (-1)^S \left(M/2^n\right) \cdot 2^{-(2^{w-1}-2)}. \tag{2}$$

Как видно из выражения (2), денормализованные числа образуют равномерную сетку с шагом $2^{-n} \cdot 2^{E_{\min}}$. Наименьшее по модулю денормализованное число равно нулю, наибольшее — $(1 - 2^{-n}) \cdot 2^{E_{\min}}$, где $E_{\min} = -2^{w-1} + 2$. Машинное число, следующее за наибольшим положительным денормализованным числом $(1-2^{-n}) \cdot 2^{E_{\min}}$, равно $1 \cdot 2^{E_{\min}}$ — наименьшему положительному нормализованному числу.

Стандарт IEEE 754 предусматривает целый ряд типов данных для представления чисел с плавающей точкой. Самый «младший» из двоичных типов, который поддерживается языком Си (тип `float`) и используется в численных расчётах, кодируется 32-битной последовательностью и называется обычно типом данных с плавающей точкой *одинарной точности*. Кроме того, стандартом предусмотрены 64-битные двоичные числа с плавающей точкой (числа *двойной точности*), которые также поддерживаются всеми современными компиляторами языка Си и математическими сопроцессорами (в отличие от 128-битных чисел с плавающей точкой, имеющих весьма ограниченную поддержку).

Заметим, что для многих современных процессоров скорость выполнения операций сложения, вычитания и умножения над числами

с двойной точностью может не уступать скорости обработки чисел с одинарной точностью[17]. В этой связи компилятор Си может генерировать машинный код, выполняющий вычисления с использованием двойной точности даже в случае, если результат будет затем помещён в переменную типа `float`. Кроме того, константы с плавающей точкой (напр., `3e10`) являются константами двойной точности (`double`) — соответственно, вычисление выражений, в которые они входят, также выполняется с двойной точностью. При необходимости определения констант типа `float` следует использовать суффикс `f` в их записи, например, `1.f`, `3e10f` и т. п.

2.3. За пределами разрядной сетки

Что будет, если $2^{E_{\max}}$ умножить на 2? Наблюдательный читатель, возможно, уже заметил, что $E_{\max} = 127$, что соответствует смещённому порядку $E_S = 254_{10} = 1111\,1110_2$. Значение $E_S = 255_{10} = 1111\,1111_2$ не используется для кодирования чисел — оно зарезервировано для сигнализации о выходе за пределы разрядной сетки. Если в результате вычислений с плавающей точкой получаются значения, бóльшие $(1 - 2^{-n}) \cdot 2^{E_{\max}}$, результатом вычислений является «бесконечность», кодируемая в соответствии с IEEE 754 как $E_S = 11\ldots1_2$, $M = 0$. Бесконечность, как и все машинные числа с плавающей точкой (включая ноль), бывает двух разных знаков. Результат сложения и умножения бесконечности на любое конечное число снова даёт бесконечность. При попытке напечатать «бесконечность» с помощью `printf` результатом может быть `1.#INF`, `inf` или что-нибудь подобное в зависимости от используемого компилятора.

Результатом операций $0 \cdot \infty$, ∞/∞, $\infty + (-\infty)$, $0/0$, $\sqrt{x}$ при $x < 0$ являются «не-числа» (not-a-numbers, NaNs), для обозначения которых используется комбинации $E_S = 11\ldots1_2$, $M \neq 0$. Не-числа обладают большой избыточностью кода — им соответствуют $2^n - 1$ комбинация битов, в то время как в соответствие со стандартом IEEE 754 существует всего два типа NaN — «тихие» (quiet) и «сигнальные» (signaling). Функция `printf` напечатает не-число как `-1.#IND00` или `nan` в зависимости от используемого компилятора.

[17] Однако на тех же процессорах быстродействие может отличаться при делении чисел с одинарной и двойной точностью, а также при вычислении квадратного корня и библиотечных функций. Кроме того, существенное отличие производительности наблюдается на графических процессорах (GPU), однако в последнее время хорошо заметна тенденция сокращения этого разрыва на новых видеокартах, ориентированных на высокопроизводительные вычисления.

В соответствии со стандартом, сравнение NaN с чем бы то ни было всегда даёт логическую ложь (`false`) — это позволяет выполнять проверку результатов вычислений, сравнивая результат с самим собой. Кроме того, в файле `<math.h>` описаны функции для определения типов чисел, названия которых говорят сами за себя: `isnan`, `isfinite`, `isinf`, `isnormal`, `fpclassify`. (Функции с аналогичными именами в пространстве имен `std::` определены в заголовочном файле `<cmath>`.)

Заметим также, что в отличие от чисел с плавающей точкой, язык Си не представляет средств для отслеживания переполнения целочисленных переменных. В случае возникновения переполнения устанавливается соответствующий флаг состояния процессора, который, однако, нельзя проверить средствами Си. Для его проверки можно использовать, например, команду условного перехода `jo` языка Ассемблер:

```
inline bool mult(int a, int b, int *pc)
{
    *pc = a*b;
    _asm {  //синтаксис Microsoft Visual C
        jo ON_FAIL
    }
    return true;        //в случае успешного выполнения
ON_FAIL: return false;  //в случае переполнения *pc
}
```

Альтернативным решением может быть использование значительно менее лаконичных (но при этом обладающих большей переносимостью) проверок средствами языка Си.

2.4. Точность вычислений

Ввиду конечности количества машинных чисел, вычисления с плавающей точкой на ЭВМ являются приближёнными. В этом несложно убедиться, выполнив в gnuplot команду `print 0.1+0.2-0.3`, либо произведя те же вычисления в программе на языке Си. Результат ($5{,}55 \cdot 10^{-17}$) близок к нулю, но тем не менее нулю не равен. Какова точность вычислений на компьютере?

Погрешность результата арифметических операций наподобие рассмотренного выше примера связана с округлением чисел. Так, точная запись каждого из слагаемых $0{,}1 = \frac{1}{10}$, $0{,}2 = \frac{1}{5}$ и $0{,}3 = \frac{3}{10}$ в двоичном виде содержит бесконечное количество разрядов, поскольку разложение на простые множители знаменателя каждой из указанных обыкновенных дробей содержит 5. При вводе в компьютер эти

числа округляются до $n = 52$ двоичных разрядов, в результате чего возникает относительная погрешность $2^{-(n+1)}$. Результат произвольной арифметической операции также может содержать относительную погрешность округления $2^{-(n+1)} = \varepsilon/2$. Число ε называют *машинным эпсилон*, или *ULP*[18], единицей в младшем разряде мантиссы. Другими словами, число ε — это минимальная степень двойки (или, в общем случае, основания системы счисления), при сложении которой с единицей результат будет отличен от единицы. Для чисел с одинарной точностью (`float`) $\varepsilon_1 = 2^{-23} \approx 1{,}2 \cdot 10^{-7}$, для чисел с двойной точностью (`double`) $\varepsilon_2 = 2^{-52} \approx 2{,}2 \cdot 10^{-16}$.

Подчеркнём, что ε характеризует именно относительную погрешность; абсолютная величина ошибки равна произведению $\frac{1}{2}\varepsilon \cdot 2^E$, где E — двоичный порядок вычисляемой величины. Указанная относительная ошибка округления ($\varepsilon/2$) относится к операциям умножения и деления, а также возникает при вводе числовых констант (наподобие 0,1, 0,2 и 0,3 в примере выше). Самой «опасной» операцией с точки зрения ошибки является вычитание чисел одного порядка величины и одного знака (или, что то же самое, сложение почти противоположных чисел).

Рассмотрим подробнее приводящий к ошибкам вычислений процесс округления — отбрасывание разряда за пределами точности вычисления. Для простоты рассмотрим примеры округления десятичных чисел до целых: $\pi \approx 3$, $e \approx 3$. В какую сторону следует округлять 2,5? Согласно общеизвестному арифметическому правилу, 5 следует округлять в бóльшую сторону, так что $2{,}5 \approx 3$. Какую ошибку мы совершим в среднем при округлении большого количества чисел в процессе вычислений? Полагая возникновение каждой из десяти цифр в отбрасываемом разряде равновероятным, несложно вычислить математическое ожидание ошибки, к которой приведёт такой способ в случае произвольного количества разрядов мантиссы:

$$\langle\delta\rangle = \frac{\varepsilon'}{10} \cdot \big(0 - 1 - 2 - 3 - 4 + 5 + 4 + 3 + 2 + 1\big) = \frac{\varepsilon'}{2}, \qquad (3)$$

где ε' — единица в первом разряде мантиссы за пределами точности вычислений (в двоичной системе $\varepsilon' = \varepsilon/2$).

В случае двоичной арифметики существование проблемы накопления ошибки округления, или *дрейфа*, ещё более очевидно ввиду наличия всего одной отличной от нуля цифры. При округлении бит 1_2 аналогичен цифре 5_{10}, равняясь половине от основания системы счисления. Если округлять 1_2 всегда в бóльшую сторону подобно тому, как мы

[18] Сокращение от англ. *unit in the last place* или *unit of least precision*.

привыкли поступать с 5_{10}, ошибка округления будет всегда неотрицательна.

Легко заметить, что ответ (3) для $\langle\delta\rangle$ будет одинаковым в любой системе счисления с чётным основанием, в том числе и в двоичной. Отличие от нуля математического ожидания ошибки округления $\langle\delta\rangle$ означает, в частности, что в сумме из N слагаемых либо в результате большого числа N арифметических операций погрешность будет возрастать приблизительно пропорционально N: $\delta_N \approx N\langle\delta\rangle = N \cdot \varepsilon'/2$.

Отличие от нуля $\langle\delta\rangle$ и постоянство знака ошибки хорошо заметно в финансовых расчётах. Поскольку доли копеек (центов, пенсов и т. п.) неизбежно округляются до целых, бухгалтерские отчёты имеют тенденцию расходиться, а организовав определённым образом (в строгом соответствии с арифметическим правилом!) процесс округления в банке, можно легко получить источник нетрудовых доходов. Для решения данной проблемы был придуман способ округления, обычно называемый «банковским» или «финансовым». В соответствии с ним, округление средней цифры (5_{10}, 1_2 и т. п.) производится в сторону чётного предыдущего разряда. Т. е. $2{,}5 \approx 2$, $3{,}5 \approx 4$ и т. д. Заметим, что количество чётных и нечётных разрядов одинаково в системах с чётным основанием, а только в таких системах и возникает проблема с округлением средней цифры. Следовательно, для «банковского» округления имеем $\langle\delta\rangle = 0$, что даёт рост погрешности $\mathcal{O}(\sqrt{N})$ вместо $\mathcal{O}(N)$.

Очевидно, можно решить проблему с $\langle\delta\rangle \neq 0$ и другими способами. Например, можно выбирать направление округления 5_{10} (1_2) случайным образом, или чередовать направления. Каждый из указанных способов не лишён недостатков: при случайном выборе направления расчёты теряют воспроизводимость (их результат становится случайной величиной); использование чётности операции подразумевает хранение номера операции и передачу его вместе с операндами и т. п. «Банковское» округление свободно от указанных недостатков, однако в случае наличия корреляций в отбрасываемом разряде оно также может приводить к отличию от нуля математического ожидания ошибки.

В соответствии со стандартом IEEE 754, по умолчанию используется «банковское» округление, однако предусмотрены ещё четыре способа: арифметическое округление, в сторону $+\infty$, в сторону $-\infty$ и в сторону нуля (отбрасывание дробной части). Используемый ЭВМ способ округления может быть изменён установкой соответствующего флага сопроцессора.

2.5. Заключение

Повседневный опыт подсказывает нам, что калькуляторы (и тем более компьютеры) выполняют арифметические операции чрезвычайно быстро и никогда не ошибаются. Многие склонны переносить этот вывод с бытовых вычислений на научные расчёты. Последние, однако, таят в себе немало особенностей и готовы предложить начинающему вычислителю множество сюрпризов и неожиданных проблем. Именно поэтому каждому исследователю, использующему в своей работе численные методы решения задач, полезно иметь представления об изложенных выше основах компьютерной арифметики.

Перечислим в порядке убывания очевидности несколько типичных примеров, демонстрирующих важность анализа ошибок и знания основных принципов компьютерной арифметики. Ещё ряд примеров приведён в конце главы в качестве упражнений для самостоятельного изучения.

1. Научные расчёты зачастую могут включать огромное количество операций, несоизмеримое с повседневным опытом: на решение одной задачи может требоваться несколько дней работы современного многопроцессорного компьютера, вычислительная мощность которого измеряется в терафлопсах[19]. При этом малые ошибки округления, допускаемые при выполнении каждой арифметической операции, могут накапливаться и приводить к полному искажению результата.

2. Решение некоторых задач (их принято называть *плохо обусловленными*) может приводить к существенному увеличению погрешности входных данных даже при относительно небольшом количестве операций. Примером может служить поиск точки пересечения почти параллельных прямых, вычисление корней многочлена высокой степени, нахождение собственных чисел несимметричной матрицы большого размера и др.

3. Наличие погрешности округления приводит к нарушению ассоциативности арифметических операций: например, $-1 + (1 + \frac{\varepsilon}{2}) \neq$ $\neq (-1 + 1) + \frac{\varepsilon}{2}$. Как следствие, при вычислении разности близких по модулю величин необходимо оптимизировать порядок выполнения операций для уменьшения погрешности результата. В некоторых случаях перед вычислением выражения необходимо выполнить тождественные преобразования либо разложение в ряд, в противном случае результат может не содержать ни одного верного знака (см. упражнение 7 на

[19] Терафлопс (от англ. аббревиатуры Flops, floating-point operations per second) — единица измерения быстродействия ЭВМ, равная 10^{12} операций с плавающей точкой в секунду.

с. 68). Например, для вычисления разности $\sqrt{1+\delta^2}-\sqrt{1-\delta^2}$ при $\delta \ll 1$ выгодно переписать её в виде $2\delta^2/(\sqrt{1+\delta^2}+\sqrt{1-\delta^2})$, что позволит избежать вычитания близких величин. Аналогичный приём необходим для предотвращения потери точности при вычислении близкого к нулю корня квадратного уравнения [A7, с. 11]: при $b^2 \gg ac$ числитель стандартной формулы $(-b \pm \sqrt{b^2-ac})/a$ содержит разность близких величин. Порядок сомножителей в произведении, хотя и не влияет на *точность* ответа, также может быть очень важен. Неправильный выбор порядка сомножителей способен привести к переполнению разрядной сетки или, наоборот, появлению в расчётах машинного нуля.

4. Зачастую совершенно неожиданное для начинающих физиков-вычислителей поведение программ связано со сравнением чисел с плавающей точкой и ветвлением алгоритмов. Так, если имеются две равных величины $a = b$, из-за наличия случайных погрешностей при их вычислении проверка условия `if (a < b)` может давать как логическую истину, так и ложь. Сравнение двух чисел с плавающей точкой на строгое равенство `if (a == b)` в большинстве случаев полностью лишено смысла — вместо этого необходимо сравнивать числа с плавающей точкой приближённо, с заданной абсолютной и относительной точностью, с которой известны данные значения. Часто встречающаяся ошибка связана с разбиением отрезка (например, области интегрирования) и перебором узлов сетки. Кажущийся на первый взгляд очевидным программный код для перебора узлов равномерной сетки от a до b

```
for(double x = a;  x <= b;  x += (b-a)/N) { ... }
```

очень часто приводит к неверному результату, поскольку в качестве условия выполнения тела цикла используется сравнение двух чисел с плавающей точкой x и b. Чтобы избежать подобной ошибки, рекомендуем вести перебор узлов сетки в цикле с целочисленным индексом:

```
double x,  h = (b-a)/N;
for(int i = 0;  i <= N;  ++i, x = a + i*h) { ... }
```

В заключение ещё раз подчеркнём, что оценка погрешностей вычислений является нетривиальной задачей, дающей зачастую весьма неожиданные результаты. Ошибки округления являются лишь одним из источников погрешностей получаемого численного решения. Кроме того, свой вклад неизбежно дают приближения, допущенные при формулировке математической модели физической задачи. Сюда может быть отнесено также задание приближённых численных параметров моделируемой физической системы и погрешности численного метода,

обусловленные заменой непрерывных величин дискретными (ошибки дискретизации), сведением математической задачи к конечномерному приближению, остановом сходящегося к решению бесконечного итерационного процесса после конечного числа шагов и т. п. При решении физических задач следует стремиться к соблюдению баланса между погрешностями различных типов. Так, например, если математическая модель описывает физическую систему с точностью порядка 1 %, вряд ли имеет смысл бороться за восемь значащих цифр в ответе.

Наконец, для удобства представим в виде таблицы основные характеристики типов данных с одинарной, двойной и четверной точностью (`float`, `double` и `long double`[20] в языке Си, или binary 32, 64 и 128 в соответствии со стандартом IEEE 754). Для использования в программном коде перечисленных в табл. 1 и других аналогичных им параметров, предусмотрены константы `FLT_EPSILON`, `FLT_MAX`, `FLT_MIN`, `DBL_EPSILON` и т. п. (см. файл `<float.h>`).

Таблица 1

	binary 32 (`float`)	binary 64 (`double`)	binary 128
Размер в битах	32	64	128
Кол-во бит мантиссы, n	23	52	112
Кол-во бит показателя, w	8	11	15
$\varepsilon = 2^{-n}$ (ULP)	$1{,}19 \cdot 10^{-7}$	$2{,}22 \cdot 10^{-16}$	$1{,}93 \cdot 10^{-34}$
max показатель $E_{\max}$	127	1023	16383
max число $2^{E_{\max}}(2-2\varepsilon)$	$3{,}40 \cdot 10^{38}$	$1{,}80 \cdot 10^{308}$	$1{,}19 \cdot 10^{4932}$
min норм. число $2^{1-E_{\max}}$	$1{,}18 \cdot 10^{-38}$	$2{,}23 \cdot 10^{-308}$	$3{,}36 \cdot 10^{-4932}$

В качестве дополнительной литературы к данному разделу можно рекомендовать вторую главу замечательной монографии [A8]. Краткое рассмотрение вопросов точности вычислений можно найти во введении учебного пособия [3]. Классическое рассмотрение ряда алгоритмов и нетривиальных вопросов, связанных с вычислениями на ЭВМ, выполнено в [A9]. Для владеющих английским языком небезынтересно будет знакомство с [A1, гл. 3 и 14].

Упражнения

1. Напишите на языке Си программу для нахождения значения машинного эпсилон, последовательно прибавляя к единице 2^{-1}, 2^{-2},

[20] Разрядность `long double` может варьироваться на разных платформах.

2^{-3} и т. д. Определите число разрядов в мантиссе, максимальный и минимальный двоичный порядок чисел при вычислениях с одинарной и двойной точностью.

2. Сравните друг с другом четыре машинных числа: 1, $1+\frac{\varepsilon}{2}$, $1+\varepsilon$ и $1+\varepsilon+\frac{\varepsilon}{2}$, объясните результат. То же для чисел $1+\varepsilon+\frac{\varepsilon}{2}$ и $1+\frac{\varepsilon}{2}+\varepsilon$.

3. Какое наименьшее число с плавающей точкой вида 2^{-n} можно прибавить к числу $x = 1000$, чтобы сумма была больше x?

4. Напишите программу, умножающую `float x = 1` на 2 до тех пор, пока результат не выйдет за пределы разрядной сетки. Сколько шагов цикла потребовалось для этого? Многократно деля `x` на 2, найдите общее количество степеней двойки, представимых во `float`. Является ли полученное число «круглым» в двоичной системе счисления? Почему?

5. Сколько всего битов используется для записи чисел типа `float`? Выведите на экран значение `sizeof(float)`. Вычтите знаковый бит и биты для записи мантиссы (см. задачу 1). Сравните остаток с двоичным логарифмом от количества степеней двойки, полученных в упражнении 4, объясните результат.

6. Напишите на языке Си функцию для сравнения друг с другом двух чисел с плавающей точкой на приближённое равенство с точностью до машинного эпсилон.

7. Напишите на языке Си программу, содержащую следующие две строки кода, объясните результат её работы:

```
float x = 0.1;
if(x == 0.1) puts("equal"); else puts("not equal");
```

8. Аналогично для следующих строк кода:

```
double x = exp(777),  y = x - x;
puts((y == 0) ? "equal" : "not equal");
puts((y == y) ? "equal" : "not equal");
```

9. Скомпилируйте и выполните программу на языке Си, содержащую следующий фрагмент кода, объясните результат:

```
double L = 1, dx = 0.1;
int i = 0;
while(L > 0) ++i, L -= dx;
printf("A section of length L=1 was divided into %d "
       "sub-sections of length %f\n", i, dx);
```

10. Что напечатает программа, содержащая следующие строки:

```
float x0 = FLT_EPSILON, x1 = x0/2, x2 = x1*2;
printf("x2-x0 = %e\n", x2-x0);
```

11. Что будет, если в предыдущем примере заменить `FLT_EPSILON` на `FLT_MIN`? В чем смысл указанных констант?

12. Пусть $f(a) := u(100, a)$, где $u(x, a)$ — решение задачи Коши для обыкновенного дифференциального уравнения $\frac{d}{dx}u = u - x$, $0 \le x \le 100$, $u(0) = a$. Используя общее решение ОДУ $u(x) = 1 + x + c \cdot e^x$, вычислите значения $f(1)$ и $f(\mathrm{tg}(\frac{\pi}{4}))$. Объясните полученные результаты.

13. Определите в gnuplot функцию $f(x, y) = \sin(x + 2\pi \cdot 10^y)$ и постройте график $f(x, y = \mathrm{const})$ при фиксированном целочисленном значении y, выполнив приведённые ниже две строки кода. Объясните результат:

```
f(x,y)=sin(x+2*pi*10**y)  #a**b означает "a в степени b"
plot [0:2*pi] f(x,0), f(x,3), f(x,5), f(x,15)
```

14. Проверьте, что интерполяционный полином второй степени $y(x) = -\frac{1}{2}x^2\delta + (1 + \frac{3}{2}\delta)\,x - \delta$ проходит через точки $(1, 1)$, $(2, 2)$ и $(3, 3-\delta)$. Решая квадратное уравнение $y(x) = 0$ на калькуляторе или ЭВМ, вычислите ближайший к нулю корень при $\delta = 10^{-4}$, 10^{-6}, $1.3 \cdot 10^{-8}$. Сколько значащих цифр содержит ответ? Как следует переписать выражение для повышения точности?

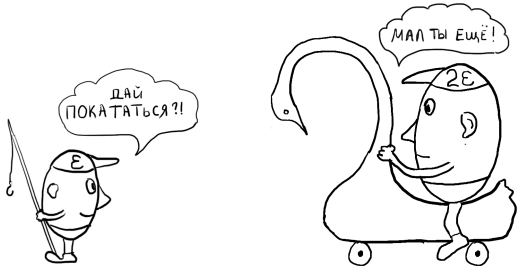
15. Опишите алгоритм, позволяющий определить систему счисления, используемую калькулятором для выполнения вычислений (двоичная или десятичная). Известно, что калькулятор работает с числами с плавающей точкой; ввод данных и отображение результатов производится в десятичной системе счисления.

16. Вычислите значение $\mathrm{B}(100, 200)$, где $\mathrm{B}(a, b) = \Gamma(a)\Gamma(b)/\Gamma(a + b)$ — бета-функция Эйлера, $\Gamma(n + 1) = n!$ — гамма-функция Эйлера.

