А. Н. ТИХОНОВ
Д. П. КОСТОМАРОВ

ВВОДНЫЕ ЛЕКЦИИ ПО ПРИКЛАДНОЙ МАТЕМАТИКЕ

А. Н. ТИХОНОВ
Д. П. КОСТОМАРОВ

ВВОДНЫЕ ЛЕКЦИИ ПО ПРИКЛАДНОЙ МАТЕМАТИКЕ

Допущено Министерством высшего и среднего специального образования СССР в качестве учебного пособия для студентов вузов, обучающихся по специальности «Прикладная математика»

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1984

22.19
Т 46
УДК 519.6

Вводные лекции по прикладной математике. Тихонов А. Н., Костомаров Д. П.— М.: Наука. Главная редакция физико-математической литературы, 1984.—192 с.

Книга является учебным пособием по курсу «Введение в специальность» для студентов высших учебных заведений, обучающихся по специальности «Прикладная математика». Изложенный в ней материал дает общее представление об идеях и методах современной прикладной математики, об ее программных и технических средствах, знакомит с проблемами и трудностями исследований, связанных с применением математических методов на базе ЭВМ к решению задач народного хозяйства.

Рецензенты:
кафедра прикладной математики Московского
института электронного машиностроения;
доктор физико-математических наук *Л. Д. Кудрявцев*

Т $\frac{1702070000-130}{053\,(02)-84}$ 165—84

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Создание в середине XX в. электронно-вычислительных машин (ЭВМ) можно сравнить по своей значимости с любым из самых выдающихся технических достижений в истории человечества. В то же время необходимо подчеркнуть их особую, специфическую роль. Если обычные машины расширяют физические возможности людей в процессе трудовой деятельности, то ЭВМ являются их интеллектуальными помощниками. Широкое применение математических методов на базе ЭВМ привело к появлению новых эффективных методов познания законов реального мира и их использованию в практической деятельности. Вычислительные машины открыли новые возможности увеличения производительности труда, дальнейшего развития производства, совершенствования управления.

Процесс математизации науки, техники, экономики потребовал подготовки высококвалифицированных специалистов, в совершенстве владеющих технологией применения ЭВМ, способных реализовать их огромные и пока еще далеко не исчерпанные возможности. ЭВМ не работают без направляющего воздействия человека. Их использование связано с построением математических моделей и созданием вычислительных алгоритмов. Машины также должны пройти соответствующее «обучение», т. е. получить программное обеспечение как общего, так и проблемно-ориентированного характера. Весь этот широкий комплекс проблем является полем деятельности специалистов по прикладной математике, для подготовки которых в семидесятые годы во многих университетах и институтах страны были созданы новые факультеты, отделения, кафедры.

Настоящая книга представляет собой учебное пособие по курсу «Введение в специальность» для студентов высших учебных заведений, обучающихся по специальности «Прикладная математика». Согласно действующему учебному плану такой курс читается в начале обучения. Он призван

дать студентам общее представление о прикладной математике, о проблемах и трудностях исследований, связанных с применением математических методов и вычислительной техники в народном хозяйстве, показать содержание творческой деятельности математика — прикладника.

Книга состоит из семи глав. Начинается она с четырех глав, посвященных четырем основным элементам современной прикладной математики: математическим моделям, вычислительным алгоритмам, электронно-вычислительным машинам, программированию и математическому обеспечению ЭВМ. Этот материал вводит читателя в круг идей и методов прикладной математики, знакомит с ее программными и техническими средствами. Три последние главы носят более специальный характер. Они посвящены системам линейных алгебраических уравнений, задачам оптимизации и численному интегрированию. Основное внимание в этих главах сосредоточено на алгоритмических и вычислительных вопросах. Цель этих глав — подготовить читателя к изучению численных методов и к работе в вычислительном практикуме.

Книга, как и курс, которому она соответствует, носит вводный характер. Многие вопросы в ней ставятся и обсуждаются на простейших примерах. Предполагается, что в дальнейшем, в соответствии с учебным планом специальности, они будут изучены более подробно в общих и специальных курсах.

Огромную помощь в работе над книгой нам оказали наши коллеги: В. Г. Баула, Н. П. Бруеенцов, Н. Н. Ефимов, Х. Д. Икрамов, В. Г. Карманов, Л. Н. Королев, М. Г. Мальковский, В. Н. Пильщиков, Р. Л. Смелянский, Е. В. Шикин. Всем им авторы выражают свою самую искреннюю благодарность.