17. Пусть $x \le y$ — два машинных числа с плавающей точкой. Всегда ли проверка условий $x \le \frac{1}{2}(x+y)$ и $\frac{1}{2}(x+y) \le y$ на ЭВМ будет давать логическую истину? (H. Björk [A9]).

18. Продемонстрируйте нарушение неравенства Коши для чисел с плавающей точкой — найдите числа x и y, для которых при вычислении на ЭВМ справедливо $(x+y)^2 > 2(x^2+y^2)$ [A9].

19. Каких чисел с плавающей точкой больше — положительных или отрицательных? Бóльших единицы или меньших единицы?



3. Решение конечных уравнений

Решение большинства физических задач приводит к возникновению *конечных* уравнений, т. е. уравнений, которые могут быть представлены в форме $f(x) = 0$. Если $f(x)$ является полиномом, такие уравнения называют *алгебраическими*, в противном случае — *трансцендентными*. Зачастую точное решение конечных уравнений не может быть выражено через элементарные функции. Примером может служить хрестоматийная задача из квантовой механики о поиске уровня энергии E основного состояния в прямоугольной яме конечной глубины:

$$-\frac{\hbar^2}{2m}\psi'' + (U(x) - E)\psi = 0, \quad U(x) = \begin{cases} -U_0, & |x| \le a, \\ 0, & |x| > a. \end{cases}$$

Волновая функция основного состояния чётная и имеет вид

$$\psi(x) = \begin{cases} A\cos kx, & |x| < a \\ Be^{-qx}, & |x| > a \end{cases}, \qquad k^2 = \frac{2m}{\hbar^2}(U_0 + E), \quad q^2 = -\frac{2m}{\hbar^2}E.$$

Условие непрерывности решения $\psi(x)$ и его первой производной в точке $x = a$ приводит к трансцендентному уравнению, точное решение которого не может быть представлено в элементарных функциях:

$$\operatorname{ctg}\sqrt{\frac{2ma^2U_0}{\hbar^2}(1-\xi)} = \sqrt{\frac{1}{\xi}-1}, \quad \xi = \frac{-E}{U_0}. \tag{4}$$

Неразрешимые в элементарных функциях трансцендентные уравнения часто возникают при решении спектральных задач для дифференциальных операторов второго порядка в частных производных, при решении задачи Коши для обыкновенных дифференциальных уравнений и в ряде других случаев. В данной главе будут рассмотрены несколько наиболее употребительных методов численного решения таких уравнений.

3.1. Метод деления отрезка пополам

Пожалуй, самым простым в реализации методом поиска корней функции одной вещественной переменной является *метод дихотомии*, или деления отрезка пополам. Пусть на отрезке $[a, b]$ задана непрерывная функция $f : [a, b] \to \mathcal{R}$, причём $f(a) \cdot f(b) \leq 0$. Из курса анализа известно, что непрерывная функция принимает на отрезке все промежуточные значения и, следовательно, существует точка $x_* \in [a, b] : \quad f(x_*) = 0$. Наша задача состоит в нахождении такой точки (корня $f(x)$).

Для решения поставленной задачи вычислим значение функции f в средней точке отрезка, $f(\frac{1}{2}(a+b))$. Если $f(a) \cdot f(\frac{1}{2}(a+b)) \leq 0$, положим $a_1 = a$, $b_1 = \frac{1}{2}(a+b)$, в противном случае возьмем $a_1 = \frac{1}{2}(a+b)$, $b_1 = b$. Шаг дихотомии выполнен. Мы свели задачу о поиске корня на отрезке $[a, b]$ к аналогичной задаче для отрезка $[a_1, b_1]$, ширина которого в два раза меньше. Повторяя процесс многократно, получим последовательность отрезков $[a_n, b_n]$, причём $b_n - a_n \to 0$ при $n \to \infty$. Предлагаем читателю самостоятельно доказать сходимость процесса к корню x_*, используя свойства непрерывности $f(x)$ и принцип математической индукции, а здесь ограничимся лишь несколькими замечаниями, касающимися использования данного метода на ЭВМ.

На каждом шаге метода дихотомии можно было бы выполнять проверку равенства нулю значения функции в средней точке отрезка $f(\frac{1}{2}(a+b))$ с остановом итерационного процесса при достижении корня. Ввиду конечности множества машинных чисел вероятность «точного» попадания в ноль функции $f(x)$ хотя и не равна нулю, но тем не менее достаточно мала на большинстве шагов дихотомии. Как следствие,

введение дополнительной проверки не приведет к повышению эффективности программы, однако может сделать программный код более читаемым и понятным для некоторых пользователей.

В эффективных реализациях алгоритма функцию f следует вычислять один раз на каждом шаге, сохраняя найденные ранее значения $f(a)$ и $f(b)$ в памяти ЭВМ. «Лишние» вычисления значений функции f могут привести к существенному снижению скорости работы программы в случаях, когда вычисление функции f требует большого количества операций.

При проверке знака функции в середине отрезка, $f(\frac{1}{2}(a+b))$, следует иметь в виду возможность выхода за пределы разрядной сетки, а также принципиальную возможность «точного» попадания в искомый корень в процессе итераций. Условие выхода из цикла итераций должно быть основано на сравнении ширины отрезка и требуемой точности (не превосходящей точность, обеспечиваемую используемым типом данных с плавающей точкой), в противном случае может произойти зацикливание программы.

Максимальная абсолютная погрешность определения корня x_* равна $\frac{1}{2}(b-a)$ и на каждой итерации уменьшается вдвое. Следовательно, если искомое значение x_* известно по порядку величины, каждый шаг дихотомии даёт один дополнительный бит мантиссы. Это позволяет оценить необходимое число шагов для достижения требуемой точности:

$$N \approx \frac{\log(\delta_0/\delta)}{\log 2}, \tag{5}$$

где δ_0 и δ — точность начального приближения и требуемая точность нахождения корня соответственно.

Метод деления отрезка пополам обладает рядом ограничений: он позволяет находить только простые корни (а также корни, кратность которых нечётна) и не обобщается на функции нескольких переменных. Ниже будет дан обзор других методов, которые при определённых условиях могут обладать более высокой скоростью сходимости, что может быть существенно в случае, когда вычисление каждого значения $f(x)$ занимает достаточно много времени, либо когда требуется решение большого количества уравнений (например, на каждом шаге численного интегрирования системы дифференциальных уравнений). Вместе с тем дихотомия обладает важным преимуществом: гарантированной сходимостью к корню непрерывной функции $f(x):\ f(a)\cdot f(b)<0$.

3.2. Метод простых итераций

Перепишем уравнение $f(x) = 0$ в виде $\varphi(x) = x$, где $\varphi(x) := f(x)+x$. Для простоты будем считать вначале, что функция $\varphi(x)$ гладкая и удовлетворяет условию $|\varphi'(x)| \le q$, где $q < 1$ — некоторая постоянная. Пусть нам также известно некоторое начальное приближение x_0 к корню x_*. Организуем итерационный процесс вида

$$x_{n+1} = \varphi(x_n), \quad n = 0, 1, 2, \ldots \tag{6}$$

Данный процесс изображён на рис. 3 (a); стрелками обозначены вычисления функции $\varphi(x)$. Покажем, что предел последовательности $\{x_n\}$ существует и равен искомому корню $x_* : \varphi(x_*) = x_*$. Чтобы убедиться в этом, заметим, что функция $\varphi(x)$ реализует *сжимающее отображение*, уменьшая длину произвольного отрезка $[a, b] \to [\varphi(a), \varphi(b)]$ минимум в $1/q$ раз в силу ограниченности производной $|\varphi'| \le q$. При этом корень x_* по определению отображается функцией $\varphi(x)$ сам в себя. Следовательно, выбирая $a = x_0$ и $b = x_*$, имеем $|x_n - x_*| \to 0$ при $n \to \infty$, т. е. $x_n \to x_*$, ч. т. д. ■

Оценим количество итераций, необходимых для получения приближённого значения корня с требуемой точностью δ. Будем полагать функцию $\varphi(x)$ гладкой, а начальное приближение x_0 достаточно точным, так что $\varphi(x_0) \approx \varphi(x_*)+\varphi'(x_*)\,(x_0-x_*)$. Учитывая, что $\varphi(x_*) = x_*$, получаем $(x_1 - x_*) \approx (x_0 - x_*) \cdot \varphi'(x_*)$, т. е. абсолютная погрешность определения искомого корня умножается на каждой итерации на величину $|\varphi'| \le q < 1$, откуда получаем оценку для числа итераций:

$$N \approx \frac{\log(\delta_0/\delta)}{-\log|\varphi'|}. \tag{7}$$

Сравнение (5) и (7) показывает, что метод простых итераций сходится быстрее метода деления отрезка пополам при условии $|\varphi'(x_*)| < \frac{1}{2}$. Особенно быстрой будет сходимость при $\varphi'(x_*) = 0$. Очевидно, что проведённое выше рассмотрение и оценка количества итераций (7) в последнем случае будут неприменимы: при малых $|\varphi'(x_*)|$ в разложении $\varphi(x_0) \approx \varphi(x_*) + \varphi'(x_*)\,(x_0 - x_*)$ в ряд Тейлора следует удерживать члены более высоких порядков.

Заметим, что сформулированное вначале ограничение $|\varphi'| \le q < 1$ является достаточным, но не необходимым условием сходимости итерационного процесса. Для сходимости итераций достаточно, чтобы отображение φ уменьшало длину произвольного отрезка, один из концов которого совпадает с искомым корнем x_*, при этом производная φ'

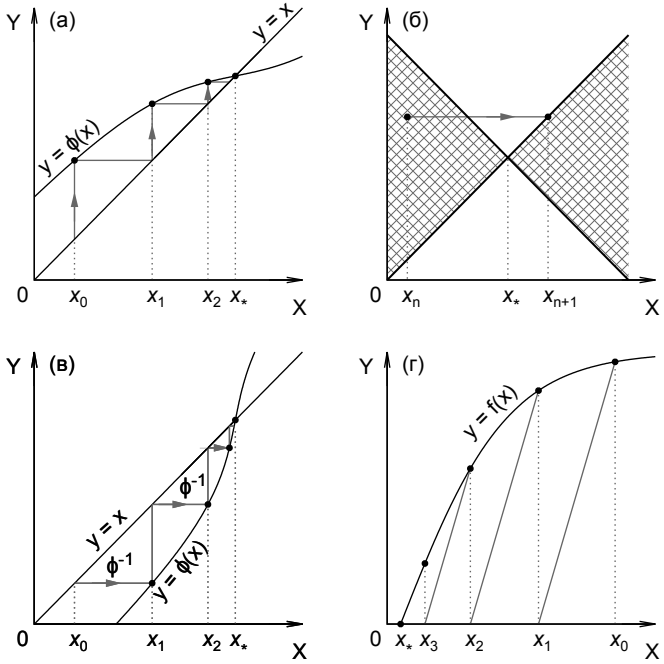


Рис. 3. (а) Метод простых итераций, (б) секторы сходимости, (в) итерации для обратной функции, (г) использование корректирующего множителя

может принимать сколь угодно большие значения или даже вовсе не существовать. Итерационный процесс будет сходиться при любом начальном приближении x_0, если график функции $\varphi(x)$ лежит между прямыми $y_1 = x_* + q(x - x_*)$ и $y_2 = x_* - q(x - x_*)$, где $0 < q < 1$ — некоторая постоянная (рис. 3 (*б*)). На рисунке видно, что итерация $x_{n+1} = \varphi(x_n)$ из любой точки $(x_n, \varphi(x_n))$ внутри заштрихованных секторов улучшит приближение к корню: $|x_{n+1} - x_*| < |x_n - x_*|$.

Очевидно, многие функции не удовлетворяют даже более слабому условию, сформулированному в предыдущем абзаце. Покажем две модификации итерационного процесса, позволяющие расширить область его применения.

Первая возможность связана с переходом от функции φ к обратной функции φ^{-1}. В случае, если $|\varphi'(x)| \geq q > 1$, можно перейти к обратным функциям в уравнении $\varphi(x) = x$, в результате получим ите-

рационный процесс $x_{n+1} = \varphi^{-1}(x_n)$, который будет сходиться в силу соотношения $(\varphi^{-1})' = 1/\varphi'$. Геометрический смысл итераций для обратной функции показан на рис. 3 (*в*), стрелками обозначены вычисления функции $\varphi^{-1}(x)$. Производная $\varphi' > 1$, поэтому итерационный процесс $x_{n+1} = \varphi(x_n)$ будет расходящимся. Для сходимости необходимо, по сути, двигаться в обратную сторону (ср. направление стрелок на рис. 3 (*а*) и (*в*)), что и достигается переходом к вычислению обратной функции $\varphi^{-1}(x_n)$ вместо $\varphi(x_n)$. Итерационный процесс для обратных функций целесообразно использовать в случае, когда обратная функция φ^{-1} может быть относительно легко вычислена, а $|\varphi'(x_*)| \gg 1$.

Другая модификация метода итераций может быть получена введением корректирующего множителя. Вместо уравнения $x = x + f(x) \equiv \varphi(x)$ будем решать уравнение $x = x - \lambda f(x)$, множество решений которого, очевидно, совпадает с множеством корней уравнения $f(x) = 0$ при условии $\lambda \neq 0$. Пусть функция $f(x)$ непрерывно дифференцируема, так что $|f'(x)| < \infty$. Производная $\tilde{\varphi}'(x) = 1 - \lambda f'(x)$ может быть сделана сколь угодно малой в точке $x = x_*$ соответствующим выбором λ, что обеспечит сходимость итерационного процесса

$$x_{n+1} = x_n - \lambda f(x_n) \tag{8}$$

для любой гладкой функции $f(x)$. Знак λ нужно выбирать равным знаку производной, а значение — по возможности ближе к обратному значению производной $1/f'(x_*)$. Геометрический смысл итерационного процесса (8) показан на рис. 3 (*г*).

Выход из цикла итераций можно осуществлять, проверяя условие малости изменения x на последней итерации: $|x_n - x_{n-1}| < \delta$, где δ — требуемая точность. Однако данная оценка может быть слишком грубой, особенно в случае $|\varphi'(x_*)| \approx 1$. Для получения более точного критерия завершения итерационного процесса рассмотрим последовательность $\{x_k\}$, сходящуюся к корню x_*, полагая x_0 достаточно близким к x_*. Обозначая $R_n := x_* - x_n$, из $x_{n+1} = x_n - \lambda f(x_n)$ получаем

$$R_{n+1} \approx \big(1 - \lambda f'(x_*)\big)\, R_n \equiv q R_n, \tag{9}$$

т. е. итерации сходятся приблизительно как сумма геометрической прогрессии со знаменателем $q = (x_n - x_{n-1})/(x_{n-1} - x_{n-2})$. Итерационный процесс может быть остановлен, когда погрешность R_n текущего приближения к корню не превосходит требуемой точности вычислений δ. Поскольку $x_n - x_{n-1} = R_{n-1} - R_n \approx R_n/q - R_n$, приходим к условию

$$|R_n| \approx \left|\frac{x_n - x_{n-1}}{1/q - 1}\right| = \frac{(x_n - x_{n-1})^2}{|2x_{n-1} - x_n - x_{n-2}|} < \delta. \tag{10}$$

Оценка для R_n в левой части неравенства (10) может быть использована для повышения точности расчётов (сокращения числа итераций, необходимых для получения заданной точности).

3.3. Метод Ньютона

Геометрический смысл рассмотренного выше метода итераций состоит в приближении к искомому корню x_* по прямым с постоянным наклоном (рис. 3 (*г*)). Скорость сходимости можно существенно повысить, положив $\lambda(x) = 1/f'(x)$ с тем, чтобы наклон был равен производной функции $f(x)$ (рис. 4 (*а*)). При этом мы получим итерационный процесс

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots, \tag{11}$$

известный как *метод Ньютона*, или *Ньютона — Рафсона*. В выражении (11) полезно увидеть также *точное* решение *линеаризованной* задачи $f(x) = 0$ — это позволит нам впоследствии обобщить данный метод на матричные и операторные уравнения.

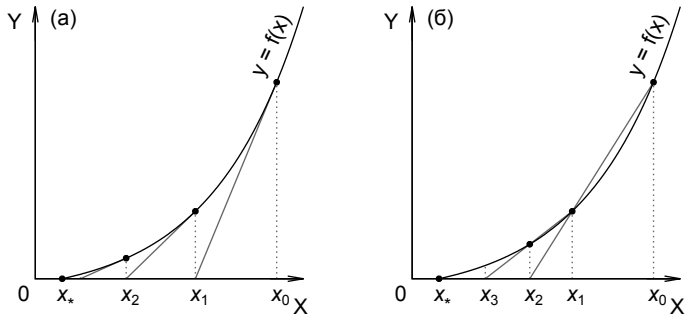


Рис. 4. (а) метод Ньютона, (б) метод секущих

Исследуем скорость сходимости процесса (11) к корню x_*. Для этого предположим, как и раньше, что нам задано достаточно хорошее приближение x_n, так что $f(x_n) \approx f'(x_*)\,(x_n - x_*) + \frac{1}{2} f''(x_*)\,(x_n - x_*)^2$, а $f'(x_n) \approx f'(x_*) + f''(x_*)\,(x_n - x_*)$. Вычитая x_* из обеих частей (11), имеем:

$$R_{n+1} \approx R_n + \frac{f'(x_*)\,(-R_n) + \frac{1}{2} f''(x_*)\,(-R_n)^2}{f'(x_*) - R_n f''(x_*)} \approx -\frac{1}{2}\frac{f''(x_*)}{f'(x_*)} R_n^2. \tag{12}$$

Сравнивая (12) с (9), легко заметить, что метод Ньютона обладает более высоким *порядком сходимости* по сравнению с методом итераций. Так, если в методе итераций погрешность убывала приблизительно в геометрической прогрессии ($R_{n+1} \approx qR_n^1$), то в методе Ньютона погрешность на каждом следующем шаге оказывается квадратично мала по сравнению с погрешностью на предыдущем шаге, $R_{n+1} = \alpha R_n^2$.

Заметим, что выражение (12) допускает возможность расходящегося процесса $|R_{n+1}| > |R_n|$ при достаточно больших $|R_n|$, тогда как при достаточно малых $|R_n|$ процесс монотонно сходится. Другими словами, в методе Ньютона может существовать *область притяжения к корню*: в случае, если начальное приближение x_0 лежит в области $a < x_0 < b$, так что $x_* \in (a, b)$, процесс (11) сходится к корню x_*, в противном случае итерации расходятся (либо сходятся к другому корню, отличному от x_* и лежащему вне отрезка $[a, b]$).

3.4. Метод секущих

Рассмотренный в п. 3.3 метод Ньютона обладает вторым порядком сходимости, однако для его реализации необходимо вычисление на каждом шаге значения функции $f(x_n)$ и её первой производной $f'(x_n)$. Зачастую, однако, производная функции $f(x)$ неизвестна (например, $f(x)$ измеряется в эксперименте или является результатом численного решения некоторой сложной задачи). Кроме того, даже если имеется выражение, позволяющее вычислить $f'(x_n)$, такое вычисление требует дополнительного времени на каждом шаге, что может сделать более выгодным использование других методов.

Например, вместо $f'(x)$ при вычислении корректирующего множителя $\lambda = 1/f'(x)$ в (8) можно использовать оценки значения производной, полученные с использованием асимптотических разложений $f(x)$ или вычисленные с помощью разделённой разности (см. п. 5.2) по результатам двух последних вычислений $f(x)$:

$$x_{n+1} = x_n - f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2, \ldots \tag{13}$$

Итерационный процесс (13) является *двухшаговым*: для нахождения следующего приближения x_{n+1} необходимо знать два предыдущих.

Исследуем скорость сходимости процесса (13). Полагая x_n и x_{n-1} достаточно близкими к корню x_* и заменяя $f(x) \approx f'(x_*)\,(x - x_*) + \frac{1}{2} f''(x_*)(x - x_*)^2$, имеем:

$$R_{n+1} \approx R_n - R_n\,\frac{1 - aR_n}{1 - a \cdot (R_n + R_{n-1})}, \quad R_n \equiv x_* - x_n,\ a \equiv \frac{f''(x_*)}{2f'(x_*)}.$$

Удерживая члены не выше второго порядка по R_n и R_{n-1}, получаем:

$$R_{n+1} \approx -aR_nR_{n-1}. \tag{14}$$

Аналогично (9) и (12), попробуем найти степенной закон убывания погрешности на каждой итерации. Подставляя $|R_{n+1}| = a^\alpha |R_n|^\beta$ в (14), получаем два условия на показатели α и β: $\alpha\beta = 1$, $\beta^2 - \beta - 1 = 0$. Для сходимости итерационного процесса необходимо $\beta > 0$, откуда $\beta = \frac{1}{2}(1 + \sqrt{5}) \approx 1{,}618$.

Порядок процесса получился дробным — метод секущих (13) сходится быстрее (по числу шагов) метода простых итераций, но медленнее метода Ньютона. Заметим, однако, что в методе Ньютона на каждом шаге необходимо вычислять значение функции $f(x_n)$ и её первой производной $f'(x_n)$. Полагая, что для вычисления f и f' требуется приблизительно одинаковое количество арифметических операций, получим, что на каждые два вычисления функции приходится один шаг метода Ньютона, дающий (приблизительно) удвоение числа значащих цифр после запятой для корня, $|R_{n+1}| \propto |R_n|^2$. Для сравнения, двукратное вычисление значения функции f позволяет сделать два шага методом секущих, что даёт улучшение точности $|R_{n+2}| \propto |R_n|^{\beta^2}$. Поскольку $\beta^2 \approx 2{,}618$, метод секущих может сходиться быстрее метода Ньютона, если сравнивать не по числу итераций, но по затратам машинного времени, необходимого для нахождения корня с заданной точностью.

Когда следует прекращать итерации (13)? Выражение (13) содержит конечную разность, погрешность вычисления которой может сильно возрастать при близких x_n и x_{n-1}, особенно если вычисление функции $f(x)$ производится с достаточно большой погрешностью[21] и/или в случае кратного корня функции $f(x)$. Как следствие, итерации (13) могут вначале сходиться к искомому корню, однако в какой-то момент может возникнуть «разболтка» процесса. Чтобы избежать этого, используется *приём Гарвика*: итерации (13) выполняют, пока изменение приближения к корню на одном шаге $|x_n - x_{n-1}|$ не станет меньше некоторого порога δ, обеспечивая тем самым попадание в достаточно малую окрестность корня, где сходимость итерационного процесса была бы монотонной в соответствии с (14). Далее, итерации продолжают до тех пор, пока величина $|x_{n+1} - x_n|$ монотонно убывает с ростом n. Начало возрастания свидетельствует о «раскачке» итерационного процесса и необходимости его прекращения.

[21] Что нередко бывает при использовании недостаточно точного метода вычисления спец.функции или интеграла с помощью квадратурной формулы.

3.5. Многомерное обобщение

Метод Ньютона очевидным образом может быть обобщён на многомерный случай. Пусть задана система уравнений $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, где $\mathbf{f} = \{f_1, \ldots, f_n\}$, $\mathbf{x} = \{x_1, \ldots, x_n\}$. Выбрав некоторое начальное приближение $\mathbf{x}^{(0)} = \{x_1^{(0)}, \ldots, x_n^{(0)}\}$ к корню $\mathbf{x}^{(*)} = \{x_1^{(*)}, \ldots, x_n^{(*)}\}$, построим последовательность $\mathbf{x}^{(n)}$, $n = 1, 2, \ldots$, сходящуюся к искомому корню. Для этого, как и в одномерном случае, линеаризуем уравнения $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Используя тензорные обозначения и предполагая суммирование по повторяющимся индексам, имеем:

$$0 = f_i\big(\mathbf{x}^{(n)} + \delta\mathbf{x}^{(n)}\big) \approx f_i\big(\mathbf{x}^{(n)}\big) + J_{ij}^{(n)}\delta x_j^{(n)}, \quad \text{где } J_{ij}^{(n)} = \left.\frac{\partial f_i}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}^{(n)}},$$

откуда получаем для $\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \delta\mathbf{x}^{(n)}$:

$$x_i^{(n+1)} = x_i^{(n)} - \left[\left(J^{(n)}\right)^{-1}\right]_{ij} f_j\big(\mathbf{x}^{(n)}\big). \tag{15}$$

3.6. Поиск комплексных корней

В некоторых задачах может возникнуть необходимость поиска нулей функции на комплексной плоскости. Пусть $f(z)$ — аналитическая функция в области $\mathcal{D}$, и $|f(z)| < \infty \quad \forall z \in \mathcal{D}$. Разложим $f(z)$ и её производную $f'(z)$ в ряд в точке $z_0 \in \mathcal{D}$:

$$f(z) = \sum_{k=0}^{\infty} \frac{c_k}{k!}(z - z_0)^k, \quad f'(x) = \sum_{k=1}^{\infty} \frac{c_k}{(k-1)!}(z - z_0)^{k-1}, \quad c_k = f^{(k)}(z_0).$$

Если z_0 — нуль функции f, то $c_0 = 0$ и $f'/f = (z - z_0)^{-1} + \mathcal{O}\left(|z - z_0|^0\right)$. Если z_0 — нуль кратности n, то

$$f(z) = \sum_{k=n}^{\infty} \frac{c_k}{k!}(z - z_0)^k, \quad \frac{f'(z)}{f(z)} = \frac{n}{z - z_0} + \mathcal{O}\left(|z - z_0|^0\right).$$

Используя интегральную формулу Коши, имеем

$$\oint_{\gamma} \frac{f'(z)}{f(z)}\, dz = 2\pi i \sum_j n_j = 2\pi i N. \tag{16}$$

С точностью до множителя $2\pi i$ получили количество нулей функции $f(z)$ внутри контура γ с учетом их кратности.

Стратегия поиска комплексных корней $f(z)$ с использованием (16) может быть следующей. Выбираем контур интегрирования γ так, чтобы внутрь него попадало 1-2 нуля функции $f(z)$. Допустим, для примера, $\sum n_j = 2$. Далее вычисляем интегралы вида

$$I_1 = \frac{1}{2\pi i} \oint\limits_{\gamma} \frac{f'(z)}{f(z)} z\, dz = \sum_j z_j n_j = z_1 + z_2,$$

$$I_2 = \frac{1}{2\pi i} \oint\limits_{\gamma} \frac{f'(z)}{f(z)} z^2\, dz = \sum_j z_j^2 n_j = z_1^2 + z_2^2.$$

Решение (z_1, z_2) полученной системы уравнений даёт нам решение исходной задачи о нахождении комплексных корней $f(z)$.

Заметим, что задача нахождения комплексных корней может быть решена с использованием рассмотренных выше методов Ньютона, секущих или метода итераций. Эти методы могут быть гораздо более эффективны в случае, если известно достаточно хорошее начальное приближение к искомому корню.

3.7. Сравнение методов

Какой из рассмотренных выше методов лучше? Однозначного ответа на этот вопрос нет — оптимальный метод зависит от условий поставленной задачи. Метод дихотомии обеспечивает сравнительно невысокую, но при этом *гарантированную* скорость сходимости для корней нечётной кратности, тогда как сходимость других рассмотренных выше методов может быть условной. В этой связи метод дихотомии может быть использован для получения грубого приближения к корню с последующим его уточнением каким-либо другим методом, имеющим более высокую скорость сходимости. Метод Ньютона эффективен, подходит в том числе для корней с чётной кратностью, но требует вычисления производной. Метод секущих также очень эффективен, однако может приводить к «разболтке» и обеспечивать более низкую точность, особенно в случае кратных корней. Метод простых итераций исключительно прост в реализации и при этом может обеспечивать быструю сходимость к корню, особенно в случае $|\varphi'(x_*)| \ll 1$.

Пожалуй, одним из оптимальных решений в случае, когда неизвестны производные функции, является *метод Брента*, позволяющий отчасти объединить достоинства дихотомии и метода Ньютона. Следующее приближение строится по трём начальным точкам с использованием квадратичной интерполяции $x(y)$, а в тех случаях, когда найденное

с помощью интерполяции приближение не обеспечивает сходимости, применяется бисекция. Такой подход позволяет одновременно достичь гарантированной сходимости при относительно высокой скорости, не уступающей методу дихотомии на самых «плохих» функциях, и значительно ускорить сходимость в «хороших» случаях. Формулы для вычислений по методу Брента вместе с программным кодом на языке Си можно найти в книге [4].

Методы итераций, Ньютона и секущих могут использоваться для поиска комплексных корней. Однако если функция $f(z)$ принимает вещественные значения на вещественной оси, для сходимости итераций к комплексному корню требуется комплексное начальное приближение x_0. Метод парабол [2] (метод Мюллера) лишён этого недостатка — он может естественным образом сойтись к комплексному корню даже в случае вещественного начального приближения. Метод, основанный на интегральной формуле Коши, значительно уступает по эффективности другим подходам при наличии хорошего начального приближения к корню, однако может оказаться более эффективным, если положение искомого корня неизвестно.

Дополнительную информацию о решении конечных уравнений, включая рассмотрение задачи о поиске нескольких корней, в том числе кратных, можно найти в монографии [2]. Целый ряд методов и их реализация на языке Си обсуждаются в [4]. Готовый код можно также найти в библиотеках GNU scientific library и boost.

Упражнения

1. Найти уровень энергии основного состояния в одномерной прямоугольной яме, решая уравнение (4).

2. Следующие фрагменты кода содержат «небольшие» ошибки, которые с большой вероятностью могут не проявляться при тестировании. Для каждого фрагмента кода объясните, в чём заключаются ошибки, а также укажите значения параметров a, b, k и d, при которых программа будет давать *существенно* неверный результат при поиске корня линейной функции $f(x) = kx + d$:

```
//способ 1: каждый шаг дихотомии даёт 1 бит мантиссы -
//в цикле делаем столько шагов, сколько бит в мантиссе:
for(int i = 0;  i < DBL_MANT_DIG;  ++i) {
    c = (a+b)/2;
    if(f(c)*f(a) > 0)  a=c;  else b=c;
}
```

```
//способ 2: продолжаем итерации до тех пор, пока
//разность a и b не станет машинным нулем:
while(b-a > DBL_MIN) {
    c = (a+b)/2;
    if(f(c)*f(b) < 0)  a=c;  else b=c;
}

//способ 3: продолжаем итерации до тех пор, пока
//разность a и b не станет меньше машинного эпсилон:
while(b-a > DBL_EPSILON) {
    c = (a+b)/2;
    if(f(c)*f(a) > 0)  a=c;  else b=c;
}

//способ 4:
c = (a+b)/2;
do {
    if(f(c)*f(a) < 0)  b=c;  else a=c;
    c = (a+b)/2;
}while((a<c)&&(c<b));

//способ 5:
c = (a+b)/2;
do {
    if(f(c)*f(a) > 0)  a=c;  else b=c;
    c = (a+b)/2;
}while((a<c)&&(c<b));
```

3. Пусть $f(x)$ — непрерывная функция, и $f(x_*) = 0$, $x_* \in [a, b]$. Может ли метод дихотомии на отрезке $[a, b]$ не сойтись к x_*?

4. Пусть $f(x) = P_n(\mathrm{tg}\,(x))$ — полином степени n от $\mathrm{tg}\,(x)$. Если x_* — единственный корень функции $f(x)$ на отрезке $[a, b]$. Может ли метод дихотомии на указанном отрезке не сойтись к x_*? Изменится ли ответ, если x_* — простой корень?

5. Исследовать зависимость остаточного члена $R_n \equiv x_* - x_n$ от номера итерации n при решении уравнения $x = \frac{1}{2}(x + \cos x)$ методом итераций (6) с использованием и без использования поправки Эйткена (10). Что будет, если уравнение заменить на $x = \frac{1}{2}(x + \frac{2}{x})$?

6. Найти область притяжения к корню уравнения $\operatorname{th} x = 0$ при решении методом Ньютона.

7. Из (12) следует, что *для любой* гладкой функции $f(x)$, такой что $f'(x_*) \neq 0$ и $f''(x_*) \neq 0$, где x_* — корень $f(x_*) = 0$, $\exists \delta > 0$ такая, что $\forall x_0 : |x_0 - x_*| > \delta$ будет выполняться $|R_n| < |R_{n+1}|$, где $R_n \equiv x_* - x_n$, т. е. итерации будут расходится, если x_0 лежит вне области притяжения к корню, имеющей конечную ширину. Найдите ошибку в рассуждениях и приведите примеры функций, для которых итерации в методе Ньютона сходятся безусловно.

4. Численное интегрирование

Многие физические величины выражаются через интегралы: работа равна интегралу от силы $A = \int F dx$, перемещение — интеграл от скорости $x = \int \dot{x} dt$, вероятность в квантовой механике — $p = \int |\psi|^2 \, dx$ и т. д. Необходимость приближённого численного нахождения интегралов возникает, когда результат не может быть представлен точно через доступные на ЭВМ библиотечные функции, а также когда подынтегральное выражение известно не в символьном виде, а получено в результате численных расчётов (задано в виде таблицы значений на некоторой сетке). В данном разделе мы рассмотрим несколько способов приближённого вычисления определённых интегралов, начав с наиболее простых квадратурных формул и постепенно переходя к более сложным и точным.

4.1. Формула левых (правых) прямоугольников

Пожалуй, самая простая квадратурная формула получается, если заменить значение интеграла $\int_a^b f(x)\,dx$ площадью прямоугольника со сторонами, равными длине отрезка $b-a$ и значению подынтегральной функции на левом краю отрезка $f(a)$:

$$S_{[a,b]} \equiv \int\limits_a^b f(x)\,dx = f(a)\cdot(b-a) + R, \tag{17}$$

где S — точное значение искомого интеграла, $f(a)\cdot(b-a)$ — аппроксимирующая интеграл площадь прямоугольника (рис. 5 (a)), R — *остаточный член*, равный разности точного и приближённого значений.

Очевидно, остаточный член в выражении (17) можно уменьшить, если разбить отрезок $[a,b]$, введя сетку $a = x_0 < x_1 < \ldots < x_N = b$ и применив формулу (17) N раз. Получим *составную*, или *обобщённую* формулу прямоугольников:

$$S_{[a,b]} = \sum_{i=1}^{N} S_{[x_{i-1},x_i]} = \sum_{i=1}^{N} f(x_{i-1})h_i \,+\, R_\Sigma, \quad \text{где} \quad h_i = x_i - x_{i-1}. \tag{18}$$

Сумма в выражении (18) в пределе $N\to\infty$, $\max h_i \to 0$ переходит в определение интеграла Римана, откуда немедленно следует $R_\Sigma \to 0$. Для вычисления (18) на ЭВМ необходимо использовать конечные N, что будет приводить к погрешности метода $R_\Sigma \neq 0$. Оценим остаточный член R, для простоты рассматривая вначале интеграл на одном отрезке (17) и полагая функцию f непрерывно дифференцируемой требуемое число раз. Заменяя в (17) $f(x)$ суммой ряда Тейлора $f(a) + f'(a)\cdot(x-a) + \ldots$, получаем

$$R = \frac{1}{2}f'(a)\cdot(b-a)^2 + \mathcal{O}\left((b-a)^3\right). \tag{19}$$

При разбиении отрезка $[a,b]$ на N отрезков в (18) войдет сумма остаточных членов вида (19). Получим для неё асимптотическую оценку, полагая для простоты сетку $\{x_0,\ldots,x_N\}$ равномерной, т. е. $h_i \equiv h = \frac{b-a}{N} = \text{const}$:

$$R_\Sigma = h\sum_{i=1}^{N}\left[\frac{1}{2}f'(x_{i-1})h + \mathcal{O}(h^2)\right] = \frac{h}{2}\int\limits_a^b f'(x)\,dx \,+\, \mathcal{O}(h^2). \tag{20}$$

Замена $f'(x_{i-1})h \approx \int_{x_{i-1}}^{x_i} f'(x)\,dx$ в (20) даёт ошибку $\mathcal{O}(h^2)$, что находится за пределами точности сделанной оценки. Заметим также, что при вычислении суммы в (20) было использовано $\sum \mathcal{O}(h^2) = \mathcal{O}(h)$, поскольку число слагаемых в сумме $N = \mathcal{O}(h^{-1})$.

Как видно из (20), остаточный член убывает пропорционально $h^1 \sim 1/N^1$, в связи с чем говорят, что составная формула левых (правых) прямоугольников имеет *первый порядок точности по шагу сетки* h.

Подчеркнём, что для каждой квадратурной формулы существует три связанных друг с другом значения порядка точности: n и $(n-1)$ — порядки «простой» и составной формулы по шагу сетки h, $(n-2)$ — алгебраический порядок точности формулы (см. определение ниже).

Определение. Алгебраическим порядком точности называют максимальную степень полинома, для которого квадратурная формула является точной.

Очевидно, что метод правых (левых) прямоугольников даст точный результат только при интегрировании полиномов нулевой степени, т. е. алгебраический порядок точности этих формул равен нулю.

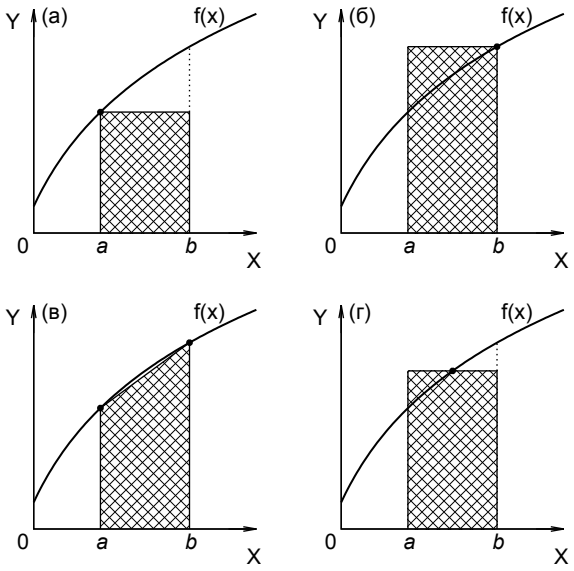


Рис. 5. Квадратурные формулы: (а,б) левых / правых прямоугольников, (в) трапеций, (г) центральных прямоугольников (формула средних)

Аналогично формулам левых прямоугольников (17) и (18), мож-

но получить выражения для интеграла, аппроксимируя его площадью прямоугольника, высота которого равна значению функции на правом конце отрезка, $f(b)$:

$$S_{[a,b]} \equiv \int\limits_a^b f(x)\,dx = f(b)\cdot(b-a) + R, \tag{21}$$

$$S_{[a,b]} = \sum_{i=1}^{N} f(x_i)h_i \;+\; R_\Sigma. \tag{22}$$

Для оценки погрешности формулы правых прямоугольников нужно заменять функцию $f(x)$ её разложением в ряд Тейлора как под интегралом, так и $f(b)$ в правой части (21), в результате чего получим:

$$R = -\frac{1}{2}f'(a)\cdot(b-a)^2 + \mathcal{O}\left((b-a)^3\right). \tag{23}$$

Суммируя остаточный член (23) по отрезкам $[x_{i-1}, x_i]$, можно получить $R_\Sigma = \mathcal{O}\,(h)$, аналогично формуле (20) для левых прямоугольников. Ввиду медленного ($\sim 1/N$) убывания погрешности в формулах левых (правых) прямоугольников, данные методы практически не используются в численных расчётах.

4.2. Формула трапеций

Легко заметить, что главные члены разложения ошибки по степеням $b-a$ в формуле левых (19) и правых (23) прямоугольников в точности противоположны. Суммируя (17) и (21) и деля на 2, получаем *формулу трапеций*:

$$S_{[a,b]} \approx \frac{f(a)+f(b)}{2}(b-a). \tag{24}$$

Геометрический смысл (24) — аппроксимация интеграла площадью трапеции (рис. 5 (*в*)). Разбивая отрезок $[a,b]$ на N частей $[x_0, x_1]$, …, $[x_{N-1}, x_N]$ и применяя к каждой части формулу (24), получем обобщённую, или составную формулу трапеций, которая в случае равномерного разбиения $x_i - x_{i-1} \equiv h = \mathrm{const}$ будет иметь вид:

$$S_{[a,b]} \approx h\cdot\left(\frac{1}{2}f_0 + f_1 + f_2 + \ldots + f_{N-1} + \frac{1}{2}f_N\right), \quad \text{где}\, f_i \equiv f(x_i). \tag{25}$$

Для вычисления остаточного члена в (24) разложим функцию f в ряд Тейлора в точке $c = \frac{1}{2}(a+b)$, удерживая в разложении члены до третьей степени включительно:

$$R = -\frac{1}{12} f''(c) \cdot (b-a)^3 + \mathcal{O}\left((b-a)^4\right). \tag{26}$$

Суммируя (26) по отрезкам $[x_0, x_1], \ldots, [x_{N-1}, x_N]$, получим асимптотическую оценку остаточного члена составной формулы трапеций (25):

$$R_\Sigma = -\frac{h^2}{12} \int\limits_a^b f''(x)\, dx \, + \, \mathcal{O}(h^3). \tag{27}$$

При увеличении числа интервалов разбиения N ошибка в формуле трапеций убывает как $\mathcal{O}(h^2) \sim 1/N^2$, т. е. формула трапеций имеет второй порядок точности относительно шага сетки. Алгебраический порядок точности формулы трапеций, очевидно, равен 1, так как формула точна для полиномов степени не выше первой. На рис. 5 (*в*) хорошо видно, что погрешность формулы трапеций существенно меньше погрешности формул левых (правых) прямоугольников.

4.3. Формула средних

Ещё одна квадратурная формула второго порядка может быть получена при аппроксимации интеграла $\int_a^b f(x)\, dx$ площадью прямоугольника с высотой равной $f(\frac{a+b}{2})$ (рис. 5 (*г*)):

$$S_{[a,b]} \approx f\left(\frac{a+b}{2}\right) \cdot (b-a). \tag{28}$$

Остаточный член для формулы (28) и соответствующей составной формулы:

$$R = \frac{(b-a)^3}{24} \cdot f''\left(\frac{a+b}{2}\right) \, + \, \mathcal{O}\left((b-a)^4\right). \tag{29}$$

$$R_\Sigma = \frac{h^2}{24} \int\limits_a^b f''(x)\, dx \, + \, \mathcal{O}(h^3). \tag{30}$$

Обратим внимание, что главный член асимптотического разложения остаточного члена (29) в формуле средних в два раза меньше по величине и противоположен по знаку главному члену разложения погрешности для формулы трапеций (26). Поэтому формулу средних (28)

следует предпочесть формуле трапеций (24), если значения подынтегральной функции f могут быть одинаково легко вычислены в любых точках. Заметим, однако, что использование (28) не всегда возможно (например, если подынтегральная функция f задана в виде таблицы значений, или, как говорят, на некоторой сетке $x_0, x_1, \ldots, x_N$).

4.4. Формула Симпсона

Другой важный вывод, который можно сделать из отмеченного выше соотношения главных членов разложения погрешностей, состоит в том, что порядок квадратурных формул (24) и (28) можно легко повысить, занулив главный член асимптотического разложения ошибки. Действительно, умножая (28) на 2, суммируя с (24) и деля на 3, приходим к *формуле Симпсона*:

$$S_{[a,b]} \approx \frac{b-a}{6} \cdot \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]. \tag{31}$$

Вводя на отрезке $[a, b]$ равномерную сетку $a = x_0, x_1, \ldots, x_{2k-1}, x_{2k} = b$, можно записать составную (обобщённую) формулу Симпсона:

$$S_{[a,b]} \approx \frac{h}{3} \cdot \big(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{2k-2} + 4f_{2k-1} + f_{2k} \big). \tag{32}$$

Для остаточного члена в формулах (31) и (32) на равномерной сетке с шагом h несложно получить:

$$R = -\frac{h^5}{90} f^{(\mathrm{iv})}\left(\frac{a+b}{2}\right) + \mathcal{O}(h^6), \quad h = \frac{b-a}{2}, \tag{33}$$

$$R_\Sigma = -\frac{h^4}{180} \int\limits_a^b f^{(\mathrm{iv})}(x)\, dx + \mathcal{O}(h^5). \tag{34}$$

При увеличении числа интервалов разбиения $N = 2k$ ошибка в формуле Симпсона убывает как $\mathcal{O}(h^4) \sim 1/N^4$, т. е. составная формула Симпсона (32) имеет четвёртый порядок точности относительно шага сетки.

Благодаря быстрому убыванию ошибки с увеличением числа узлов сетки, малому численному коэффициенту в остаточном члене (34) и простому виду, формула (32) широко применяется в численных расчётах.

4.5. Метод Рунге

Рассмотрим регулярный способ, позволяющий повышать порядок точности квадратурных формул. Для этого запишем точное значение интеграла S в виде

$$S = F(h) + R(h) = F(h) + Ah^{\rho} + \mathcal{O}(h^{\rho+1}), \tag{35}$$

где $F(h)$ — значение интеграла, вычисленное с помощью квадратурной формулы на сетке с шагом h. Обозначив за ρ порядок точности квадратурной формулы, мы записали в (35) остаточный член $R(h)$ в виде $Ah^{\rho} + \mathcal{O}(h^{\rho+1})$, где A — коэффициент пропорциональности. Повторяя расчёты на сетке с шагом rh, где r — некоторый коэффициент, можем записать аналогично (35):

$$S = F(rh) + A \cdot (rh)^{\rho} + \mathcal{O}(h^{\rho+1}). \tag{36}$$

Рассматривая (35) и (36) как систему уравнений относительно $R = Ah^{\rho}$ и S, имеем:

$$R = \frac{F(h) - F(rh)}{r^{\rho} - 1} + \mathcal{O}(h^{\rho+1}), \tag{37}$$

$$S = \frac{F(h)r^{\rho} - F(rh)}{r^{\rho} - 1} + \mathcal{O}(h^{\rho+1}). \tag{38}$$

Формула (37) даёт апостериорную оценку точности вычисленного значения интеграла $F(h)$, тогда как выражение (38) позволяет уточнить найденное значение интеграла, повысив порядок точности квадратурной формулы с ρ до $\rho + 1$.

Применимость метода Рунге не ограничивается численным интегрированием: аналогичный приём может быть использован для оценки погрешности либо повышения скорости сходимости самых разных численных процедур. В случае, если порядок численного метода (итерационного процесса) *a priori* не известен, можно провести аналогичное рассмотрение, считая ρ одной из неизвестных в системе (35, 36), дополненной уравнением вида $S = F(r^2h) + A \cdot (r^2h)^{\rho}$. Данный алгоритм известен как *процесс Эйткена* и рассмотрен в [2].

Заметим также, что если проводить вычисления на последовательности сеток $h, rh, \ldots, r^n h$, можно добиться повышения порядка точности численного метода на n.

4.6. Интегралы по бесконечной области

Часто в физике возникают несобственные интегралы по бесконечной области: по всему пространству, по бесконечному времени и т. п.