Академик А. Н. Тихонов
Профессор Д. П. Костомаров

Факультет вычислительной математики
и кибернетики МГУ

ВВЕДЕНИЕ

НАУЧНО-ТЕХНИЧЕСКИЙ ПРОГРЕСС И МАТЕМАТИКА

Одной из характерных особенностей нашего времени является широкое применение математических методов и ЭВМ в самых различных областях человеческой деятельности. «Диагноз ставит ЭВМ», «Соавтор конструктора» — такие заголовки нередко встречаются сегодня в газетах. Бурный процесс математизации науки, техники и всего народного хозяйства в целом начался в 50-х годах после появления и быстрого совершенствования ЭВМ. Он привел к формированию современной прикладной математики, которая включает в себя широкий круг вопросов, связанных с применением математических методов на базе ЭВМ к решению задач народного хозяйства.

Математика — одна из самых древних наук. Она зародилась на заре человеческой цивилизации из потребностей практики. Строительство, измерение площадей земельных участков, навигация, торговые расчеты, управление государством требовали умения производить арифметические вычисления и определенных геометрических представлений. В дальнейшем математика сформировалась в стройную логическую систему, как составная часть общего комплекса научных знаний. Потребности естествознания, техники, всей практической деятельности людей постоянно ставили перед математикой новые задачи и стимулировали ее развитие. В свою очередь прогресс в математике делал математические методы более эффективными, расширял сферу их применения и, тем самым, способствовал общему научно-техническому прогрессу.

Роль математики в различных областях человеческой деятельности и в разное время была различной. Она складывалась исторически, и существенное влияние на нее оказывали два фактора: уровень развития математического аппарата и степень зрелости знаний об изучаемом объекте,

возможность описать его наиболее существенные черты и свойства на языке математических понятий и уравнений или, как теперь принято говорить, возможность построить математическую модель изучаемого объекта.

Математическая модель, основанная на некотором упрощении, идеализации, не тождественна объекту, а является его приближенным описанием. Однако благодаря замене реального объекта соответствующей ему моделью появляется возможность сформулировать задачу его изучения как математическую и воспользоваться для анализа универсальным математическим аппаратом, который не зависит от конкретной природы объекта. Математика позволяет единообразно описать широкий круг фактов и наблюдений, провести их детальный количественный анализ, предсказать, как поведет себя объект в различных условиях, т. е. спрогнозировать результаты будущих наблюдений. А ведь прогнозирование — всегда трудная задача, и оправдывающиеся прогнозы являются предметом особой гордости любой науки.

Сложность построения и исследования математической модели существенно зависит от сложности изучаемого объекта. Математические методы давно и весьма успешно применяются в механике, физике, астрономии, т. е. в науках, в которых изучаются наиболее простые формы движения материи. Математика стала языком этих наук, относящихся к разряду «точных». Значительную роль играет также математика в технике. Этим вплоть до недавнего времени исчерпывалась сфера широкого применения математических методов. Ситуация резко изменилась с появлением ЭВМ. Причина этого заключается в следующем. В математике часто встречаются задачи, решение которых не удается получить в виде формулы, связывающей искомые величины с заданными. Про такие задачи говорят, что они не решаются в явном виде. Для их решения стремятся найти какой-нибудь бесконечный процесс, сходящийся к искомому ответу. Если такой процесс указан, то, выполняя некоторое число шагов и затем обрывая вычисления (их нельзя продолжать бесконечно), мы получим приближенное решение задачи. Эта процедура связана с проведением вычислений по строго определенной системе правил, которая задается характером процесса и называется *алгоритмом*.

Такой подход к решению математических задач был известен еще до появления ЭВМ, но применялся весьма редко из-за исключительной трудоемкости больших вычислений.

Открытие У. Леверье за письменным столом «на кончике пера» планеты Нептун (он рассчитал ее траекторию по возмущениям траектории планеты Уран) было научным подвигом, навсегда вошедшим в историю науки. Однако в большинстве случаев исследователи стремились избегать больших вычислений. Поэтому сложные математические модели, для которых не удавалось получить ответ в виде формул, либо вообще не рассматривались, либо упрощались с помощью дополнительных предположений. Упрощение модели снижало степень ее соответствия изучаемому объекту, делало результаты исследования объекта менее точными и, следовательно, менее интересными, а иногда и приводило к ошибкам.