В данном разделе кратко рассмотрены основные приёмы, используемых для нахождения численных значений таких интегралов на ЭВМ.

В случае, если подынтегральная функция убывает достаточно быстро (экспоненциально) при больших значениях аргумента, можно использовать самый простой способ — ограничение области интегрирования. Так, пусть необходимо вычислить интеграл $\int_a^\infty f(x)\,dx$ и пусть $\exists k > 0$, A и x_0: $|f(x)| \le Ae^{-kx}$ $\forall x > x_0$. Заменяя верхний предел интегрирования конечным значением $b \ge x_0$, мы допустим ошибку $R = \int_b^\infty f(x)\,dx$. Оценим погрешность, мажорируя интеграл: $|R| \le \int_b^\infty Ae^{-kx} = Ae^{-kb}$. Требуя, чтобы погрешность метода не превосходила некоторого δ, можно оценить верхний предел интегрирования: $b = \frac{1}{k}\ln(A/\delta)$. Видно, что уменьшение δ приводит к медленному (логарифмическому) росту верхнего предела интегрирования b. Например, при $k = 1$, $A = 1$, $\delta = 10^{-15}$ получаем $b = 34{,}5$.

Подчеркнём, что указанный способ неприменим для функций, убывающих на бесконечности медленно (по степенному закону). Рассмотрим $|f(x)| \le Ax^{-n}$ $\forall x > x_0$, $n > 1$. В этом случае мажорантная оценка для погрешности замены верхнего предела интегрирования на b будет иметь вид $|R| = |\int_b^\infty f(x)\,dx| \le |\frac{A}{n-1}|b^{1-n}$. Накладывая условие $|R| \le \delta$, получаем

$$b = \left|\frac{A}{(n-1)\delta}\right|^{\frac{1}{n-1}}.$$

Для $A = 1$, $n = 2$, $\delta = 10^{-8}$ имеем $b = 10^8$. Большая величина полученного значения обуславливает необходимость использования более эффективных методов интегрирования медленно убывающих функций.

Для вычисления несобственных интегралов функций, имеющих на бесконечности степенну́ю асимптотику, обычно используют замену переменной. Например, делая замену $x = t/(1-t)$, получаем

$$S = \int\limits_0^\infty f(x)\,dx = \int\limits_0^1 \frac{f(x(t))\,dt}{(1-t)^2}.$$

В результате имеем интеграл по конечной области, однако как правило, он также является несобственным ввиду наличия особенности в подынтегральном выражении при $t \to 1-0$. Способам вычисления интегралов такого типа посвящён следующий пункт.

4.7. Интегралы с особенностью

Пусть требуется вычислить значение интеграла $\int_a^b f(x)\,dx$ в конечных пределах, при этом функция $f(x)$ может иметь интегрируемые особенности в точках a и b.

В случае, если $f(x)$ имеет лишь устранимые особенности, для вычисления интеграла достаточно переопределить функцию f в особых точках либо в их малых окрестностях. Например, пусть требуется вычислить $S = \int_0^1 \frac{\sin x}{x}\,dx$. Подынтегральная функция имеет устранимую особенность в точке $x = 0$, поэтому в программном коде следует доопределить подынтегральную функцию в нуле:

```
double f(double x) { return (x == 0) ? 1 : sin(x)/x; }
```

Если проверка условия `x==0` даст логическую истину, функция `f` вернёт 1, в противном случае будет вычислено отношение `sin(x)/x`.

В основу указанного решения положено сравнение двух чисел с плавающей точкой на точное равенство — неиссякаемый источник ошибок численных алгоритмов, см. п. 4 на с. 39. Как следствие, описанный способ может плохо работать или даже приводить к сбоям в программе в случае, когда погрешность вычисления числителя дроби существенна, а знаменатель близок к нулю. Более надёжным решением является замена подынтегральной функции суммой первых членов ряда Тейлора в окрестности устранимой особой точки. Для рассмотренного выше примера имеем:

$$\frac{\sin x}{x} = 1 - \frac{x^2}{6} + \frac{x^4}{120} - \frac{x^6}{5040} + \mathcal{O}(x^8).$$

Откуда делаем вывод, что при $|x| < \sqrt{6\varepsilon}$ значение подынтегральной функции будет неотличимо от 1 (здесь ε — машинное эпсилон), что позволяет безопасно вычислять подынтегральную функцию следующим образом:

```
double f(double x)
{
    const double x0 = 3.6e-8;  //3.6e-8 ~ sqrt(6*DBL_EPSILON)
    return (fabs(x) < x0) ? 1 : sin(x)/x;
}
```

Рассмотрим теперь другой случай: пусть функция $f(x)$ имеет *неустранимую* интегрируемую особенность, так что $|f|$ неограниченно возрастает в окрестности конечного множества точек на промежутке интегрирования. Без ограничения общности можно считать, что $f(x)$

имеет особенность только на концах промежутка — в противном случае следует разбить отрезок на несколько частей, представив интеграл в виде суммы интегралов по отрезкам с концами в особых точках подынтегральной функции.

Для вычисления интегралов в указанных условиях существует ряд приёмов. Например, можно использовать аддитивное выделение особенности, представляя $f(x) = \varphi(x) + \psi(x)$, где $|\varphi(x)| < \infty$, а $\psi(x)$ интегрируется аналитически. Другой способ состоит в мультипликативном выделении особенности: $f(x) = \varphi(x) \cdot \psi(x)$; полученный интеграл с весовой функцией может быть вычислен с помощью квадратурных формул Гаусса-Кристоффеля [2]. Недостатком этого и ряда других приёмов, описанных в [2], является необходимость поиска нестандартного решения для каждой задачи. В главе 4.5 книги [4] рассмотрен простой алгоритм, свободный от указанного недостатка. Он позволяет вычислять в общем виде несобственные интегралы и обладает при этом высокой скоростью сходимости. Суть его состоит в следующем. Пусть требуется вычислить интеграл $S = \int_a^b f(x)\,dx$. Сделаем под интегралом замену переменной:

$$x = \frac{1}{2}(a+b) + \frac{1}{2}(b-a) \cdot \operatorname{th} t, \tag{39}$$

Тогда

$$S = \int\limits_{-\infty}^{+\infty} f(x(t)) \frac{dx}{dt}\,dt = R_1 + \int\limits_{-T}^{T} f(x(t)) \cdot \frac{b-a}{2} \cdot \frac{dt}{\operatorname{ch}^2 t}, \tag{40}$$

где $\operatorname{th} t$ и $\operatorname{ch} t$ — гиперболический тангенс и косинус t, соответственно. Интегрирование в (40) ведется в бесконечных пределах, однако подынтегральная функция быстро убывает ввиду наличия множителя $\operatorname{ch}^{-2} t$, что позволяет заменить пределы интегрирования на конечные, от $-T$ до T, и вычислить полученный интеграл с помощью любой квадратурной формулы (например, формулы Симпсона). Заметим, что погрешность при вычислении интеграла будет складываться из двух составляющих, $R = R_1 + R_2$. Первая связана с отбрасыванием экспоненциально убывающих бесконечных «хвостов», вторая — с заменой интеграла в конечных пределах на значение квадратурной формулы. Увеличивая T, можно сделать ошибку R_1 сколь угодно малой. Однако при фиксированном числе точек N увеличение T будет приводить к росту величины шага сетки $h = 2T/(N-1)$ и, как следствие, к увеличению остаточного члена R_2 в квадратурной формуле. Напротив, уменьшая шаг сетки h, можно уменьшить остаточный член R_2, однако при $N = \text{const}$ это

приведёт к уменьшению T и росту R_1. Следовательно, должен существовать некоторый компромисс между указанными двумя источниками погрешности, позволяющий минимизировать величину суммарной ошибки $R = R_1 + R_2$. Условие $R_1 \approx R_2$ приводит к оценке оптимальной величины шага [4]

$$h \approx \frac{\pi}{\sqrt{2N}}. \tag{41}$$

Заметим, что для различных подынтегральных функций, обладающих различной расходимостью в точках $x = a, b$, коэффициент в (41) может варьироваться, поэтому выражение (41) даёт, как правило, субоптимальную величину шага. Однако ввиду быстрой сходимости рассмотренного метода и удобства использования универсальных квадратурных формул, оптимизацию в каждом отдельном случае обычно не выполняют.

Также заметим, что хотя замена (39) даёт $a < x(t) < b$ $\forall t$, из-за ограниченной точности машинных вычислений при достаточно больших $|t|$ возможно выполнение условий $x(t) = a$ или $x(t) = b$. Поскольку подынтегральная функция может иметь особенность в точках $x = a, b$, вычисление f в этих точках будет приводить к сбоям в программе или получению в качестве результата не-числовых значений `inf` или `NaN`. Следовательно, для корректной работы алгоритма необходимо отслеживать в программе выполнение условий $a < x(t) < b$.

4.8. Сравнение методов

В данном разделе мы рассмотрели лишь наиболее простые квадратурные формулы, ограничившись лишь случаем равномерных сеток (формулы *Ньютона-Котеса*). На практике широко используется формула Симпсона, реже применяются формулы, имеющий первый алгебраический порядок точности ($R_\Sigma = \mathcal{O}\left(h^2\right)$), тогда как формулы нулевого порядка практически не используются ввиду крайне низкой скорости сходимости. Исключением является вычисление нормы функций на равномерных сопряжённых сетках, связанных преобразованием Фурье. В этом случае формулы правых, левых и центральных прямоугольников и трапеций дают $\frac{1}{N}\sum|f_i|^2$. Указанная сумма сохраняется при дискретном преобразовании Фурье (равенство Парсеваля), что делает использование формул прямоугольников предпочтительнее метода Симпсона для данной задачи.

При интегрировании функций, имеющих гладкие производные высоких порядков, могут применяться квадратурные формулы более высоких порядков точности. Наиболее употребительными являются квад-

ратурные формулы Гаусса:

$$\int_a^b f(x)\,dx \approx \sum_{j=1}^{N} w_j f(x_j),$$

где весовые коэффициенты w_j и расположение узлов x_j, $j = 1, \ldots, N$ выбираются с целью достижения максимально возможного[22] порядка точности [A10] при заданном числе узлов сетки N. Кроме того, формулы Гаусса являются устойчивыми независимо от N, тогда как коэффициенты в формулах Ньютона-Котеса высокого порядка точности ($N \geq 10$) становятся знакопеременными [A10], что приводит к резкому росту погрешности квадратурной формулы (численной неустойчивости). Анализ свойств формул Гаусса с доказательством соответствующих теорем можно найти в монографии [A10]. В книге [4] приведён готовый к использованию код программы на языке Си для вычисления интегралов по десяти точкам с использованием квадратурной формулы Гаусса.

Подчеркнём, что использование формул высокого *порядка* ещё не означает получение высокой *точности* численного решения. Применение формул высокого порядка даёт выигрыш в точности только для достаточно гладких функций, которые притом хорошо аппроксимируются полиномами.

Помимо значений подынтегральной функции, квадратурные формулы могут включать также значения её производных [2], что позволяет повысить их точность, однако усложняет вычисления.

Заметим также, что хотя приведённые в данном разделе квадратурные формулы и позволяют решить подавляющее большинство практических задач, в некоторых случаях эффективным решением является использование нестандартных формул. Это связано, прежде всего, с вычислением интегралов с весовой функцией вида $\int y(x)w(x)\,dx$. Весовая функция $w(x)$ может иметь особенность или, например, быстро осциллировать на фоне медленно меняющейся амплитуды $y(x)$. В этом случае применение специальных квадратурных формул позволяет существенно повысить эффективность расчётов за счёт использования более крупного шага численного интегрирования. Для ряда часто встречающихся весовых функций готовые решения можно найти в литературе. Например, для вычисления интегралов вида $\int y(x)\exp(i\omega x)\,dx$ используются *формулы Филона* [2].

[22] Алгебраический порядок точности формул Гаусса равен $2N - 1$, так что остаточный член составных формул $R_\Sigma = \mathcal{O}(h^{2N})$.

Упражнения

1. Полагая функцию $f(x)$ представимой в виде суммы ряда Тейлора на промежутке интегрирования $[a, b]$, показать, что если некоторая квадратурная формула $\int_a^b f(x)\,dx = \sum w_i f(x_i) + R$ имеет алгебраический порядок точности n, то остаточный член этой формулы $R = \mathcal{O}(h^{n+2})$, а остаточный член построенной на её основе составной формулы $R_\Sigma = \mathcal{O}(h^{n+1})$.

2. Оценить *a priori* число точек, в которых необходимо вычислить подынтегральную функцию для нахождения значения функции стандартного нормального распределения $\Phi_{0,1}(2)$ с точностью не хуже 0,1 % с использованием формулы Симпсона:

$$\Phi_{0,1}(x) = \frac{1}{\sqrt{2\pi}} \int\limits_{-\infty}^{x} \exp\left(-\frac{\xi^2}{2}\right) d\xi. \tag{42}$$

3. Вычислить значение интеграла $\int_0^{\pi/4} \sin x\,dx$ различными методами. Построить графики апостериорной оценки погрешности интегрирования в зависимости от числа точек N, в которых вычислялась подынтегральная функция. При построении графика использовать двойной логарифмический масштаб (`set logscale xy` в gnuplot). Сравнить с графиками степенных функций, построенных в тех же осях, объяснить результат.

4. Написать программу для вычисления $\Phi_{0,1}(x)$ по формуле (42) в произвольной точке x с абсолютной погрешностью не выше 10^{-10}.

5. Написать программный код для вычисления функции Бесселя целого порядка n с абсолютной погрешностью не выше 10^{-10}. Как следует вычислять интеграл (43) при $n \gg 1$ и (или) $|x| \gg 1$?

$$J_n(x) = \frac{1}{\pi} \int\limits_0^{\pi} \cos(n\varphi - x\sin\varphi)\,d\varphi, \quad n = 0, 1, 2, \ldots \tag{43}$$

6. Может ли погрешность численного интегрирования при использовании квадратурной формулы Симпсона убывать быстрее, чем h^4? Медленнее, чем h^4? (Здесь h — шаг равномерной сетки).

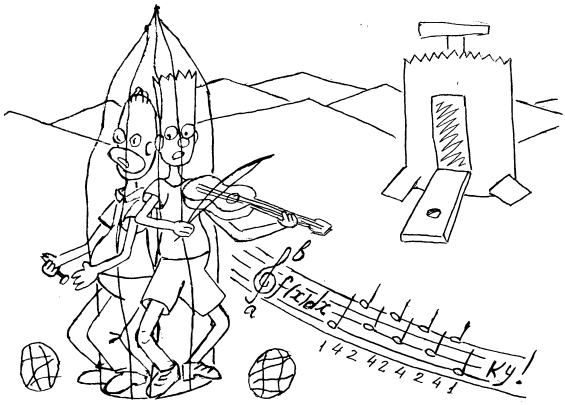
7. Вычислить значение интеграла

$$8A^{5/2}\int\limits_0^1\left(\sqrt{A+\sin x}-\sqrt{A-\sin x}-\frac{\sin x}{\sqrt{A}}\right)\frac{dx}{x^3},\quad A=10^5.$$

8. Вычислить несобственный интеграл $\int_0^\infty x^{-1/3}\exp[-x+\sin x]\,dx$.

9. Построить график зависимости периода осцилляций от амплитуды для физического маятника: $L(\theta,\dot\theta)=\frac{1}{2}ml^2\dot\theta^2+mgl\cos\theta$, $T=\sqrt{2m}\int dx/\sqrt{E-U(x)}$.

10. Вычислить интеграл Кирхгофа в параксиальном приближении и построить двумерный график распределения интенсивности в плоскости на расстоянии L за непрозрачным экраном с отверстием квадратной / круглой / треугольной формы.



5. Интерполяция полиномами

В математической физике функции $y(x)$ обычно являются отображением некоторого непрерывного множества значений аргумента x (например, всей числовой оси) на множество значений y. Функция y может быть вычислена в любой точке x из области определения. Однако на практике при решении большого количества научно-технических

задач приходится иметь дело с функциями, заданными конечным набором значений $y(x_i)$, $i = 0, \dots n$. Вычисление функции y в большем числе точек может оказаться нежелательным ввиду значительных затрат времени или иных ресурсов. Например, каждое значение $y(x_i)$ может получаться в результате решения некоторой сложной математической задачи, либо быть результатом длительного и дорогостоящего эксперимента. Зачастую в решении задачи возникают специальные функции, заданные в виде интеграла либо как решение дифференциального уравнения. Если значения таких функций необходимо вычислять многократно, это может быть сопряжено со значительными затратами времени. В таких случаях желательно вычислить функцию y в некотором относительно небольшом наборе точек, и затем использовать вместо $y(x)$ в расчётах некоторую другую функцию $\varphi(x) \approx y(x)$, значения которой могут быть относительно легко и быстро вычислены в любой нужной точке. Процесс построения и вычисления $\varphi(x)$ называют *аппроксимацией*. В случае, если $\varphi(x)$ в точности проходит через имеющиеся точки $x_i, y(x_i)$, говорят о частном случае аппроксимации — *интерполяции*. Нередко (хотя и далеко не всегда) на практике для интерполяции используются полиномы (либо кусочно-полиномиальная интерполяция — сплайны), поэтому знакомство с задачей интерполяции мы начнём именно с полиномов.

Пусть значения функции $y(x)$ известны на сетке $x_0, x_1, \dots, x_n$. Необходимо найти полином степени n, который бы совпадал с $y(x)$ во всех узлах указанной сетки:

$$P_n(x) = a_0 + a_1 x + \ldots + a_n x^n, \quad P_n(x_i) = y(x_i) \quad \forall i = 0, 1, \dots, n, \quad (44)$$

Докажем *корректность* поставленной задачи.

Теорема. В случае если все x_i различны, решение задачи о построении интерполяционного полинома (44) существует и единственно.

Прежде чем приступать к доказательству данного утверждения, докажем вначале следующую лемму.

Лемма. Определитель Вандермонда ($\Delta_n \equiv |A_{ij}|, A_{ij} = x_i^j$) может быть представлен в виде произведения $\frac{1}{2}n(n+1)$ сомножителей:

$$\Delta_n = \begin{vmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{vmatrix} = \prod_{0 \le i < j \le n} (x_j - x_i). \quad (45)$$

Доказательство. Используем метод математической индукции.

Для этого рассмотрим исходное утверждение при $n = 2$ (базис индукции):

$$\Delta_2 = \begin{vmatrix} 1 & x_0 \\ 1 & x_1 \end{vmatrix} = x_1 - x_0,$$

что совпадает с общей формулой (45).

Предположение индукции: пусть формула (45) доказана для $n = 2$, $3, \ldots, m-1$. Тогда для $n = m-1$ можем записать:

$$\Delta_{m-1} = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{m-1} & x_{m-1}^2 & \ldots & x_{m-1}^{m-1} \end{vmatrix} = \prod_{0 \le i < j \le m-1} (x_j - x_i). \quad (46)$$

Сделаем шаг индукции — докажем (45) для $n = m$. Запишем определитель (45) для $n = m$, при этом вычитая из k-го столбца $(k-1)$-й, умноженный на x_0, последовательно для $k = m+1, m, m-1, \ldots, 2$. Поскольку данное преобразование не изменяет значение определителя Δ_m, получим

$$\Delta_m = \begin{vmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & x_1 - x_0 & (x_1 - x_0)x_1 & \ldots & (x_1 - x_0)x_1^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m - x_0 & (x_m - x_0)x_m & \ldots & (x_m - x_0)x_m^{m-1} \end{vmatrix}.$$

Разложим этот определитель по первой строке:

$$\Delta_m = \begin{vmatrix} x_1 - x_0 & (x_1 - x_0)x_1 & (x_1 - x_0)x_1^2 & \ldots & (x_1 - x_0)x_1^{m-1} \\ x_2 - x_0 & (x_2 - x_0)x_1 & (x_2 - x_0)x_2^2 & \ldots & (x_2 - x_0)x_2^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ x_m - x_0 & (x_m - x_0)x_m & (x_m - x_0)x_m^2 & \ldots & (x_m - x_0)x_m^{m-1} \end{vmatrix}.$$

Вынесем из первой строки множитель $(x_1 - x_0)$, из второй — разность $(x_2 - x_0)$ и т. д.:

$$\Delta_m = \begin{vmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} \\ 1 & x_1 & x_2^2 & \ldots & x_2^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^{m-1} \end{vmatrix} \cdot \prod_{j=1}^{m} (x_j - x_0).$$

Полученный определитель матрицы $m \times m$ может быть вычислен по индукционному предположению (46), совпадающему с полученным выражением с точностью до переобозначения x_0 на x_1, x_1 на x_2 и т. д. до

x_{m-1} на x_m. Это приводит нас к

$$\Delta_m = \left(\prod_{1 \le i < j \le m} (x_j - x_i) \right) \cdot \left(\prod_{j=1}^{m} (x_j - x_0) \right) = \prod_{0 \le i < j \le m} (x_j - x_i).$$

Лемма доказана. ■

Докажем теперь теорему о существовании и единственности. Для построения полинома (44) необходимо найти коэффициенты $a_0, \ldots, a_n$, используя условия интерполяции $P_n(x_0) = y_0, P_n(x_1) = y_1, \ldots, P_n(x_n) = y_n$, где $y_i \equiv y(x_i)$. Это приводит к системе линейных алгебраических уравнений на коэффициенты $a_0, a_1, \ldots, a_n$:

$$\begin{cases} a_0 + a_1 \cdot x_0 + a_2 \cdot x_0^2 + \ldots + a_n \cdot x_0^n & = & y_0 \\ a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \ldots + a_n \cdot x_1^n & = & y_1 \\ \ldots & = & \ldots \\ a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \ldots + a_n \cdot x_n^n & = & y_n. \end{cases}$$

Определитель данной системы есть определитель Вандермонда (45). Поскольку все x_i различны, определитель (45) отличен от нуля. Следовательно, система линейных алгебраических уравнений на коэффициенты $a_0, \ldots, a_n$ имеет решение при любых y_i, причём это решение единственно. Корректность поставленной задачи доказана. ■

Ниже будут показаны два наиболее часто встречающихся способа записи интерполяционных полиномов. В силу доказанной выше теоремы очевидно, что разные способы отличаются друг от другой лишь формой записи, при этом сами полиномы совпадают.

5.1. Полиномы Лагранжа

Одним из наиболее очевидных способов записи интерполяционных полиномов являются *полиномы Лагранжа*. Рассмотрим вначале вспомогательный полином степени n:

$$\begin{aligned} l_i(x) &= \prod_{0 \le j \le n, j \ne i} (x - x_j) = \\ &= (x - x_0) \cdot (x - x_1) \cdot \ldots \cdot (x - x_{i-1}) \cdot (x - x_{i+1}) \cdot \ldots \cdot (x - x_n). \end{aligned} \tag{47}$$

Данный полином, очевидно, зануляется в n узлах сетки $x_0, x_1, \ldots, x_n$ — в каждом узле, за исключением i-го. Нормируя значение полинома $l_i(x)$ в i-м узле и суммируя по i, получим требуемый интерполяционный полином:

$$P_n(x) = \sum_{i=0}^{n} y(x_i) \cdot \frac{l_i(x)}{l_i(x_i)}. \tag{48}$$

Выражения (47), (48) дают решение поставленной задачи об интерполяции полиномами. Важным достоинством интерполяционных полиномов Лагранжа является их простота — ответ в форме (47), (48) может быть выписан сразу. К недостаткам следует отнести относительно большое число операций, которые необходимо выполнить для построения полиномов и их вычисления — в ряде случаев это заставляет отдать предпочтение другим формам записи интерполяционных полиномов.

Оценим число операций, которые необходимо выполнить для построения интерполяционных полиномов Лагранжа. Для вычисления каждого из $l_i(x)$ по формуле (47) необходимо выполнить n операций вычитания и $n-1$ умножение. Хотя выполнение данных операций может требовать различного машинного времени, для простоты далее будем игнорировать этот факт и считать их вместе как $2n-1$ арифметическую операцию. Для вычисления значения $P_n(x)$ по формуле (48) нужно умножить каждый из $l_i(x)$ на нормировочный множитель и сложить $n+1$ значений, итого получаем $2n^2+3n = \mathcal{O}(n^2)$ арифметических операций.

Для построения интерполяционного полинома Лагранжа необходимо, кроме того, вычислить ещё $n+1$ нормировочных коэффициентов $l_i(x_i)$; для вычисления каждого из них, очевидно, требуется выполнить ещё $2n-1$ операцию и умножить результат на y_i. Итого получаем $2n(n+1) = 2n^2+2n$ арифметических операций для вычисления коэффициентов перед первым вычислением значений интерполяционного полинома.

Ниже будет рассмотрена другая широко распространённая форма записи интерполяционных полиномов, требующая меньшего числа операций.

5.2. Полиномы Ньютона

Для построения альтернативной формы записи интерполяционных полиномов введём вначале понятие разделённой разности.

Определение. Разделённой разностью первого порядка $y(x_i, x_j)$ для функции $y(x)$ называется величина $y(x_i, x_j) \equiv (y(x_i) - y(x_j))/$ $/(x_i - x_j)$. Разделённой разностью второго порядка называется отношение $y(x_i, x_j, x_k) \equiv (y(x_i, x_j) - y(x_j, x_k))/(x_i - x_k)$. Аналогично определяются разделённые разности третьего ($y(x_i, x_j, x_k, x_l) \equiv (y(x_i, x_j, x_k) -$ $- y(x_j, x_k, x_l))/(x_i - x_l)$) и более высоких порядков. Можно показать,

что разделённая разность представима в виде

$$y(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \frac{y(x_i)}{\prod\limits_{j \neq i} (x_i - x_j)}. \tag{49}$$

Из последнего выражения немедленно следует, что разделённая разность симметрична по всем перестановкам своих аргументов $x_1, \ldots, x_n$.

Разделённые разности имеют размерности производных функции. В п. 5.4 будет показано, что разделённая разность порядка k аппроксимирует k-ю производную функции: $k! y(x_0, x_1, \ldots, x_k) = y^{(k)} + \mathcal{O}(h)$.

Покажем, как разделённые разности могут быть использованы для построения интерполяционного полинома. Рассмотрим интерполяционный полином $P(x)$ степени n, принимающий в узлах $x_0, x_1, \ldots, x_n$ значения, совпадающие со значениями интерполируемой функции y: $P(x_i) = y(x_i)$.

Вычислим для полинома $P(x)$ разделённую разность

$$P(x, x_0) = \frac{P(x) - P(x_0)}{x - x_0} \quad \Rightarrow P(x) = P(x_0) + (x - x_0) \cdot P(x, x_0). \tag{50}$$

Выражение в числителе является полиномом степени n, который зануляется в точке $x = x_0$. Следовательно, данный полином может быть разделен нацело на $x - x_0$, а полученная в результате разделённая разность $P(x, x_0)$ является полиномом степени $n-1$. Аналогично, для разделённой разности второго порядка:

$$P(x, x_0, x_1) = \frac{P(x, x_0) - P(x_0, x_1)}{x - x_1} \quad \Rightarrow$$

$$P(x, x_0) = P(x_0, x_1) + (x - x_1) \cdot P(x, x_0, x_1). \tag{51}$$

Числитель дроби вновь зануляется в силу симметрии разделённой разности по перестановкам аргументов. Следовательно, разделённая разность второго порядка является полиномом степени $n - 2$. Действуя аналогично дальше, можно показать, что разделённая разность порядка k является полиномом степени $n - k$ при $k = 1, 2, \ldots, n$. Для разделённой разности порядка $n + 1$ имеем:

$$P(x, x_0, x_1, \ldots, x_n) = \frac{P(x, x_0, \ldots, x_{n-1}) - P(x_0, \ldots, x_n)}{x - x_n}. \tag{52}$$

При $x = x_n$ числитель дроби зануляется в силу симметрии разделённых разностей по перестановкам аргументов. С другой стороны, выражение

в числителе дроби является полиномом степени 0, т. е. константой. Следовательно, числитель дроби тождественно равен нулю $\forall x$.

Последовательно подставляя друг в друга выражения (50), (51), ..., (52) и учитывая на последнем шаге, что

$$P(x, x_0, x_1, \ldots, x_{n-1}) - P(x_0, x_1, \ldots, x_{n-1}, x_n) = 0,$$

приходим к полиному, записанному по *схеме Горнера*:

$$\begin{aligned} P(x) = P(x_0) \quad & + (x - x_0) \cdot \quad [P(x_0, x_1) + \\ & + (x - x_1) \cdot \quad [P(x_0, x_1, x_2) + \ldots \\ \ldots \quad & + (x - x_{n-2}) \cdot [P(x_0, \ldots, x_{n-1}) + \\ & + (x - x_{n-1}) \cdot P(x_0, x_1, \ldots, x_n)] \ldots]] \,. \end{aligned} \tag{53}$$

Или, раскрывая все квадратные скобки в (53), можно записать формулу Ньютона в виде суммы произведений:

$$P(x) = P(x_0) + \sum_{k=1}^{n} P(x_0, x_1, \ldots, x_k) \prod_{j=0}^{k-1} (x - x_j). \tag{54}$$

Для вычисления полученных выражений необходимо заметить, что разделённые разности для полиномов $P(x_0, x_1, \ldots, x_k)$ совпадают с разделёнными разностями для интерполируемой функции $y(x)$, поскольку $P(x_i) = y(x_i) \quad \forall i = 0, \ldots, n$. Следовательно, формулы (53) и (54) после замены $P(x_0, x_1, \ldots, x_k)$ на $y(x_0, x_1, \ldots, x_k)$ дают решение задачи о построении интерполяционного полинома в форме Ньютона. При фиксированном числе узлов сетки для вычислений более удобна схема Горнера (53). Напротив, формула Ньютона (54) предпочтительнее в случае, когда необходимо контролировать точность и добавлять новые узлы в расчёт. При этом полезно помнить о том, что порядок, в котором перенумерованы узлы интерполяции x_0, x_1, ..., не играет роли в силу симметрии разделённых разностей по перестановкам аргументов.

Для вычисления разделённых разностей, входящих в выражения (53) и (54), удобно записывать коэффициенты в виде таблицы (в виде соответствующего двумерного массива при вычислении на ЭВМ). Для последующего вычисления значений интерполяционных полиномов необходимо сохранять в памяти ЭВМ только верхнюю косую строку табл. 2.

Число операций, необходимых для вычисления значения интерполяционного полинома, записаного по схеме Горнера (53), равно $3n$: на каждом из n шагов необходимо выполнить два сложения и одно

умножение. Обратим внимание, что число операций в формуле Ньютона растёт линейно по n, а не квадратично, как в формуле Лагранжа (48). Из сравнения полученных ответов для числа операций видно, что формула Ньютона более экономична по сравнению с формулой Лагранжа для любой степени n интерполяционного полинома.

Таблица 2

x_0	y_0				
x_1	y_1	$y(x_0,x_1)$			
x_2	y_2	$y(x_1,x_2)$	$y(x_0,x_1,x_2)$		
x_3	y_3	$y(x_2,x_3)$	$y(x_1,x_2,x_3)$	$y(x_0,x_1,x_2,x_3)$	
x_4	y_4	$y(x_3,x_4)$	$y(x_2,x_3,x_4)$	$y(x_1,x_2,x_3,x_4)$	$y(x_0,x_1,\ldots,x_4)$

Для построения табл. 2 (нахождения коэффициентов полинома) необходимо вычислить $n+(n-1)+\ldots+1=\frac{1}{2}n(n+1)$ разделённых разностей; для вычисления каждой из них требуется два вычитания и одно деление, итого $\frac{3}{2}n(n+1)$ арифметических операций. Как и в случае формулы Лагранжа, число «подготовительных» операций растёт как $\mathcal{O}(n^2)$, хотя численный коэффициент для формулы Ньютона получился приблизительно на четверть меньше. Заметим, однако, что эти операции выполняются однократно, так что временем на их выполнение в большинстве задач можно пренебречь.

Заметим также, что нахождение полиномиальных коэффициентов в формуле Ньютона (53) возможно и без введения понятия разделённых разностей. Действительно, запишем интерполяционный полином Ньютона по схеме Горнера с неопределёнными коэффициентами $a_0, a_1, \ldots, a_n$:

$$P(x) = a_0 + (x-x_0)\left[a_1 + \ldots + (x-x_{n-2})\left[a_{n-1} + (x-x_{n-1})\cdot a_n\right]\ldots\right].$$

Требуя совпадения значений полинмома с интерполируемой функцией в узлах сетки $P(x_i) = y(x_i) \equiv y_i$, получим систему линейных алгебраических уравнений на коэффициенты $a_0, a_1, \ldots, a_n$:

$$\begin{array}{lcl}
y_0 & = & a_0 \\
y_1 & = & a_0 + (x_1 - x_0)a_1 \\
y_2 & = & a_0 + (x_2 - x_0)\left[a_1 + (x_2 - x_1)a_2\right] \\
 & \ldots & \\
y_n & = & a_0 + (x_n - x_0)\left[a_1 + \ldots + (x_n - x_{n-2})\left[a_{n-1} + \right.\right. \\
 & & \qquad\qquad \left.\left. + (x_n - x_{n-1})a_n\right]\ldots\right]
\end{array}$$

Из первого уравнения имеем $a_0 = y_0$; подставляя a_0 во второе, находим $a_1 = \frac{y_1 - a_0}{x_1 - x_0}$; далее подставляем a_0, a_1 в третье уравнение и т. д.

5.3. Погрешность интерполяции

Оценим погрешность интерполяции полиномами в случае, когда интерполируемая функция y имеет $n+1$ непрерывную производную. Поскольку погрешность интерполяции обращается в ноль во всех узлах сетки $x_0, x_1, \ldots, x_n$, её можно представить в следующем виде:

$$y(x) - P(x) = \omega(x) \cdot r(x), \qquad \omega(x) = \prod_{j=0}^{n} (x - x_j). \tag{55}$$

Введём вспомогательную функцию

$$q(x) \equiv y(x) - P(x) - \omega(x) r(\lambda), \tag{56}$$

где λ играет роль параметра. Очевидно, что функция $q(x)$ обращается в ноль в каждом узле сетки $x_0, x_1, \ldots, x_n$ и в точке $x = \lambda$. Поскольку по нашему предположению $y(x)$ имеет n+1 непрерывную производную, $q(x)$ также должна быть $n+1$ раз непрерывно дифференцируемой, причём для $(n+1)$-й производной в силу определения $q(x)$ можно записать

$$q^{(n+1)}(x) = y^{(n+1)}(x) - (n+1)!\, r(\lambda). \tag{57}$$

Из курса анализа известно, что между двумя нулями гладкой функции лежит ноль её производной. Последовательно применяя это правило, можно сделать вывод, что в промежутке между крайними из $n+2$ нулей функции $q(x)$ лежит ноль её $(n+1)$-й производной. Обозначим точку, в которой зануляется производная $q^{(n+1)}(x)$ за $x^* = x^*(\lambda)$ (она будет являться некоторой функцией от параметра λ). Приравнивая (57) к нулю, получим

$$r(\lambda) = \frac{y^{(n+1)}(x^*(\lambda))}{(n+1)!}. \tag{58}$$

Поскольку параметр λ изначально был выбран произвольно, равенство (58) позволяет сделать оценку сверху для погрешности полиномиальной интерполяции (55):

$$|y(x) - P(x)| \leq \frac{\max |y^{(n+1)}(x)|}{(n+1)!} \cdot |\omega(x)|. \tag{59}$$

Для примера на рис. 6 (*а*) показан график зависимости разности $y(x) - P(x)$ при интерполяции функции $y(x) = \sin(x)$ полиномами степени $n = 5$ на промежутке от 0 до $\pi/2$, а также зависимость максимальной ошибки от степени n (рис. 6 (*б*)).

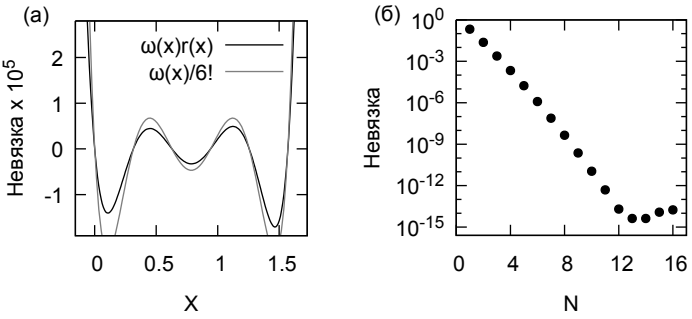


Рис. 6. Зависимость погрешности от x (а) и степени полинома N (б)

На графике видно, что погрешность зануляется в шести точках $x_0, x_1, \ldots, x_5$. Ближе к краям отрезка $[0, \pi/2]$, на котором производится интерполяция, ошибка больше, чем в середине. За пределами отрезка ошибка резко возрастает — погрешность экстраполяции намного превышает погрешность интерполяции. В целом погрешность близка к мажорантной оценке (59) (показана на рис. 6 (*а*) серой линией).

На рис. 6 (*б*) приведена также зависимость максимальной (по x при фиксированном n) ошибки как функции степени полинома n (по оси y использован логарифмический масштаб). Видно, что ошибка быстро убывает до $n = 13$, после чего вновь начинает возрастать из-за погрешности вычисления разделённых разностей (см. п. 5.4).

5.4. Численное дифференцирование

Покажем, как разделённые разности, введённые на с. 73, могут быть использованы для аппроксимации значений производных функции. Пусть $y(x)$ — достаточное число раз гладкая функция, для которой справедливо разложение

$$y(x) = \sum_{k=0}^{n} y^{(k)}(a) \frac{(x-a)^k}{k!} + \mathcal{O}\left((x-a)^{n+1}\right) \tag{60}$$

Вычислим разделённую разность n-го порядка $y(x_0, x_1, \ldots, x_n)$, переписав (60) в виде $y(x) = P_n(x) + \mathcal{O}(h^{n+1})$, где $P_n(x)$ — полином степени n, $h \leq \max |x_i - x_j|/n$ — характерный шаг сетки $\{x_0, \ldots, x_n\}$:

$$y(x_0, \ldots, x_n) = P_n(x_0, \ldots, x_n) + \mathcal{O}(h). \tag{61}$$

Для вычисления разделённой разности от величины $\mathcal{O}(h^{n+1})$ мы воспользовались формулой (49) на с. 74, получив $\mathcal{O}(h)$.

Входящая в (61) разделённая разность n-го порядка для полинома степени n равна n-й производной $(d^n/dx^n)P_n$ этого полинома, делённой на $n!$ (в этом легко убедиться, продифференцировав n раз выражение (54) на с. 75). Заметим, что тот факт, что выражение (54) было получено нами для интерполяционного полинома, абсолютно несущественен: в силу доказанной на с. 72 теоремы о существовании и единственности, любой полином P_n можно считать интерполяционным по отношению к самому себе. Учитывая, что $(d^n/dx^n)P_n = (d^n/dx^n)y$ в силу определения P_n (60), получаем из (61) окончательно:

$$y^{(n)}(a) = n!\, y(x_0, \ldots, x_n) + \mathcal{O}(h). \tag{62}$$

Выбор точки a достаточно произволен — она может совпадать с любым из узлов сетки x_i или находиться на расстоянии $\mathcal{O}(h)$ от него. Порядок точности аппроксимации можно повысить, используя формулы с бóльшим числом точек. (В общем случае произвольной неравномерной сетки порядок точности по h равен разности числа узлов и порядка производной [2, гл. 3].)

Разделённую разность порядка k, вычисленную на равномерной сетке и умноженную на $k!$, называют *конечной разностью*. Использование равномерной сетки обычно позволяет упростить формулы и повысить порядок точности аппроксимации. Для примера рассмотрим конечную разность второго порядка:

$$2y(h, 0, -h) = \frac{y(-h) - 2y(0) + y(h)}{h^2} = y''(0) + \frac{h^2}{12} y^{(\mathrm{iv})}(0) + \ldots \tag{63}$$

Конечная разность второго порядка (63) с точностью до $\mathcal{O}(h^2)$ аппроксимирует значение второй производной функции в среднем узле. Аналогично, конечная разность третьего порядка

$$3!\, y(0, h, 2h, 3h) = \frac{1}{h^3}\Big(y(3h) - 3y(2h) + 3y(h) - y(0)\Big)$$

аппроксимирует $y'''(3h/2)$ с точностью $\approx h^2 y^{(\mathrm{v})}/8 = \mathcal{O}(h^2)$ и т. д.

Заметим, что конечные разности нечётного порядка аппроксимируют значения соответствующих производных со вторым порядком точности в средней точке между центральными узлами сетки, тогда как аппроксимация в узлах имеет лишь первый порядок по h. Как следствие, для аппроксимации производных нечётного порядка в узлах x_i, конечные разности выгодно вычислять на сетке с удвоенным шагом $2h$.

Хотя это и увеличивает численный коэффициент в остаточном члене в 2^n раз, но обеспечивает более высокий порядок точности по h.

Несмотря на то, что из выражения (62) следует $y(x_0, \ldots, x_n) \to y^{(n)}$ в пределе $\max |x_i - x_j| \to 0$, вычисление производных, особенно высоких порядков, может представлять существенные сложности. Действительно, пусть требуется аппроксимировать величину $y^{(n)}$, вычисляя конечную разность $n!\, y(x_0, \ldots, x_n)$, $x_j = x_0 + jh$. Изменим входные данные — значения $y_j \equiv y(x_j)$ — на малую величину $\delta \cdot \exp(iqx_j)$, $|\delta| \ll |y_j|$. При этом конечная разность, вычисленная по изменённым данным, очевидно, будет аппроксимировать величину $(d^n/dx^n)[y(x) + \delta \exp(iqx)] = y^{(n)} + (iq)^n \delta \exp(iqx)$. При уменьшении шага сетки h (в пределе $h \to 0$) входные данные могут содержать всё более высокочастотные добавки (шумы, погрешности) — сверху их спектр ограничен частотой Найквиста $q_{\max} = \pi/h$. Как следствие, погрешность численного дифференцирования $q^n \delta$ резко возрастает при уменьшении шага сетки h и увеличении порядка производной n. В частности, в пределе $h \to 0$ разделённые разности *не аппроксимируют* производные при наличии погрешности во входных данных. В этой связи численное дифференцирование относят к классу *некорректных задач*.

На практике при численном дифференцировании необходимо помнить, что погрешность результата складывается из двух составляющих, $R = R_1 + R_2$. Первая связана с отбрасыванием высших производных в ряде Тейлора и убывает вместе с шагом сетки, $R_1 = \mathcal{O}(h^p)$, где p — порядок точности формулы дифференцирования. Вторая составляющая ошибки связана с наличием погрешности вычисления функции $\delta y \geq \varepsilon/2$, где $\varepsilon/2$ — ошибка округления. Итого, имеем $R = \mathcal{O}(h^p) + \mathcal{O}(h^{-n})$, откуда следует, что при численном дифференцировании существует оптимальный шаг сетки h, обеспечивающий минимальную погрешность. Более подробно о методах регуляризации дифференцирования можно прочитать в [2, гл. 3.6].

5.5. Другие виды интерполяции

Полиномиальная интерполяция, рассмотренная в данной главе, является достаточно простым и эффективным способом приближённого вычисления функций. Решение задачи может быть легко выписано в явном виде (54), что позволяет создавать на его основе квадратурные формулы и методы решения обыкновенных дифференциальных уравнений высокого порядка точности.

Вместе с тем необходимо отметить, что использование полиномиальной интерполяции на равномерной сетке в некоторых случаях даёт

неудовлетворительные результаты даже на гладких функциях. Ярким примером служит *феномен Рунге*, который мы рекомендуем читателю рассмотреть самостоятельно (см. упражнение 7 на с. 85). В этой связи в заключение данной главы будет дан краткий обзор альтернативных решений задачи об интерполяции.

Один из очевидных способов повышения точности полиномиальной интерполяции состоит в использовании неравномерных сеток. По сути, аналогичный подход используется для построения квадратурных формул Гаусса, выгодно отличающихся от формул Ньютона-Котеса более высоким порядком точности и устойчивостью. Из оценки погрешности интерполяции (59) следует, что оптимальный выбор узлов сетки должен обеспечивать минимизацию полинома $\omega(x)$ (55) на промежутке интерполяции $[x_0, x_n]$. Решение данной задачи известно [A10] и состоит в том, что $\omega(x)$ должен являться полиномом Чебышёва $\cos((n+1)\arccos\xi)$, $\xi \in [0,1]$. Из определения $\omega(x)$ (55) следует, что для этого следует выбрать в качестве узлов сетки нули многочлена Чебышёва:

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\frac{\pi(2k+1)}{2(n+1)}, \quad k = 0, 1, \ldots, n. \tag{64}$$

Заметим, что использование данного подхода не всегда возможно. Например, интерполируемая функция может быть получена в результате численных расчётов, так что её значения могут быть известны лишь в узлах определённой сетки (чаще всего равномерной).

В случае, если в узлах сетки известны не только значения интерполируемой функции $y(x_k)$, но и значения её производных вплоть до m-го порядка, можно наложить дополнительные условия на интерполяционный полином, потребовав равенства его производных соответствующим значениям производных интерполируемой функции. Данный способ интерполяции называют *эрмитовым* [A10]. Интерполяционные многочлены Эрмита обеспечивают тот же порядок точности, что и полиномы Ньютона, однако численное значение погрешности для многочлена Эрмита будет меньше. «Минусами» эрмитовой интерполяции является необходимость вычисления производных интерполируемой функции, а также достаточно сложные формулы для вычисления полиномиальных коэффициентов [A10]. Кроме того, при достаточно большом числе узлов сетки полиномиальная интерполяция может давать неудовлетворительные результаты из-за неустойчивости, связанной с вычислением многочленов высокой степени.

Ещё одним способом повышения точности интерполяции является построение «локальных» интерполяционных полиномов, или *кусочно-*

полиномиальная интерполяция. Так, если значения интерполируемой функции $y(x)$ известны на сетке $x_0, \ldots, x_N$, можно построить интерполяционный полином $P_{n,k}(x)$ степени n, значения которого будут совпадать со значениями функции y в точках $x_k, x_{k+1}, \ldots, x_{k+n}$ (при $0 \le k \le N-n$) и использовать его для интерполяции $y(x)$ на отрезке $x_k \le x \le x_{k+n}$. Область $x_0 \le x \le x_N$ может быть разбита на отрезки, на каждом из которых будет использоваться свой «локальный» интерполяционный полином. Данное решение широко используется, например, при построении обобщённых квадратурных формул (см. главу 4). Погрешность интерполяции можно уменьшить, если использовать каждый из интерполяционных полиномов $P_{n,k}(x)$ не на всем отрезке $[x_k, x_{k+n}]$, но лишь в средней его части, где погрешность интерполяции минимальна (см. рис. 6 (a)). Однако данный подход не является универсальным — одним из факторов, ограничивающих его применимость, является отсутствие непрерывности производных построенной указанным способом интерполирующей функции.