Опытный вычислитель тратил на выполнение одного арифметического действия в среднем за рабочую смену около полминуты. Современные ЭВМ выполняют миллионы операций в секунду. Таким образом, за короткий промежуток времени, порядка 40 лет, благодаря ЭВМ скорость проведения вычислений возросла примерно в 100 млн. раз. Такого скачка не было за всю историю человечества ни в одной сфере человеческой деятельности.

Применение численных методов на базе ЭВМ сразу существенно расширило класс математических задач, допускающих исчерпывающий анализ. Теперь уже исследователю при построении математической модели какого-то объекта не нужно стремиться к сильным упрощениям, которые были необходимы раньше для получения ответа в явном виде. Его внимание, прежде всего, должно быть направлено на то, чтобы правильно учесть все наиболее существенные особенности изучаемого объекта и отразить их в математической модели. После того как модель построена, встает вопрос о разработке алгоритма решения соответствующей математической задачи и о его реализации на ЭВМ. Таким образом, ЭВМ изменили подход к применению математики как метода исследования. Они вызвали переориентацию многих сложившихся направлений математики и развитие ряда новых.

Сегодня ЭВМ являются одним из определяющих факторов научно-технического прогресса. Их применение способствует ускорению развития ведущих отраслей народного хозяйства, открывает принципиально новые возможности проектирования сложных систем при значительном сокращении сроков их разработки и внедрения в производство, обеспечивает выбор оптимальных режимов п оизводствен-

но-технологических процессов, создает условия для совершенствования управления и повышения производительности труда. Если обычно машины брали на себя физические функции человека в процессе производства, то ЭВМ расширили его интеллектуальные возможности в умственной деятельности. Они являются одним из важных факторов превращения науки в непосредственную производительную силу нашего общества. Без ЭВМ не могли бы развиваться многие крупные научно-технические проекты (космические исследования, атомная энергетика, сверхзвуковая авиация и т. д.).

Благодаря ЭВМ идет интенсивный процесс математизации не только естественных и технических, но также и общественных наук; важное значение приобрело применение математических методов в экономике. Математическое моделирование начинает широко использоваться в химии, геологии, биологии, медицине, психологии, лингвистике. Большое внимание уделяется подготовке высококвалифицированных кадров, способных реализовать те огромные возможности, которые открывает эффективное использование ЭВМ. Во многих университетах и институтах созданы факультеты прикладной или вычислительной математики. Подтверждается точка зрения К. Маркса, который, по свидетельству П. Лафарга, считал, что «наука только тогда достигает совершенства, когда ей удается пользоваться математикой».

ГЛАВА 1

МАТЕМАТИЧЕСКИЕ МОДЕЛИ

§ 1. Пусть дано...

Такими словами явно или неявно обычно начинаются формулировки теорем и условий математических задач. Затем на языке строго определенных математических понятий следует полное изложение исходных предпосылок, которое воспринимается одинаково любым математиком, являющимся специалистом в соответствующей области.

Иначе обстоит дело с прикладными задачами. В них непосредственно задается реальный «нематематический» объект: явление природы, производственный процесс, конструкция, система управления, экономический план и т. д. Исследование начинается с формализации объекта, с построения соответствующей *математической модели*: выделяются его наиболее существенные черты и свойства и описываются с помощью математических соотношений. Только после того, как построена математическая модель, т. е. задаче придана математическая форма, мы можем воспользоваться для ее изучения математическими методами.

Вы знакомы с математическими моделями, хотя, может быть, раньше и не встречали этого термина. Представьте себе, что нужно определить площадь поверхности письменного стола. Для этого измеряют его длину и ширину, а затем перемножают полученные числа. Такая элементарная процедура фактически означает следующее. Реальный объект — поверхность стола — заменяется абстрактной математической моделью — прямоугольником. Прямоугольнику приписываются размеры, полученные в результате измерения, и площадь такого прямоугольника приближенно принимается за искомую площадь.