Избавиться от указанного недостатка можно, накладывая дополнительные условия на сшивку первых $(n-p)$ производных кусочно-полиномиальной функции в узлах сетки. Полученную в результате этого функцию называют *сплайном*[23] *степени* n *дефекта* p. По сравнению с полиномами, интерполяция сплайнами S_{np} выгодно отличается гарантированной сходимостью и устойчивостью вычислений за счёт использования «локальных» полиномов невысокой степени, в то время как общее число узлов сетки может быть очень большим, что позволяет интерполировать осциллирующие функции с большим количеством экстремумов.

Наиболее часто употребляются два частных случая сплайнов.

- S_{31} — *интерполяционный кубический сплайн* ($n = 3$, $p = 1$), часто называемый просто *сплайном*, имеет непрерывную первую и вторую производные; значения сплайна в узлах сетки совпадают со значениями интерполируемой функции.

- S_{11} — *линейный сплайн* ($n = 1$, $p = 1$) представляет собой кусочно-линейную функцию, значения которой в узлах сетки совпадают со значениями интерполируемой функции.

Построение кусочно-линейной функции S_{11} является тривиальной задачей, решение которой ничем не отличается от построения не свя-

[23] От англ. *spline* — гибкая линейка, используемая в черчении и инженерных расчётах для проведения гладких кривых через заданные точки.

занных друг с другом интерполяционных полиномов Ньютона первой степени.

Задача о построении кубических сплайнов S_{31} также не представляет большой сложности. Для её решения необходимо записать на каждом интервале сетки $x_0, x_1, \ldots, x_N$ полином третьей степени с неопределёнными коэффициентами:

$$g(x) = \frac{(x_i - x)^3}{6h_i} m_{i-1} + \frac{(x - x_{i-1})^3}{6h_i} m_i + A_i \frac{x_i - x}{h_i} + B_i \frac{x - x_{i-1}}{h_i}, \quad (65)$$

где $x \in [x_{i-1}, x_i]$, $h_i = x_i - x_{i-1}$, а m_i — значение второй производной сплайна в i-м узле сетки. Условия интерполяции $g(x_k) = f_k$ дают на каждом отрезке $\frac{1}{6}h_i^2 m_{i-1} + A_i = f_{i-1}$, $\frac{1}{6}h_i^2 m_i + B_i = f_i$. Условия сшивки первой производной в узлах сетки приводят к уравнениям ($i = 1, \ldots, N-1$):

$$\frac{h_i}{6} m_{i-1} + \frac{h_i + h_{i+1}}{3} m_i + \frac{h_{i+1}}{6} m_{i+1} = \frac{f_{i-1}}{h_i} - \left(\frac{1}{h} + \frac{1}{h_{i+1}}\right) f_i + \frac{f_{i+1}}{h_{i+1}}. \quad (66)$$

Полученная система является линейной, её матрица трёхдиагональна, что позволяет эффективно решить (66) методом прогонки. Для замыкания системы необходимо задать вторые производные $g''(x)$ на краях сетки: m_0 и m_N. В отсутствие дополнительных соображений их обычно полагают равными нулю. Отметим, что кубические сплайны могут быть найдены вариационным способом минимизацией функционала

$$\Phi(u) = \int_{x_0}^{x_N} (u''(x))^2 \, dx, \quad u(x_i) = f_i, \; u''(x_0) = u''(x_N) = 0. \quad (67)$$

Развивая этот подход, можно получить решение задачи *аппроксимации* набора данных, известных из эксперимента. Действительно, полагая, что сплайн-функция должна удовлетворять условиям $u(x_i) = f_i$ лишь приближённо, с учетом некоторой известной погрешности измерений, можно модифицировать функционал (67):

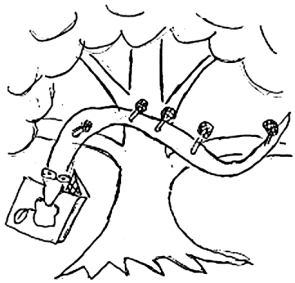
$$\Phi_1(u) = \int_{x_0}^{x_N} (u''(x))^2 \, dx + \sum_{i=0}^{N} p_i \cdot (u(x_i) - f_i)^2, \quad p_i > 0, \quad (68)$$

где коэффициенты p_i связаны с погрешностью измерения экспериментальных данных f_i (чем выше точность измерений f_i, тем больше ко-

эффициент p_i[24]). Экстремум функционала (68) достигается на кубических сплайнах S_{31}, для определения его коэффициентов необходимо решить систему линейных уравнений с пятидиагональной матрицей. Построенные таким образом сплайны называют *сглаживающими* и широко используют для аппроксимации экспериментальных данных. В частности, для построения сглаживающих сплайнов в gnuplot предусмотрен режим `smooth acsplines` команды `plot`, для построения интерполяционных сплайнов — `smooth csplines`.

Значительно более подробное решение задачи о построении кубических сплайнов можно найти в монографиях [2] и [A10]; готовый к использованию программный код на языке Си приведён в [4].

Сообразно характеру интерполируемых функций, в некоторых задачах целесообразно использовать в качестве базисных функций для линейной интерполяции не алгебраические полиномы, а, например, тригонометрические функции или экспоненты, либо интерполировать $y(x)$ с помощью дробно-линейных функций [4], [A10]. В случае, если функция $y(x)$ вычислена на сетке с достаточно большим шагом так, что $y(x)$ сильно меняется между узлами сетки, целесообразно использовать *нелинейную* интерполяцию, подбирая преобразование $\eta = \eta(y)$, $\xi = \xi(x)$ так, чтобы в новых переменных зависимость $\eta(\xi)$ мало отличалась от линейной на протяжении нескольких шагов сетки. Конкретный вид нелинейного преобразования следует выбирать из физических соображений [2].



[24] В частности, в случае равномерного стремления $p_i \to \infty$, минимум (68) достигается на интерполяционных кубических сплайнах, доставляющих экстремум (67).

Упражнения

1. Получить квадратурные формулы трапеций (24) и Симпсона (31), интегрируя интерполяционные полиномы первой и второй степени, построенные на равномерной сетке.

2. Написать на языке Си функцию для вычисления разделённых разностей произвольного порядка. Используя равномерную сетку, исследовать точность, с которой разделённая разность порядка k аппроксимирует величину $k!y^{(k)}$ при $k = 1, 3, 5, 7$ для $y(x) = e^x$ в точке $x = 0$ в зависимости от величины шага сетки h.

3. Получить априорную оценку оптимального шага сетки h_{opt} для численного дифференцирования. Наложить априорные оценки погрешности численного дифференцирования и величины оптимального шага на график из упражнения 2.

Для выполнения следующих упражнений необходимо написать программный код для интерполяции полиномами Ньютона. В программе следует предусмотреть возможность простой замены интерполируемой функции и промежутка интерполирования, числа узлов сетки (степени полинома), а также возможность использования неравномерной сетки. Для отладки программы удобно дублировать расчёты для полиномов невысокой степени ($n = 2, 3, 4$) в Microsoft Excel.

4. Построить интерполяционный полином Ньютона для функции $y(x) = \operatorname{tg} x$ на промежутке $[0, 1]$, используя $n = 2, 3, 5, 10$ точек. Построить график зависимости ошибки интерполяции от x, сравнить с (59). Вывести на экран значение максимума невязки для разных степеней полинома. Построить график зависимости невязки от x в более широких пределах (например, от $-0{,}2$ до $1{,}2$), сравнить погрешность интерполяции и экстраполяции.

5. То же для функций $\sqrt[3]{\cos^2 x}$ и $\sqrt[3]{\sin^2 x}$. Сравните погрешности интерполяции для указанных функций, объясните результат.

6. То же для функции $\cos(x^4)$ на промежутке от $[-\pi, \pi]$. Как изменяется максимальное значение невязки на интервале интерполяции при увеличении числа узлов сетки (степени полинома)?

7. То же для *функции Рунге* $y(x) = 1/(1 + 25x^2)$ на промежутке $[-1, 1]$. Как изменяется максимальное значение невязки на интервале интерполяции при увеличении числа узлов сетки?

8. Для интерполяции функции Рунге из упражнения 7 использовать неравномерную сетку, узлы которой совпадают с нулями многочлена Чебышёва (64) соответствующей степени.

9. Написать программный код для вычисления функции Бесселя $J_0(x)$ с абсолютной погрешностью не выше 10^{-10} с использованием интегрального представления (69). Построить интерполяционный полином Ньютона P_n степени n для $J_0(x)$ на промежутке $[0, 20]$, используя равномерную сетку. Вычислить невязку δ:

$$\delta = \min_n \max_x |J_0(x) - P_n(x)|, \quad J_0(x) = \frac{1}{\pi} \int\limits_0^\pi \cos\big(x \sin\varphi\big)\, d\varphi. \quad (69)$$

6. Решение обыкновенных дифференциальных уравнений

Пусть требуется численно найти решение задачи Коши для обыкновенного дифференциального уравнения (ОДУ)

$$\frac{du}{dx} = f\big(x, u(x)\big), \qquad a \le x \le b, \qquad u(a) = u_a. \qquad (70)$$

Для простоты в рассмотрении ниже мы будем ограничиваться случаем одного уравнения, имея при этом в виду, что многие методы элементарно обобщаются на случай системы

$$\frac{d\mathbf{u}}{dx} = \mathbf{f}(x, \mathbf{u}).$$

Условимся использовать $y(x)$ для обозначения численного решения (70), а $u(x)$ — для точного решения той же задачи. Численное решение задачи Коши (70) будем искать при $a \le x \le b$ на сетке (в общем случае неравномерной) $x_0, x_1, x_2, \ldots$ как набор значений $y_i \equiv y(x_i)$. Ниже мы рассмотрим несколько методов поиска численного решения задачи (70).

6.1. Метод Эйлера (схема ломаных)

Предполагая, что функция $f(x, u)$ в правой части уравнения (70) имеет достаточное число непрерывных производных, можем записать

для производных решения $u(x)$:

$$\begin{aligned} u' &= f(x,u), \\ u'' &= f_x + f_u f, \\ u''' &= f_{xx} + 2f_{xu}f + f_{uu}f^2 + f_u f_x + f_u^2 f, \\ \dots \end{aligned} \tag{71}$$

Раскладывая решение $u(x)$ в ряд Тейлора, получаем:

$$u(x+h) = \sum_{k=0}^{M} u^{(k)}(x)\frac{h^k}{k!} \quad + \quad \mathcal{O}\left(h^{M+1}\right). \tag{72}$$

Значения производных $u^{(k)}(x)$, входящие в (72), следует подставлять из (71). В простейшем случае, ограничиваясь только первым членом разложения ($M = 1$), имеем:

$$u(x+h) = u(x) + h\, f(x,u) + \mathcal{O}(h^2).$$

Как следствие, получаем *схему ломаных*, предложенную Леонардом Эйлером в 1768 г и потому известную также как *метод Эйлера*:

$$y_{n+1} = y_n + h_n\, f(x, y_n), \qquad h_n \equiv x_{n+1} - x_n. \tag{73}$$

Геометрический смысл метода Эйлера иллюстрирует рис. 7 (a): на каждом шаге происходит смещение из текущей точки (x_n, y_n) вдоль касательной к интегральной линии решения, проходящей через данную точку. Траектория движения в плоскости (x, y) получается ломаной линией, что и отражает название метода.

Ввиду крайне низкой точности метод Эйлера редко применяется в численных расчётах. Вместе с тем, схема ломаных является наиболее простой, что позволяет на её примере разобраться в общих подходах к исследованию сходимости и точности численных методов.

Если в разложении (72) удержать не только первый член, но также и члены более высоких порядков, то, подставив выражения для производных $u^{(k)}(x)$ из (71), можно было бы получить численные схемы более высокого порядка точности. Однако такой подход нечасто используется на практике ввиду громоздкости получающихся при этом выражений. Кроме того, если правая часть $f(x,u)$ уравнения (70) известна лишь приближённо, то вычисление её производных может приводить к резкому росту погрешности вычислений.

Исследуем сходимость численного решения задачи Коши $y(x)$, построенного по схеме ломаных (73) к точному решению $u(x)$, полагая

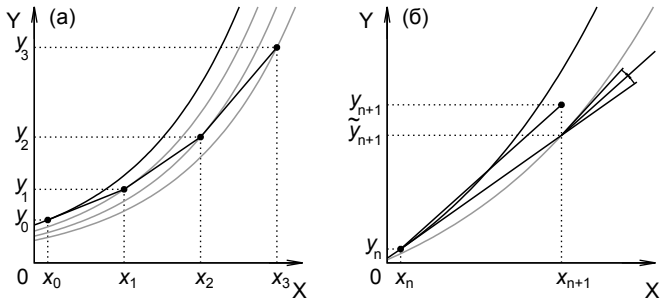


Рис. 7. (а) схема ломаных, (б) исправленный метод Эйлера

для этого $u(x)$ известным. Для этого вычтем численное решение (73) из разложения точного решения (72), положив $x = x_n$ и $h = x_{n+1} - x_n$. Вводя обозначение R_n для погрешности на n-м шаге ($R_n \equiv u_n - y_n$), где $u_n \equiv u(x_n)$, имеем:

$$R_{n+1} = R_n + (f(x_n, u_n) - f(x_n, y_n))\ h_n + \frac{1}{2}u''(x_n)h_n^2 + \mathcal{O}(h^3).$$

Подставляя сюда $f(x_n, y_n) = f(x_n, u_n) + \frac{\partial f}{\partial u}(x_n, u_n) \cdot (y_n - u_n) + \mathcal{O}(R_n^2)$:

$$R_{n+1} = R_n \cdot (1 + f_u h_n) + \frac{h_n^2}{2}u_n'' + \mathcal{O}(R^2, h^3).$$

Таким образом, погрешность численного решения домножается на каждом шаге на величину $(1 + hf_u)$ и к полученному произведению прибавляется величина $\frac{1}{2}h^2u'' + \mathcal{O}(R^2, h^3)$. Применяя полученную формулу рекурсивно, имеем для погрешности на n-м шаге:

$$R_n \approx R_0 \prod_{k=0}^{n-1}\left(1 + \frac{\partial f}{\partial u}h_k\right) + \frac{1}{2}\sum_{m=0}^{n-1} h_m^2 u_m'' \prod_{k=m+1}^{n-2}\left(1 + \frac{\partial f}{\partial u}h_k\right).$$

С точностью до членов второго порядка малости по h можно заменить $1 + \frac{\partial f}{\partial u}h \approx \exp(f_u h)$, тогда:

$$R_n \approx R_0\ \exp\left[\sum_{k=0}^{n-1}\frac{\partial f}{\partial u}h_k\right] + \frac{1}{2}\sum_{m=0}^{n-1} h_m^2 u_m'' \exp\left[\sum_{k=m+1}^{n-2}\frac{\partial f}{\partial u}h_k\right].$$

Заменяя суммы в полученном выражении интегралами (совершая при этом ошибку, равную $\mathcal{O}(h^2)$) и меняя в пределах той же ошибки пределы интегрирования на величину порядка h, имеем

$$R_n \approx R_0 \cdot \exp\left[\int\limits_{x_0}^{x_n} f_u d\xi\right] + \frac{1}{2}\int\limits_{x_0}^{x_n} h(\zeta)u''(\zeta) \cdot \exp\left[\int\limits_{\zeta}^{x_n} f_u d\mu\right] d\zeta. \qquad (74)$$

Здесь $h(x)$ — непрерывная функция, дающая в каждом узле сетки x_n значение шага сетки $h(x_n) = h_n$.

Проанализируем полученный результат. Первое слагаемое связано с наличием погрешности начального условия $u_0 = u(x_0)$ задачи Коши (70). Положим данный член равным нулю ($R_0 = 0$), считая, что начальное значение задано точно.

Второе слагаемое в (74) может быть легко оценено сверху путём замены входящих в него величин модулями их максимальных значений в области $a \le x \le b$. Проделав это и вынеся из-под интеграла $h = \max_x h(x)$, получим, что в пределе $h \to 0$ приближённое решение $y(x)$ сходится к точному решению $u(x)$ равномерно на ограниченном отрезке $a \le x \le b$ с *первым порядком точности* по шагу сетки h.

Экспоненциальный член, входящий в (74), характеризует расхождение интегральных кривых. Больша́я величина данного члена означает плохую обусловленность задачи Коши.

Определение. Задача называется *хорошо обусловленной*, если малое изменение начальных условий приводит к малому изменению интегральных кривых. В противном случае задача называется *плохо обусловленной.*

В качестве примера рассмотрим задачу Коши $u'(x) = u - x$, $u(0) = 1$, $0 \le x \le 100$. Общее решение $u(x) = 1 + x + ce^x$. Решение, удовлетворяющее задаче Коши $u(0) = 1$, получается при $c = 0$: $u(x) = 1 + x$, $u(100) = 101$. Изменим начальное условие в задаче Коши на машинное эпсилон: $u(0) = 1+\varepsilon = 1+2^{-52}$. Тогда получим $c = \varepsilon$, $u(100) \approx 5{,}97 \cdot 10^{27}$. Изменение начального условия в шестнадцатом десятичном знаке после запятой приводит в данном примере к изменению ответа более чем в 10^{25} раз. Очевидно, такая задача является плохо обусловленной, и мы не сможем построить для неё удовлетворительного численного решения, используя стандартные типы данных.

Заметим, что полученное выше выражение (74) — *априорная оценка погрешности* — достаточно редко используется на практике ввиду своей сложности. Как правило, вместо этого в расчётах используется *апостериорная* оценка, основанная на проведении расчётов на сетках

с разным (отличающимся в заданное число r раз) шагом h и последующим использованием формулы Рунге (см. п. 4.5). Для использования апостериорных оценок погрешности желательно знать *порядок точности метода*, т. е. показатель степени шага сетки h в формуле остаточного члена. Сейчас мы покажем, как эта значительно более простая задача может быть решена на примере метода Эйлера, но вначале дадим несколько определений.

Определение. Невязкой, или *погрешностью* аппроксимации разностного уравнения на решении исходного дифференциального уравнения ψ_n, называют результат подстановки точного решения $u = u(x)$ в разностное уравнение.

Определение. Говорят, что *разностный метод аппроксимирует дифференциальное уравнение*, если $\psi_n \to 0$ при $h \to 0$, где ψ_n — невязка, h — шаг сетки. Говорят, что *разностный метод имеет p-й порядок аппроксимации*, если $\psi_n = \mathcal{O}(h^p)$.

Например, для метода Эйлера, который может быть записан в виде

$$\frac{y_{n+1} - y_n}{h_n} - f(x_n, y_n) = 0,$$

невязка равна

$$\psi_n = \frac{u_{n+1} - u_n}{h_n} - f(x_n, u(x_n)).$$

Подставляя разложение $u_{n+1} = u_n + u'_n h + \mathcal{O}(h^2)$ и заменяя u'_n на $f(x, u_n)$, имеем

$$\psi_n = \frac{u_n + u'_n h + \mathcal{O}(h^2) - u_n}{h} - f(x_n, u(x_n)) = \mathcal{O}(h).$$

Таким образом, мы пришли к тому же выводу, что и на основе анализа асимптотической оценки погрешности (74), а именно, что метод Эйлера (схема ломаных) имеет первый порядок точности.

6.2. Исправленный и модифицированный методы Эйлера

Рассмотрим две модификации метода Эйлера (73), позволяющие повысить точность расчётов, уменьшив погрешность до второго порядка по величине шага сетки, $\mathcal{O}(h^2)$. Прежде чем переходить к последовательному рассмотрению этих методов, попытаемся понять их основную идею, рассмотрев аналогию с методами численного интегрирования.

Для этого исследуем частный случай задачи (70), когда функция f не зависит от u, т. е. $f(x,u) \equiv g(x)$:

$$\frac{du}{dx} = g(x), \qquad a \le x \le b, \qquad u(a) = u_a. \tag{75}$$

Точное решение задачи Коши (75) может быть выписано в явном виде:

$$u(x) = \int\limits_a^x g(\xi)\, d\xi, \quad a \le x \le b. \tag{76}$$

Рассмотренный выше метод Эйлера (73) является в данном случае не чем иным, как методом левых прямоугольников для вычисления значения интеграла (76).

Как было показано в п. 4.2, точность вычисления интегралов методом прямоугольников можно значительно (на порядок по шагу сетки h) повысить, используя полусумму площади левых и правых прямоугольников, т. е. вместо величины $h_n\, f(x_n)$ вычисляя площадь трапеции $[f(x_n) + f(x_n + h_n)]\, h_n/2$. Тот же порядок точности ($R_\Sigma = \mathcal{O}(h^2)$) можно получить, аппроксимируя интеграл площадью прямоугольников с высотой, равной значению функции посередине между соседними узлами сетки, $h_n\, f(x_n + \frac{1}{2}h_n)$. Эти же идеи лежат в основе *исправленного* и *модифицированного* методов Эйлера, которые мы сейчас рассмотрим.

Исправленный метод Эйлера состоит в построении численного решения y_n задачи Коши (70) по схеме

$$y_{n+1} = y_n + \frac{h}{2}\Big[f(x_n, y_n) \,+\, f\Big(x_{n+1},\, y_n + h\, f(x_n, y_n)\Big)\Big]. \tag{77}$$

Геометрический смысл разностной схемы (77) представлен на рис. 7 (*б*). Из точки (x_n, y_n) выполняется шаг h методом Эйлера (73) по касательной к интегральной линии уравнения (70), в результате чего мы приходим в точку $(x_n + h, y_n + h\, f(x_n, y_n)) \equiv (x_{n+1}, \tilde{y}_{n+1})$, см. рис. 7 (*б*). В полученной точке $(x_{n+1}, \tilde{y}_{n+1})$ вычисляется правая часть уравнения (70) и находится среднее арифметическое между $f(x_n, y_n)$ и $f(x_{n+1}, \tilde{y}_{n+1})$ аналогично тому, как мы вычисляли полусумму значений функции на правом и левом концах отрезка при вычислении интеграла методом трапеций. После этого делается шаг h из точки (x_n, y_n) вдоль прямой, имеющей наклон $\frac{1}{2}[f(x_n, y_n) + f(x_{n+1}, \tilde{y}_{n+1})]$, что приводит нас к следующему $(n+1)$-му узлу сетки, в точку (x_{n+1}, y_{n+1}). На рис. 7 (*б*) видно, что ошибка, допускаемая на шаге численного интегрирования

исправленным методом Эйлера (77), значительно меньше аналогичной ошибки в схеме ломаных (73).

Определим порядок точности исправленного метода Эйлера (77). Для этого вычислим невязку ψ_n, подставляя точное решение задачи (70) в численную схему исправленного метода Эйлера (77), предварительно переписав её в виде, соответствующем дифференциальному уравнению (70):

$$\frac{y_{n+1} - y_n}{h} = \frac{1}{2}\Big[f(x_n, y_n) \,+\, f\Big(x_{n+1},\, y_n + h\, f(x_n, y_n)\Big)\Big].$$

Для невязки имеем:

$$\psi_n = \frac{u_{n+1} - u_n}{h} - \frac{1}{2}\Big[f(x_n, u_n) \,+\, f\Big(x_{n+1},\, u_n + h\, f(x_n, u_n)\Big)\Big],$$

где, как и раньше, используется обозначение $u_k \equiv u(x_k)$. Подставляя в полученное равенство разложение $u(x)$ в ряд Тейлора в точке x_n, а также раскладывая $f(x_{n+1},\, u_n + h\, f(x_n, u_n))$, имеем:

$$\psi_n = \frac{u_n + u_n' h + u_n'' \frac{h^2}{2} + \mathcal{O}(h^3) - u_n}{h} - \frac{1}{2}\left[f \,+\, f + f_x h + f_u f h + \mathcal{O}(h^2)\right].$$

В правой части последнего равенства значения функции f и её частные производные вычисляются в точке (x_n, u_n). Используя разложение (71) решения $u(x)$ в ряд Тейлора, получаем окончательно $\psi_n = \mathcal{O}(h^2)$, т. е. исправленный метод Эйлера имеет второй порядок точности.

Перейдем теперь к рассмотрению *модифицированного метода Эйлера*, являющегося аналогом метода центральных прямоугольников для вычисления интегралов (модифицированный метод Эйлера в точности совпадает с методом центральных прямоугольников для задачи (75), см. п. 4.3 на с. 60):

$$y_{n+1} = y_n + h \cdot f\left(x_n + \frac{h}{2},\, y_n + \frac{h}{2} f(x_n, y_n)\right). \tag{78}$$

Геометрический смысл выражения (78) состоит в следующем. Вначале мы делаем шаг $h/2$ из текущей точки (x_n, y_n) методом Эйлера (73), получая точку $(x_{n+1/2}, y_{n+1/2}) = (x_n + \frac{1}{2}h, y_n + \frac{1}{2}hf(x_n, y_n))$. В этой промежуточной точке, лежащей посередине между узлами сетки, мы вычисляем функцию $f(x_{n+1/2}, y_{n+1/2})$ и используем полученное значение вместо $f(x_n, y_n)$ для выполнения шага в методе Эйлера (73).

Определим порядок точности модифицированного метода Эйлера (78). Для невязки ψ_n имеем:

$$\psi_n = \frac{u_{n+1} - u_n}{h} - f\left(x_n + \frac{h}{2},\, u_n + \frac{h}{2} f(x_n, u_n)\right),$$

$$\psi_n = \frac{u_n + u_n' h + u_n'' \frac{h^2}{2} + \mathcal{O}(h^3) - u_n}{h} - \left[f + f_x \frac{h}{2} + f_u \frac{hf}{2} + \mathcal{O}(h^2)\right] = \mathcal{O}(h^2).$$

Следовательно, модифицированный метод Эйлера также имеет второй порядок точности.

6.3. Методы Рунге — Кутты 2-го порядка

Рассмотренные выше исправленный и модифицированный методы Эйлера являются представителями *семейства методов Рунге — Кутты второго порядка.* Покажем, как оба этих метода могут быть получены без помощи аналогий с задачей о вычислении интегралов, и укажем наиболее точную схему в семействе методов Рунге — Кутты второго порядка. Будем искать численное решение y_n задачи Коши (70) в виде

$$y_{n+1} = y_n + h \cdot \Big[a_1 f(x, y) + a_2 f\big(x + hb_1, y + hb_2 f(x, y)\big)\Big], \tag{79}$$

где a_1, a_2, b_1, b_2 — некоторые коэффициенты, $h \equiv x_{n+1} - x_n$. Подставляя в (79) вместо y точное решение $u(x)$, получаем невязку:

$$\psi_n = \frac{u_{n+1} - u_n}{h} - a_1 f(x_n, u_n) - a_2 f(x_n + hb_1, u_n + hb_2 f).$$

Здесь и далее мы обозначаем $f(x_n, u_n)$ за f для краткости. Раскладывая в ряд функцию двух переменных $f(x_n + hb_1, u_n + hb_2 f)$ в точке (x_n, u_n), имеем для невязки:

$$\begin{aligned}\psi_n \;&= \frac{1}{h}\left(u_n + u_n' h + u_n'' \frac{h^2}{2} + u_n''' \frac{h^3}{6} + \mathcal{O}(h^4) - u_n\right) - \\ &- a_1 f - a_2 \left[f + f_x h b_1 + f_u f h b_2\right] - \\ &- a_2 \frac{h^2}{2}\left[f_{xx} b_1^2 + 2 f_{xu} f b_1 b_2 + f_{uu} f^2 b_2^2\right] + \mathcal{O}(h^3).\end{aligned} \tag{80}$$

Подставляя разложение (72) точного решения $u(x)$ с производными (71) и приравнивая нулю коэффициенты при $h^0 f$, $h^1 f_x$ и $h^1 f_u f$, по-

лучаем систему из трёх уравнений:

$$\begin{cases} 1 - a_1 - a_2 & = & 0 \\ \frac{1}{2} - a_2 \cdot b_1 & = & 0 \\ \frac{1}{2} - a_2 \cdot b_2 & = & 0 \end{cases} \quad \Rightarrow \quad \begin{array}{l} a_2 \equiv \alpha, \quad a_1 = 1 - \alpha, \\ b_1 = b_2 = \frac{1}{2\alpha}. \end{array} \tag{81}$$

При выполнении условий (81) невязка (80) занулится в нулевом и первом порядках по шагу сетки h — схема (79) будет иметь второй порядок точности при любых α. С учетом (81) можно переписать схему (79) в виде:

$$y_{n+1} = y_n + h\left[(1-\alpha)f(x,y) + \alpha f\left(x + \frac{h}{2\alpha}, y + \frac{h}{2\alpha}f(x_n, y_n)\right)\right]. \tag{82}$$

Таким образом, мы получили *однопараметрическое семейство методов Рунге — Кутты второго порядка.* В частности, при $\alpha = \frac{1}{2}$ выражение (82) даёт исправленный метод Эйлера (77), при $\alpha = 1$ — модифицированный метод Эйлера (78).

Зададимся теперь вопросом, какое значение параметра α обеспечивает наиболее высокую точность расчётов в методах Рунге — Кутты второго порядка (82)? Для ответа на этот вопрос выпишем невязку (80) при выполнении условий (81):

$$\begin{aligned} \psi_n &= \frac{h^2}{6}\left[f_{xx} + 2f_{xu}f + f_{uu}f^2 + f_u f_x + f_u^2 f\right] - \\ &- \alpha\frac{h^2}{2}\left[\frac{1}{4\alpha^2}f_{xx} + \frac{2}{4\alpha^2}f_{xu}f + \frac{1}{4\alpha^2}f_{uu}f^2\right] + \mathcal{O}(h^3). \end{aligned}$$

Группируя в полученном выражении частные производные и вынося общий множитель, приходим к ответу:

$$\begin{aligned} \psi_n &= \frac{h^2}{6}\left(1 - \frac{3}{4\alpha}\right)\left[f_{xx} + 2f_{xu}f + f_{uu}f^2\right] + \\ &+ \frac{h^2}{6}\left[f_u f_x + f_u^2 f\right] + \mathcal{O}(h^3). \end{aligned} \tag{83}$$

Из полученного выражения (83) видно, что при любых α невязка не обращается тождественно в ноль во втором порядке по шагу сетки h, однако при $\alpha = 3/4$ зануляется первая группа слагаемых, пропорциональная $(f_{xx} + 2f_{xu}f + f_{uu}f^2)$. Таким образом, все схемы семейства (82) имеют второй порядок точности, однако из (83) можно ожидать, что для минимизации численного коэффициента при h^2 в остаточном члене в общем случае следует использовать $\alpha = 3/4$.

В частных случаях $\alpha = \frac{1}{2}$ и $\alpha = 1$ (исправленный и модифицированный метод Эйлера соответственно) коэффициент в круглой скобке перед первым слагаемым в (83) равен $-\frac{1}{2}$ и $\frac{1}{4}$ соответственно, что позволяет ожидать, что модифицированный метод Эйлера в общем случае будет давать несколько меньшую погрешность, чем исправленный метод Эйлера (для квадратурных формул отношение погрешностей метода трапеций к погрешности метода центральных прямоугольников было равно $2 + \mathcal{O}(h)$).

Заметим, что в случае решения задачи Коши (75) с правой частью, зависящей только от x (не зависящей от u), выражение для невязки (83) зануляется во втором порядке точности при выборе $\alpha = \frac{3}{4}$, что позволяет вычислять интегралы по схеме (82) с погрешностью $\mathcal{O}(h^3)$:

$$\int\limits_{x_0}^{x_n} g(\xi)\, d\xi = \sum_{i=0}^{n-1} h_i \left[\frac{1}{4} g(x_i) + \frac{3}{4} g\left(x_i + \frac{2}{3} h_i\right)\right] + \mathcal{O}\left(\max_i h_i^3\right), \qquad (84)$$

где $h_i \equiv x_{i+1} - x_i$. Заметим, что формула (84) занимает промежуточное место между обобщённой формулой трапеций ($R_\Sigma = \mathcal{O}(h^2)$) и формулой Гаусса при $n = 2$ ($R_\Sigma = \mathcal{O}(h^4)$).

6.4. Метод Рунге — Кутты 4-го порядка

Использованный выше подход — суммирование значений функции $f(x, y)$, вычисленных в промежуточных узлах сетки и умноженных на правильным образом подобранные коэффициенты, — позволяет повысить порядок точности интегрирования ОДУ. Рассмотрим без доказательства способ построения численного решения (70) с четвёртым порядком точности.

Метод Рунге — Кутты четвёртого порядка требует вычисления правой части уравнения (70) в четырёх точках на каждом шаге:

$$\begin{aligned}
y_{n+1} &= y_n + \frac{h}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right], \\
k_1 &= f(x_n,\, y_n), \\
k_2 &= f\left(x_n + \frac{h}{2},\, y_n + \frac{h}{2} k_1\right), \\
k_3 &= f\left(x_n + \frac{h}{2},\, y_n + \frac{h}{2} k_2\right), \\
k_4 &= f(x_n + h,\, y_n + h k_3).
\end{aligned} \qquad (85)$$

Можно заметить, что в частном случае (75), т. е. при $f(x,u) = g(x)$, имеем $k_2 = k_3$, и схема Рунге — Кутты 4-го порядка (85) переходит в метод Симпсона для вычисления интегралов.

Схема (85) Рунге — Кутты 4-го порядка реализована в большом количестве программ и пакетов, выполняющих численное интегрирование ОДУ и их систем. Методы Рунге — Кутты более высокого порядка точности практически не употребляются. Пятичленные формулы имеют четвёртый порядок точности; шестичленные имеют шестой порядок точности, но слишком громоздки. Кроме того, высокий порядок точности схем реализуется лишь при наличии у правой части ОДУ $f(x,u)$ непрерывных производных соответствующего порядка.

6.5. Использование адаптивного шага

В различных физических задачах возникают дифференциальные уравнения, правая часть $f(x,u)$ которых может существенно (на порядок величины и более) изменяться на промежутке интегрирования $[a,b]$. Например, при вычислении траектории движения кометы гравитационные силы будут возрастать обратно пропорционально квадрату расстояния при приближении к Солнцу и планетам и убывать при выходе на периферию Солнечной системы. Для повышения скорости расчётов естественно использовать переменный шаг сетки, увеличивая его там, где решение меняется медленно, и уменьшая в тех областях, где правая часть (70) принимает относительно большие значения.

Одной из наиболее простых и притом достаточно универсальных возможностей для автоматической коррекции шага численного интегрирования является пересчёт на сетке с удвоенным шагом. Каждая пара шагов численного интегрирования дублируется при этом шагом двойной величины; из сравнения полученных результатов делается вывод относительно погрешности вычислений и необходимости корректировки шага.

Оценим замедление расчётов, обусловленное наличием дополнительных вычислений правой части уравнения (70). На каждом шаге численного интегрирования по схеме Рунге — Кутты четвёртого порядка (85) правую часть $f(x, y)$ необходимо вычислять четыре раза (один раз в начале шага, два — в середине, и ещё один — на правом конце отрезка). Следовательно, на паре шагов правая часть $f(x, y)$ должна быть вычислена восемь раз. При пересчёте с двойным шагом необходимо вычислить $f(x, y)$ ещё четыре раза. Итого получаем двенадцать вычислений $f(x, y)$. Заметим, однако, что начальная точка для двойного шага совпадает с начальной точкой для пары двойных шагов (рис. 8), что позволяет сократить число вычислений $f(x, y)$ до одиннадцати на каждой паре шагов. Учитывая, что без пересчёта на сетке с шагом $2h$ мы должны были сделать 8 шагов, получаем коэффициент замедления $11/8 = 1{,}375$. Однако, «теряя» 40 % операций на дублирование расчётов на сетке с двойным шагом, при решении ряда задач можно добиться сокращения общего времени вычислений на порядок величины и более при фиксированной точности за счёт увеличения шага сетки там, где функция $f(x, y)$ мала, и решение меняется медленно.

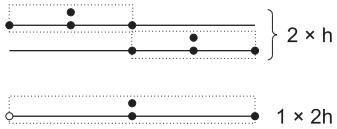


Рис. 8. Вычисление $f(x, y)$ при пересчёте с двойным шагом

После выполнения двух шагов h численного интегрирования по схеме (85) и одного шага $2h$ мы получим пару значений сеточной функции y, назовём их y_1 и y_2 (NB: речь идёт о значениях численного решения в одном и том же узле $x + 2h$, полученных в результате вычислений на сетках с одинарным и двойным шагом):

$$y(x + 2h) = y_1 + 2K \cdot h^5 + \mathcal{O}(h^6), \qquad (86)$$
$$y(x + 2h) = y_2 + K \cdot (2h)^5 + \mathcal{O}(h^6),$$

где K — коэффициент пропорциональности в остаточном члене $R = \mathcal{O}(h^5) = Kh^5$. В первом случае (расчёт на сетке с шагом h) суммарная погрешность складывается из погрешностей на двух шагах h, поэтому перед коэффициентом пропорциональности K в остаточном члене стоит множитель 2.

Вычитая одно уравнение из другого, несложно получить оценку для погрешности вычислений:

$$R_1 \equiv 2Kh^5 + \mathcal{O}(h^6) = \frac{y_2 - y_1}{15} + \mathcal{O}(h^6). \tag{87}$$

Полученная разность является мерой погрешности численного интегрирования на одном шаге и может быть использована для принятия решения о корректировке шага численного интегрирования. Для этого необходимо сравнить фактическую погрешность (87) с заданным значением так, чтобы выполнялось неравенство

$$R_1 < \delta + |y| \cdot \varepsilon,$$

где δ и ε — максимально допустимая на одном шаге абсолютная и относительная погрешность соответственно. В случае невыполнения указанного неравенства нужно уменьшить шаг h и выполнить пересчёт последнего шага. Если $R_1/(\delta + |y|\varepsilon) < 1$, можно оценить безопасное увеличение величины шага сетки на следующем шаге интегрирования, используя известную степенную зависимость (86) остаточного члена от шага сетки h.

Заметим также, что учитывая найденную погрешность в выражении (86), можно повысить порядок точности численной схемы (фактически, используя *метод Рунге — Ромберга*):

$$y(x + 2h) = y_1 + \frac{y_2 - y_1}{15} + \mathcal{O}(h^6). \tag{88}$$

Проводя вычисления с использованием (88), можно повысить порядок точности метода, однако при этом нельзя будет оценить погрешность вычислений, не прибегая к дублированию расчётов ещё на одной сетке.

6.6. Многошаговые методы

В случае, если правая часть уравнения (70) имеет большое число непрерывных производных, может быть целесообразно использовать численные схемы более высокого порядка точности. Время, затраченное на написание и отладку программ, реализующих более сложные

методы высокого порядка точности, зачастую с лихвой компенсируется повышением скорости расчётов за счёт использования сетки с меньшим числом узлов при сохранении заданного уровня точности вычислений. Ниже будет кратко рассмотрен класс относительно простых по форме записи и в реализации многошаговых методов высокого порядка точности.

Идея, лежащая в основе многошаговых методов, заключается в использовании в расчётах вместо $f(x,u)$ в правой части уравнения (70) некоторой другой функции $F(x)$, аппроксимирующей f с заданным порядком точности:

$$F(x) = f(x, u(x)) + \mathcal{O}(h^p).$$

В наиболее простом и часто встречающемся случае в качестве $F(x)$ используют интерполяционный полином. Очевидно, что для построения интерполяционного полинома степени $p \geq 1$ и, следовательно, для выполнения шага численного интегрирования ОДУ уже недостаточно знания одного лишь значения $y(x_n)$ — необходимо иметь значения $y(x)$ в $p+1$ узле сетки (на $p+1$ предыдущем шаге численного интегрирования). Именно поэтому рассматриваемые методы называются *многошаговыми*. Считая значения $y_n, y_{n-1}, \ldots, y_{n-p}$ известными, запишем интерполяционный полином в форме Ньютона (см. (54) на с. 75):

$$\begin{aligned} F(x) \quad = \quad & F(x_n) + \\ & + (x - x_n)F(x_n, x_{n-1}) + \\ & + (x - x_n)(x - x_{n-1})F(x_n, x_{n-1}, x_{n-2}) + \\ & + (x - x_n)(x - x_{n-1})(x - x_{n-2})F(x_n, x_{n-1}, x_{n-2}, x_{n-3}) + \ldots \end{aligned} \tag{89}$$

Для вычисления решения $u(x)$ в следующем узле сетки запишем дифференциальное уравнение (70) в интегральной форме

$$u_{n+1} = u_n + \int\limits_{x_n}^{x_{n+1}} f(x, u(x))\, dx \approx u_n + \int\limits_{x_n}^{x_{n+1}} F(x)\, dx$$

и подставим в него интерполяционный многочлен (89) третьей степени. Вычисляя возникающие при этом элементарные интегралы вида

$$\int\limits_{x_n}^{x_{n+1}} (x - x_n)\, dx = \frac{h_n^2}{2}, \quad \text{где } h_n = x_{n+1} - x_n,$$

$$\int\limits_{x_n}^{x_{n+1}} (x - x_n) \cdot (x - x_{n-1})\, dx = \frac{h_n^3}{3} + \frac{h_n^2}{2} h_{n-1},$$

$$\int\limits_{x_n}^{x_{n+1}} \left(\prod_{k=n-2}^{n} (x - x_k) \right) dx =$$

$$= \frac{h_n^4}{4} + \frac{h_n^3}{3}(2h_{n-1} + h_{n-2}) + \frac{h_n^2}{2} h_{n-1}(h_{n-1} + h_{n-2}),$$

приходим в итоге к *формуле Адамса для переменного шага*:

$$\begin{aligned} y_{n+1} \quad &= \quad y_n + h_n F(x_n) + \frac{1}{2} h_n^2 F(x_n, x_{n-1}) + \\ &+ \left[\frac{1}{3} h_n^3 + \frac{1}{2} h_n^2 h_{n-1} \right] \cdot F(x_n, x_{n-1}, x_{n-2}) + \\ &+ \left[\frac{h_n^4}{4} + \frac{1}{3} h_n^3 (2h_{n-1} + h_{n-2}) + \frac{1}{2} h_n^2 h_{n-1}(h_{n-1} + h_{n-2}) \right] \times \\ &\times F(x_n, x_{n-1}, x_{n-2}, x_{n-3}). \end{aligned} \tag{90}$$

Схема (90) имеет четвёртый порядок точности. Если отбросить последнее слагаемое, получим формулу третьего порядка точности и т. д. Оставляя в (90) только одно слагаемое $y_{n+1} = y_n + h_n F(x_n)$, получим схему ломаных (73), имеющую первый порядок точности.

Рассмотренный выше метод Адамса четвёртого порядка является *явным*, т.к. значение y_{n+1} может быть вычислено по предыдущим значениям y_n, y_{n-1}, ... с использованием выражения (90) за фиксированное число операций. Явные методы Адамса также называют в литературе методами *Адамса — Башфорта* (Adams, Bashforth). *Неявные* методы Адамса — схемами *Адамса — Мултона* (Adams, Moulton).

На первый взгляд, метод Адамса (90) кажется привлекательным тем, что на каждом шаге необходимо вычислять $f(x, y)$ лишь один раз, тогда как в методе Рунге-Кутты (85), также имеющем четвёртый порядок точности, $f(x, y)$ необходимо вычислять четыре раза на каждом шаге. Это могло бы дать существенный выигрыш в скорости счёта, особенно для тех случаев, когда вычисление $f(x, y)$ занимает много времени. Однако если оценить коэффициент в остаточном члене схемы (90) в простейшем случае равномерной сетки, мы получим $R_{\text{Adams}} = \frac{251}{750} h^5 F^{\text{IV}}(x)$ [2, гл. VIII §7], тогда как остаточный член для метода Рунге — Кутты (85) на равномерной сетке совпадает с остаточным членом для метода Симпсона и равен $R_{\text{Simpson}} = \frac{1}{2880} h^5 f^{\text{IV}}$. Таким образом, остаточный член в четырёхчленной схеме Рунге — Кутты (85) в 960 раз меньше, чем в методе Адамса, что позволяет использовать сетку с шагом, увеличенным в $\sqrt[4]{960} \approx 5{,}57$ раз, т. е. фактически вычислять $f(x, y)$ меньшее число раз, чем в методе Адамса того же порядка

точности. Относительно большое значение численного коэффициента в остаточном члене метода Адамса обусловлено использованием полиномов для экстраполяции функции $F(x) \equiv f(x, u(x))$ за пределы отрезка $[x_{n-p}, x_n]$, на котором был построен интерполяционный полином (см. п. 5.3).

Повышая степень интерполяционного полинома (89), несложно построить численную схему Адамса более высокого порядка точности, хотя даже схема четвёртого порядка (90) на неравномерной сетке достаточно громоздка. К числу «минусов» методов Адамса следует также отнести необходимость задания начальных условий в p узлах сетки (для использования метода p-го порядка). Это приводит к необходимости выполнения начальных расчётов с использованием других (самостартующих) схем (например, метода Рунге — Кутты четвёртого порядка (85)), что может вдвое увеличить размер программы для ЭВМ. Данное обстоятельство, наряду с относительно большим коэффициентом в остаточном члене, ограничивает практическую применимость методов Адамса.

6.7. Жёсткие системы уравнений

Рассмотрим понятие устойчивости методов численного интегрирования ОДУ на примере уравнения

$$u' = -\lambda u, \quad \lambda > 0 \tag{91}$$

Решение уравнения (91): $u(x) = u(0)e^{-\lambda x}$. Для любого $a > 0$ справедливо $|u(x+a)| \leq u(x)$, т. е. решение уравнения асимптотически устойчиво. Естественно ожидать устойчивости и от численного решения. Для y_n, построенного по схеме Эйлера (73), имеем:

$$y_{n+1} = y_n + hf(x_n, y_n) = (1 - h\lambda)\, y_n.$$

Для того, чтобы численное решение было устойчивым, необходимо выполнение условия $|1 - h\lambda| < 1$, откуда

$$h < \frac{2}{\lambda}. \tag{92}$$

Следовательно, численное решение уравнения (91), полученное по схеме Эйлера (73), является устойчивым при выполнении условия (92). В этой связи говорят, что численная схема (73) является *условно устойчивой*.