Выбор для поверхности письменного стола модели прямоугольника мы обычно делаем, полагаясь на свое зрительное восприятие. Однако человеческий глаз как измеритель-

ный инструмент не отличается высокой точностью. Поэтому при более серьезном подходе к задаче, прежде чем воспользоваться для определения площади моделью прямоугольника, эту модель нужно проверить. Проверку можно осуществить следующим образом: измерить длины противоположных сторон стола, а также длины его диагоналей, и сравнить результаты измерения между собой. Если с требуемой степенью точности длины противоположных сторон и длины диагоналей попарно равны, то поверхность стола действительно можно рассматривать как прямоугольник. В противном случае модель прямоугольника придется отвергнуть и заменить моделью четырехугольника общего вида. При более высоком требовании точности может возникнуть необходимость пойти в уточнении модели еще дальше, например, учесть закругления углов стола.

Мы так подробно обсуждаем этот простой пример, чтобы с самого начала подчеркнуть важную мысль: математическая модель не определяется однозначно исследуемым объектом. Для одного и того же стола мы можем принять либо модель прямоугольника, либо более сложную модель четырехугольника общего вида, либо еще более сложную модель «четырехугольника с закругленными углами». Выбор той или иной модели определяется требованием точности. С повышением точности модель приходится усложнять, учитывая все новые и новые особенности изучаемого объекта.

В школе с построением математических моделей вы чаще всего встречались, решая задачи по физике. В них обычно задается некоторая физическая система и описываются условия, в которых она находится. Вы должны сделать предположения о возможной идеализации этой системы (например рассматривать некоторое реальное тело как материальную точку), определить физические законы, которые нужно принять во внимание при ее изучении, и записать их в виде математических уравнений. Это и есть математическая модель рассматриваемой физической системы.

Обсудим в качестве примера следующую задачу по механике. Телу на Земле сообщили начальную скорость v_0, направленную под углом α к ее поверхности. Требуется найти траекторию движения тела и вычислить расстояние между ее начальной и конечной точками.

Чтобы сделать задачу более конкретной, предположим, что речь идет о камне, брошенном с помощью катапульты. Это уточнение определяет характерные размеры тела, его массу и возможную начальную скорость. Построим для дан-

ного случая математическую модель, основанную на следующих предположениях:

1) Земля — инерциальная система отсчета;

2) ускорение свободного падения g постоянно;

3) кривизной Земли можно пренебречь и считать ее плоской;

4) действием воздуха на движущийся камень можно пренебречь.

Введем систему координат. Ее начало совместим с катапультой, ось x направим горизонтально в сторону движения камня, ось y — вертикально вверх. При сделанных предположениях проекция камня на ось x будет двигаться равномерно со скоростью $v_x = v_0 \cos\alpha$. Движение проекции камня на ось y равноускоренное с ускорением $a_y = -g$ и начальной скоростью $v_y = v_0 \sin\alpha$. Таким образом, характер движения камня определяется формулами

$$x = tv_0 \cos\alpha, \tag{1}$$

$$y = tv_0 \sin\alpha - gt^2/2, \tag{2}$$

которые дают математическую модель задачи при предположениях 1)—4). Полученная модель является очень простой, и с ее помощью легко получить ответ на поставленный вопрос. Выразим из (1) время t через координату x:

$$t = \frac{x}{v_0 \cos\alpha}$$

и подставим в (2). В результате получим уравнение траектории камня:

$$y = x \operatorname{tg}\alpha - x^2 \frac{g}{2v_0^2 \cos^2\alpha}, \tag{3}$$

представляющей собой параболу (рис. 1). Эта парабола пересекает ось x в двух точках: $x = 0$ и $x = l$, где

$$l = \frac{v_0^2}{g} \sin 2\alpha. \tag{4}$$

Первая точка является началом траектории, в ней камень вылетает из катапульты. Вторая точка соответствует месту падения камня на землю. Формула (4) в рамках принятой модели определяет искомое расстояние l. Эта формула вам хорошо известна: она выводится и подробно обсуждается в учебнике по физике для 8-го класса.

В прикладных задачах построение математической модели — это один из наиболее сложных и ответственных эта-

пов работы. Опыт показывает, что во многих случаях правильно выбрать модель — значит решить проблему более чем наполовину. Трудность данного этапа состоит в том

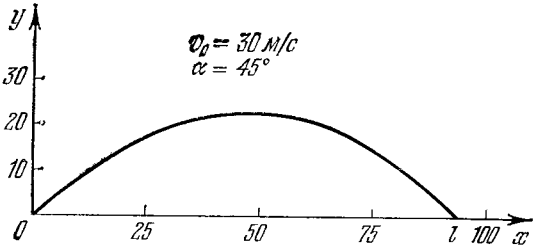

Рис. 1. Параболическая траектория движения камня.