Рассмотрим численное решение того же уравнения (91), полученного с помощью *неявной* схемы Эйлера:

$$y_{n+1} = y_n + h\, f(x_{n+1}, y_{n+1}). \tag{93}$$

Обратим внимание, что если явный метод Эйлера наряду с другими *явными* методами даёт выражение для вычисления y_{n+1}, то *неявная* схема (93) представляет собой уравнение, в которое искомое значение y_{n+1} численного решения в следующем узле сетки входит в аргумент функции $f(x, u)$. Схема (93), следовательно, является нелинейным конечным уравнением на y_{n+1}. Исследуем асимптотическую устойчивость численного решения, построенного по схеме (93) для уравнения (91):

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) = y_n - h\lambda\, y_{n+1},$$

откуда

$$y_{n+1} = \frac{y_n}{1 + h\lambda}.$$

Видно, что для любого шага сетки h численное решение, построенное по неявной схеме (93), будет устойчивым. В этой связи говорят, что неявная численная схема (93) является *абсолютно устойчивой*, в отличие от *условно устойчивой* явной схемы (73). Общий вывод может быть сформулирован следующим образом: *явные численные схемы для решения ОДУ могут обладать лишь условной устойчивостью, тогда как среди неявных схем есть абсолютно устойчивые.*

Обратим внимание, что для получения приемлемой точности численного решения уравнения (91) с использованием схем первого порядка (73) и (93) необходимо выполнение условия $h \ll 1/\lambda$, которое является более сильным, чем полученное выше условие устойчивости (92). Если неустойчивость проявляется при использовании настолько больших шагов сетки h, что получаемое численное решение уже не аппроксимирует точное решение ОДУ, зачем в таком случае нужно вводить понятие условной устойчивости? Зачем вообще рассматривать и исследовать численные решения на сетках со столь большим шагом h?

Ответ становится понятным при переходе от одного уравнения (70) к системе ОДУ. Рассмотрим суть проблемы на самом простом примере — системе двух независимых линейных ОДУ:

$$\left\{ \begin{array}{lcl} u_1' & = & -u_1 \\ u_2' & = & -1000u_2 \end{array} \right. \quad \Rightarrow \quad \left\{ \begin{array}{lcl} u_1(x) & = & u_1(0) \cdot e^{-x} \\ u_2(x) & = & u_2(0) \cdot e^{-1000x} \end{array} \right. \tag{94}$$

Функция $u_2(x)$ быстро затухает, так что поведение решения $\mathbf{u}(x) = (u_1(x), u_2(x))$ практически полностью определяется первой компонентой $u_1(x)$. Тем не менее, шаг численного интегрирования системы ОДУ

должен определяться компонентой u_2, несущественной с точки зрения физики. Для обеспечения устойчивости численного решения $\mathbf{u}(x)$, полученного методом Эйлера, в соответствии с (92) необходимо выбрать $h \le 0{,}002$, в противном случае u_2 будет экспоненциально возрастать — возникнет неустойчивость. В следующих лекциях будет показано, что данная проблема усугубляется при численном решении уравнений в частных производных, разностные схемы для которых являются, по сути, системой большого количества ОДУ.

Система (94) называется *жёсткой*, поскольку имеет существенно различные (в частном случае (94) отличающиеся в тысячу раз) масштабы изменения различных компонент решения u_1 и u_2 (в общем случае — некоторых их линейных комбинаций). Обобщим сказанное на случай нераспавшихся систем.

Определение. Говорят, что система линейных уравнений $A\mathbf{u}' = \mathbf{f}$ *является жёсткой*, если велико отношение максимального и минимального модуля собственных значений

$$s = \frac{\max \mathrm{Re}|\lambda_k|}{\min \mathrm{Re}|\lambda_k|}, \qquad \lambda_k < 0, \quad k = 1, 2, \ldots, n.$$

Указанное отношение называют *числом жёсткости системы.* В случае, если матричные коэффициенты зависят от x, система может оказаться жёсткой в некоторой области по x.

В качестве примера рассмотрим систему из двух уравнений с постоянными коэффициентами [4],[A11]:

$$\left\{ \begin{array}{rcrcl} u' & = & 998\,u & + & 1998\,v, \\ v' & = & -999\,u & - & 1999\,v. \end{array} \right. \tag{95}$$

Собственные числа матрицы системы (95) равны $\lambda_1 = -1, \lambda_2 = -1000$, откуда число жёсткости равно $\lambda_2/\lambda_1 = 10^3 \gg 1$ и, следовательно, система (95) является жёсткой. Общее решение системы имеет вид

$$\begin{pmatrix} u(x) \\ v(x) \end{pmatrix} = \alpha \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} e^{-x} + \beta \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} e^{-1000x}.$$

При $\alpha \approx 1$, $\beta \ll \alpha$ значения в правой части (95) будут порядка 1, однако при использовании метода Эйлера (73) следует выбирать шаг сетки $h < 0{,}002$, в противном случае численное решение будет неустойчивым.

Использование неявных схем для численного решения систем ОДУ позволяет существенно повысить скорость вычислений за счёт использования сетки с меньшим числом узлов. Очевидно, что за это приходит-

ся платить необходимостью решения конечного (чаще всего, нелинейного трансцендентного) уравнения на каждом шаге численного интегрирования. Рассмотрим данную задачу на примере простейшей неявной схемы первого порядка (93). Для выполнения шага численного интегрирования нам необходимо разрешить уравнение (93) относительно y_{n+1}. Какими методами может быть решена эта задача?

Метод дихотомии не обобщается на системы уравнений и потому не подходит для данной задачи. Посмотрим, возможно ли применение метода простых итераций и метода Ньютона — Рафсона.

Для использования итерационных методов необходимо, чтобы итерационный процесс сходился, и, кроме того, нужно указать условие завершения итераций при вычислении y_{n+1} в программе. На практике зачастую пользуются так называемыми *полунеявными* методами, которые заключаются в выполнении фиксированного числа итераций (например, одной или двух). Несложно понять, что при использовании конечного фиксированного числа шагов результат использования итерационного процесса — значение y_{n+1} — можно *явным образом* выразить через y_n. Как следствие, полунеявные методы по сути являются разновидностью явных численных схем и потому не могут обладать абсолютной устойчивостью. Тем не менее, они успешно используются при интегрировании жёстких систем, позволяя существенно ослабить ограничение на шаг сетки и достичь тем самым выигрыша в быстродействии.

Рассмотрим использование неявных схем на примере системы из двух уравнений общего вида:

$$\left\{ \begin{array}{lcl} u' & = & f(u,v), \\ v' & = & g(u,v). \end{array} \right. \tag{96}$$

Зависимость f и g от x мы предполагаем, но для краткости не пишем. Будем решать систему (96) по неявной схеме (93). Полагая функции f и g гладкими, можем записать:

$$\left\{ \begin{array}{lcl} \xi_n \equiv u_{n+1} - u_n & = & hf + hf_u \cdot \xi_n + hf_v \cdot \eta_n + \mathcal{O}(\xi_n^2, \eta_n^2), \\ \eta_n \equiv v_{n+1} - v_n & = & hg + hg_u \cdot \xi_n + hg_v \cdot \eta_n + \mathcal{O}(\xi_n^2, \eta_n^2), \end{array} \right.$$

где для краткости значения функций и их частных производных на n-м шаге по x обозначены $f \equiv f(u_n, v_n)$, $f_u \equiv \frac{\partial f}{\partial u}(u_n, v_n)$, ... Удобно переписать полученные соотношения в матричном виде:

$$\begin{pmatrix} \xi_n \\ \eta_n \end{pmatrix} = h\hat{J} \begin{pmatrix} \xi_n \\ \eta_n \end{pmatrix} + h\mathbf{f} + \mathcal{O}(\xi^2, \eta^2), \tag{97}$$

$$\hat{J} \equiv \begin{pmatrix} f_u & f_v \\ g_u & g_v \end{pmatrix}, \quad \mathbf{f} \equiv \begin{pmatrix} f \\ g \end{pmatrix}.$$

Ввиду наличия нелинейных слагаемых $\mathcal{O}(\xi^2, \eta^2)$ уравнение (97) в общем случае не решается точно. Попытаемся построить его численное решение, используя вначале метод простых итераций:

$$\begin{pmatrix} \xi_n^{(s+1)} \\ \eta_n^{(s+1)} \end{pmatrix} = h\hat{J} \begin{pmatrix} \xi_n^{(s)} \\ \eta_n^{(s)} \end{pmatrix} + h\mathbf{f} + \mathcal{O}(\xi^2, \eta^2). \tag{98}$$

здесь верхний индекс обозначает номер итерации, нижний — узел сетки. Исследуем сходимость итерационного процесса (98). Пусть $\mathbf{u}_*$ — решение системы (97), т. е. в первом порядке по h можно записать

$$\mathbf{u}_* = h\hat{J}\mathbf{u}_* + h\mathbf{f}.$$

Для того, чтобы итерационный процесс сходился, необходима устойчивость по малым отклонениям от $\mathbf{u}_*$. Подставляя в (97) $(\xi^{(s)}, \eta^{(s)}) = \mathbf{u}_* + \varepsilon^{(s)}$, получаем с точностью до членов второго порядка малости

$$\varepsilon^{(s+1)} = h\hat{J}\varepsilon^{(s)}.$$

Следовательно, итерационный процесс (98) будет сходиться, если все собственные числа матрицы $h\hat{J}$ (97) по модулю меньше 1. (В случае одной переменной данное условие переходит в $|\varphi'| < 1$ — условие сходимости итерационного процесса $x^{(s+1)} = \varphi(x^{(s)})$ для уравнения $x = \varphi(x)$.) Указанное условие приводит нас к ограничению на максимальное значение h, аналогичное условию (92). Действительно, как и (92), условие $h \cdot |\lambda_k(J)| < 1$ накладывает ограничение на произведение шага сетки h на некоторую комбинацию частных производных f_u, f_v, g_u и g_v, входящих в матрицу Якоби $\hat{J}$.

Заметим, что уменьшение h при поиске y_{n+1} хотя и обеспечивает сходимость итерационного процесса (98), но при этом приводит ровно к той же потере скорости счёта, что и ограничение на шаг сетки, связанное с условием устойчивости (92) для явной схемы Эйлера (73).

В этой связи представляется целесообразным использовать для численного решения уравнений (97) метод Ньютона. По сути, шаг итераций в методе Ньютона есть точное решение линеаризованной задачи (97). Решая (97) в линейном приближении по h, имеем:

$$\left[E - h\hat{J}\right] \begin{pmatrix} \xi_n \\ \eta_n \end{pmatrix} = h\mathbf{f}.$$

Организуя итерационный процесс в соответствии с полученными формулами и записывая номер итерации в верхнем индексе:

$$\begin{pmatrix} \xi_n^{(s+1)} \\ \eta_n^{(s+1)} \end{pmatrix} = \left[E - h\hat{J}^{(s)}\right]^{-1} \left(h\mathbf{f}^{(s)} + \mathbf{u}_n - \mathbf{u}_{n+1}^{(s)}\right),$$

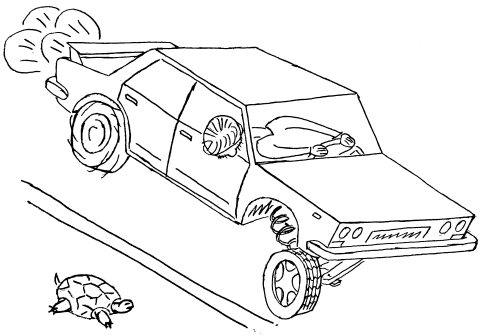
$$\begin{pmatrix} u_{n+1}^{(s+1)} \\ v_{n+1}^{(s+1)} \end{pmatrix} = \begin{pmatrix} u_{n+1}^{(s)} \\ v_{n+1}^{(s)} \end{pmatrix} + \left[E - h\hat{J}^{(s)}\right]^{-1} \begin{pmatrix} u_n - u_{n+1}^{(s)} + hf^{(s)} \\ v_n - v_{n+1}^{(s)} + hg^{(s)} \end{pmatrix} \quad (99)$$

где $f^{(s)}$, $g^{(s)}$ и $\hat{J}^{(s)}$ — значения функций и их частных производных, вычисленные в точке $(u_{n+1}^{(s)}, v_{n+1}^{(s)})$. В качестве начального приближения $(u_{n+1}^{(0)}, v_{n+1}^{(0)})$ в (99) естественно использовать (u_n, v_n).

В случае линейных систем с постоянными коэффициентами (таких, например, как (94) и (95)), однократное применение формулы Ньютона (99) даёт точное решение задачи: итерационный процесс (99) сходится за один шаг.

В общем случае сходимость итерационного процесса (99) не гарантируется, однако в ряде случаев данный метод может быть успешно использован на практике, обеспечивая приемлемую точность при использовании 1-2 итераций для неявной схемы первого порядка (93).

Как и в случае явных схем, в расчётах целесообразно использовать методы более высокого порядка точности при условии, что входящие в уравнения функции являются достаточно гладкими. Неявные схемы, возникающие в результате обобщения методов Рунге — Кутты, известны как методы Розенброка (Rosenbrock), а также методы Канса — Рентропа (Kans, Rentrop). Неявные многошаговые методы Адамса известны как методы Адамса — Мултона (Adams, Moulton).



6.8. Заключение

Таким образом, в данной главе был рассмотрен ряд базовых методов интегрирования обыкновенных дифференциальных уравнений и их

систем. Наиболее простым для анализа и реализации является явный метод Эйлера, однако на практике следует избегать его использования ввиду крайне низкой скорости сходимости. Наиболее употребительным в приложениях является метод Рунге — Кутты 4-го порядка точности. Достичь ещё более высокого порядка точности (что имеет смысл лишь для достаточно гладких функций в правой части ОДУ, которые притом хорошо интерполируются полиномами) позволяют методы Адамса. К их недостаткам относится отсутствие самостарта, относительно большие численные коэффициенты в остаточном члене и отсутствие абсолютной устойчивости даже при использовании неявных модификаций многошаговых методов (см. [A8, с. 357] и [A1, с. 277]). При интегрировании жёстких систем ОДУ численные решения, построенные по явным схемам, могут быть неустойчивыми при недостаточно мелком шаге сетки. Достичь существенного выигрыша в скорости интегрирования жёстких систем можно с использованием неявных схем, обладающих абсолютной устойчивостью.

Дополнительную информацию по интегрированию ОДУ и их систем можно найти в [2, гл. 8]. Достаточно подробное рассмотрение вопросов, связанных с устойчивостью решений и жёсткими системами можно найти в [A8, гл. 8]. Пяти- и шестичленные формулы Рунге — Кутты можно найти в [A1, с. 277]. Последовательное рассмотрение смежных вопросов, важных для построения эффективной и надёжной программы для численного интегрирования ОДУ, — инициализации, контроля шага интегрирования и порядка точности метода, контроля ошибки результатов, проверок корректности — можно найти в [A1, гл. 9.3].

Упражнения

1. Построить численное решение уравнения $\dot{x} = -x$ с начальным условием $x(0) = 1$ на отрезке $0 \le t \le 3$ методами Эйлера и Рунге — Кутты второго порядка точности. Исследовать зависимость погрешности численного решения $R(t)$ при фиксированном шаге сетки h, а также зависимость $R(h)$ при $t = 3$.

2. Пушечное ядро (шар $m = 2$ кг $R = 7{,}5$ см) выпущено из рельсового ускорителя с начальной скоростью u $= 2$ км/с, направленной под углом α к горизонту. Считая, что плотность атмосферы убывает экспоненциально с высотой ($h_0 = 10$ км, $\rho_0 = 1{,}3$ кг/м^3), найти максимальную дальность полёта ядра как функцию α. Сопротивление воздуха $F = 0{,}47 \cdot \frac{1}{2}\rho v^2 \cdot \pi R^2$, $\rho = \rho_0 \exp(-h/h_0)$.

3. Найти минимальную добавку к скорости, которую необходимо

сообщить зонду на околоземной орбите, чтобы он достиг орбиты Урана, учитывая только гравитационное притяжение Солнца и Земли. Рассчитать отклонение афелия зонда, запущенного к Урану с найденной скоростью, вызванное притяжением других планет Солнечной системы. Для простоты считать, что движение планет и зонда происходит строго в плоскости эклиптики. Для определения координат планет использовать доступные в сети Интернет эфемериды с открытым исходным кодом (см. ephemeris source code в Google), либо (в простейшем случае) выбрать начальные угловые координаты планет случайным образом.

4. Исследовать движение неуправляемого космического аппарата в одной из точек Лагранжа системы Солнце — Земля. Через какое время аппарат покинет точку Лагранжа из-за возмущений, вызванных притяжением других планет и Луны? Можно ли удерживать искусственный спутник массой 0.5 т в точке Лагранжа, используя электрический ракетный двигатель с тягой 0.1 Н?

5. Используя неявную схему Эйлера, решить жёсткую систему уравнений (95) на с. 103.

6. Решить систему (95) на с. 103, используя явный метод Эйлера и метод Рунге — Кутты четвёртого порядка. Сравнить время интегрирования указанными методами с временем выполнения программы в упражнении 5.

7. Используя методы Рунге — Кутты второго и четвёртого порядка, исследовать колебания физического маятника $\ddot{\theta} + (g/L)\sin\theta = 0$. Найти зависимость периода колебаний от амплитуды. Сравнить с известными выражениями для гармонических колебаний и ангармонического осциллятора и с результатом упражнения 9 на с. 69. Меняя шаг временнóй сетки, построить зависимость погрешности результата от затраченного машинного времени для различных методов интегрирования ОДУ.

Рекомендации по написанию программ

1. Необходимо проверять результаты работы программы на разумность, предельные случаи, согласие с аналитическими решениями там, где они могут быть получены. Ценность программы, которая компилируется и «что-то» считает, близка к нулю.

2. Все константы, соответствующие физическим величинам и расчётным параметрам, должны быть объявлены в виде отдельных переменных (в том числе со спецификатором `const`) либо с помощью директивы препроцессора `#define`:

   ```
   //Под MSVC используйте #define _USE_MATH_DEFINES перед
   //подключением <math.h> !
   printf("Circle length: %f\n", 3.14159*r); //ПЛОХОЙ СПОСОБ
   printf("Circle length: %f\n", M_PI*r);   //ВЕРНОЕ РЕШЕНИЕ
   ```

3. Ставьте точки при целочисленных константах, используемых при вычислениях величин с плавающей точкой:

   ```
   const double x = 2/3;    //x будет равен нулю!
   const double y = 2./3;   //y будет равен двум третям
   ```

4. Минимизируйте повторяющийся программный код, выносите его в отдельные функции, тестируйте их работу и затем используйте повторно. Помните: грамотно спроектированную и разбитую на логические блоки программу отлаживать значительно легче.

5. Используйте форматирование кода для улучшения его читаемости — это позволит сократить число ошибок в программе [C7, гл. 1]:

   `http://ru.wikipedia.org/wiki/Стандарт_оформления_кода`
   `http://ru.wikipedia.org/wiki/Спагетти-код`

6. Используйте графики для анализа полученных результатов. Как правило, графическая информация воспринимается человеком гораздо легче и быстрее чисел.

7. Если программа работает неправильно, и ошибку не удаётся найти внимательным чтением исходного кода, добавьте вывод отладочных сообщений (значений переменных, сеточных функций и т. п.), либо используйте пошаговое выполнение в отладчике, вручную проверяя результаты промежуточных вычислений.

8. Перед тем как вносить в программу сколько-нибудь серьёзные изменения, а также каждый раз после получения значимого результата, обязательно сохраните резервную копию! Использование систем контроля версий (например, git) позволит не только сэкономить время и силы на поиск внезапно возникших проблем с программой, которая «только что работала», но и упростит повторение полученных ранее результатов, необходимость чего регулярно возникает в научных исследованиях.

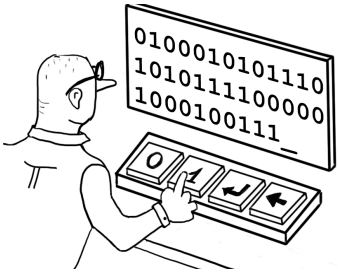
Упражнения

1. Что напечатает следующая программа, написанная на языке Си? Введите текст программы в файл с расширением `.c`, скомпилируйте и запустите его:

```
#define v putchar
#define print(x) main(){v(4+v(v(52)-4));return 0;}/*
#>+++++++4+[>++++++<-]>++++.----.++++.*/
print(202*2);exit();
#define/*>.@*/exit()
```

2. Проверьте, что программа корректно выполняется интерпретаторами языков программирования Perl, Python, Ruby, Befunge, BF.

3. Сравните стиль собственных программ с кодом из упражнения 1.



Рекомендованная литература

[1] *Керниган Б. У., Ритчи Д. М.* Язык программирования С. М.: Вильямс, 2012. 289 с.

[2] *Калиткин Н. Н.* Численные методы. М.: Наука, 1978. 512 с.

[3] *Самарский А. А.* Введение в численные методы. М.: Наука, 1987. 286 с.

[4] *Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.* Numerical recipes The art of scientific computing. N. Y.: Cambridge University Press, 2007.

Дополнительная литература и Интернет-ресурсы

[A1] *Rice J. R.* Numerical methods, software, and analysis. N. Y.: McGraw-Hill, 1983.

[A2] Gnuplot homepage. URL: http://www.gnuplot.info/

[A3] *Phillips L.* Gnuplot Cookbook. Birmingham: Packt Publishing Ltd., 2012.

[A4] *Janert P. K.* Gnuplot in action: understanding data with graphs. Greenwich CT USA: Manning, 2010.

[A5] 14882 ISO/IEC. C++ International standard. Geneva: ISO, 2011.

[A6] *Severance C.* An interview with the old man of floating-point. Reminiscences elicited from William Kahan, 1998. URL: http://www.cs.berkeley.edu/∼wkahan/ieee754status/754story.html

[A7] *Goldberg D.* What every computer scientist should know about floating-point arithmetic // ACM Computing Surveys. 1991. V. 23. N. 1. P. 5–48.

[A8] *Каханер Д., Моулер Д., Нэш С.* Численные методы и программное обеспечение. М.: Мир, 2001. 575 с.

[A9] *Кнут Д. Э.* Искусство программирования. Т.2: Получисленные алгоритмы. М.: Вильямс, 2000. 828 с.

[A10] *Самарский А. А., Гулин А. В.* Численные методы. М.: Наука, 1989. 429 с.

[A11] *Форсайт Дж., Малькольм М., Моулер К.* Машинные методы математических вычислений. М.: Мир, 1980. 279 с.

Литература и Интернет-ресурсы по языкам Си/Си++

[C1] *Керниган Б. У., Ритчи Д. М.* Язык программирования С. М.: Вильямс, 2012. 289 с.

[C2] *Шилдт Г.* Полный справочник по C++. М.: Вильямс, 2007. 800 с.

[C3] Справочник по C++. URL: http://cppreference.com/

[C4] Статьи, уроки и справочная информация по C/C++. URL: http://www.cplusplus.com/

[C5] 14882 ISO/IEC. C++ International standard. Geneva: ISO, 2011.

[C6] *Страуструп Б.* Язык программирования C++. М.: Бином, 2012. 1135 с.

[C7] *Керниган Б., Пайк Р.* Практика программирования. М.: Вильямс, 2004. 287 с.

Учебное издание

Смирнов Сергей Валерьевич

ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ФИЗИКИ
Часть I

Учебное пособие

Редактор *Т. Ю. Седыченко*
Подготовка к печати *С. В. Исакова*

Подписано в печать 25.09.2015
Формат 60x84 1/16. Уч.-изд. л. 7. Усл. печ. л. 6,6
Тираж 200 экз. Заказ №

Редакционно-издательский центр НГУ
630090, г. Новосибирск, ул. Пирогова, 2