что он требует соединения математических и специальных знаний. При решении школьных задач по физике вы выступаете одновременно как физики и математики. Однако для больших проблем, которые рассматриваются в прикладной математике, такое совмещение профессий не типично. Обычно над математической моделью совместно работают математики и специалисты из той области, к которой относится изучаемый объект. Для успеха их деятельности очень важно взаимопонимание, которое приходит тогда, когда математики обладают специальными знаниями об объекте, а их партнеры — определенной математической культурой, опытом применения математических методов исследования в своей области.

§ 2. Соответствие математической модели изучаемому объекту. Критерий практики

Математическая модель никогда не бывает тождественна рассматриваемому объекту, не передает всех его свойств и особенностей. Основанная на упрощении, идеализации, она является приближенным описанием объекта. Поэтому результаты, получаемые при анализе модели, всегда носят для объекта приближенный характер. Их точность определяется степенью соответствия, адекватности модели и объекта. Вопрос о точности, о достоверности результатов — это один из самых тонких вопросов прикладной математики.

Наиболее просто он решается в случае, когда хорошо известны законы, определяющие поведение и свойства объ-

екта, и имеется большой практический опыт их применения. Тогда можно априори (до опыта, здесь — до начала решения математической задачи) оценить точность результатов, которую обеспечивает рассматриваемая модель.

Приведем пример. В СССР 2 января 1959 г. был осуществлен запуск автоматической станции Луна-1, который открыл для человечества эру межпланетных полетов. Расчет траектории станции в межпланетном пространстве проводился на основании математической модели, использующей законы механики и закон всемирного тяготения. Многовековой опыт наблюдения за небесными телами в Солнечной системе показал, что такая модель очень точно описывает их движение. Универсальность законов природы служила гарантией применимости модели к космическому аппарату, созданному руками человека.

Более сложная ситуация возникает тогда, когда наши знания об изучаемом объекте недостаточны. В этом случае при построении математической модели приходится делать дополнительные предположения, которые носят характер *гипотез*. Выводы, полученные в результате исследования такой *гипотетической модели*, носят для изучаемого объекта условный характер. Они справедливы для него настолько, насколько правильны исходные предположения. Для их проверки необходимо сопоставить результаты исследования модели со всей имеющейся информацией об изучаемом объекте. Степень близости расчетных и экспериментальных данных позволяет судить о качестве гипотетической модели, о справедливости или ошибочности исходных предположений. Таким образом, вопрос применимости некоторой математической модели к изучению рассматриваемого объекта не является чисто математическим вопросом и не может быть решен математическими методами. Основным критерием истинности является эксперимент, практика в самом широком смысле этого слова. *Критерий практики* позволяет сравнить различные гипотетические модели и выбрать из них такую, которая является наиболее простой и в то же время в рамках требуемой точности правильно передает свойства изучаемого объекта.

Для иллюстрации этих соображений вернемся к задаче о траектории полета камня, выбрасываемого катапультой, и продолжим ее обсуждение. В § 1 мы построили математическую модель движения камня, основанную на четырех упрощающих предположениях, и получили формулу (4) для дальности броска. Теперь нам нужно оценить точность

этой формулы, установить пределы ее применимости. Для такого анализа не обязательно самому проводить прямые эксперименты с катапультой, заимствованной из музея или восстановленной по старым чертежам. По интересующим нас вопросам накоплен огромный экспериментальный и теоретический материал, так что нужно только умело воспользоваться им для анализа поставленной задачи.

Перечитайте еще раз упрощающие предположения, на основании которых была построена математическая модель движения камня, и вдумайтесь в их смысл. Пусть катапульта может метать камни на расстояние до 100 м, для чего она должна сообщить им скорость порядка 30 м/с. При этом камень поднимется на высоту 20—30 м и пробудет в воздухе около 5 с. В этих условиях первые три предположения выглядят совершенно оправданными, и нам нужно проанализировать четвертое предположение о влиянии воздуха. На всякое тело, движущееся в воздухе, он действует с некоторой силой $\boldsymbol{F}$. Ее модуль и направление зависят от формы тела и скорости движения. Силу $\boldsymbol{F}$ можно разложить на две составляющие: параллельную и перпендикулярную скорости движения тела $\boldsymbol{v}$.

Перпендикулярная составляющая возникает только при наличии асимметрии тела по отношению к направлению движения. Наиболее характерным ее проявлением является подъемная сила, действующая на крыло самолета, без которой была бы невозможна авиация. Для того чтобы эта сила могла оторвать самолет от земли и поддерживать его в воздухе, крылу придают специальную форму и располагают его под определенным углом атаки к набегающему воздушному потоку. Однако для камня, форма которого близка к сферической, перпендикулярная составляющая силы $\boldsymbol{F}$ мала, и ею можно пренебречь (для шара благодаря его симметрии она точно равна нулю).

Параллельная составляющая силы $\boldsymbol{F}$ возникает всегда. Она направлена в сторону, противоположную движению, и стремится затормозить тело; ее называют *лобовым сопротивлением*. Таким образом, в интересующем нас случае

$$F \approx F_{\text{л}}. \tag{5}$$

Модуль лобового сопротивления $\boldsymbol{F}_{\text{л}}$ можно представить в виде

$$F_{\text{л}} = CS\frac{\rho v^2}{2}, \tag{6}$$

где S — площадь поперечного сечения тела, ρ — плотность воздуха, v — скорость движения тела, C — безразмерный множитель, который называют *коэффициентом лобового сопротивления.*

Коэффициент лобового сопротивления зависит от формы тела и характеристики процесса обтекания, которую называют *числом Рейнольдса*:

$$C = C(\mathrm{Re}), \qquad \mathrm{Re} = \frac{v\rho d}{\mu}. \tag{7}$$

Здесь d — характерный размер тела, ρ и μ — плотность и вязкость воздуха ($\rho = 1{,}3$ кг/м^3, $\mu = 1{,}7 \cdot 10^{-5}$ кг/(м·с)). Из размерностей величин, входящих в (7), видно, что число Рейнольдса — безразмерная величина.

Оценим величину числа Рейнольдса в интересующем нас случае, когда $v = 30$ м/с, $d = 0{,}2$ м $= 20$ см:

$$\mathrm{Re} = \frac{30 \cdot 0{,}2 \cdot 1{,}3}{1{,}7 \cdot 10^{-5}} = 4{,}6 \cdot 10^5. \tag{8}$$

Экспериментальные и теоретические исследования показывают, что для шара, в широком диапазоне чисел Рейнольдса, включающих значение (8):

$$3 \cdot 10^5 \leqslant \mathrm{Re} \leqslant 7 \cdot 10^6, \tag{9}$$

коэффициент лобового сопротивления C очень слабо зависит от своего аргумента Re, и его можно считать постоянным:

$$C \approx 0{,}15. \tag{10}$$

Полагая в (6) $S = \pi R^2$ и принимая во внимание (10), получим простую формулу для модуля лобового сопротивления шара при условии (9):

$$F_л = \frac{C\pi}{2} R^2 \rho v^2. \tag{11}$$

Эта формула, в частности, указывает на квадратичную зависимость $F_л$ от скорости.

Для того чтобы оценить влияние сопротивления воздуха на характер движения, сравним его с основной силой в рассматриваемой задаче — с силой тяжести:

$$P = mg = \frac{4\pi}{3} R^3 \rho_0 g,$$

где ρ_0 — плотность камня ($\rho_0 = 2{,}3 \cdot 10^3$ кг/м^3). Заменим в

формуле (6) v^2 на lg и составим отношение $F_л/P$:

$$\frac{F_л}{P} = \frac{C\pi R^2 \rho\, lg/2}{4\pi R^3 \rho_0 g/3} = \frac{3C}{8} \cdot \frac{\rho l}{\rho_0 R}. \qquad (12)$$

При $l=100$ м, $R=0{,}1$ м, считая $C\approx 0{,}15$, получаем $F_л/P = 0{,}03$.

Обозначим через Δl абсолютную погрешность, которую мы допускаем в определении дальности броска l, пренебрегая сопротивлением воздуха. При малом сопротивлении воздуха ($F_л/P \ll 1$) относительная погрешность $\Delta l/l$ пропорциональна $F_л/P$ с коэффициентом пропорциональности порядка единицы. Таким образом, в рассматриваемом диапазоне параметров получаем $\Delta l/l \sim 2$—3% и $\Delta l \sim 2$—3м. Если не требуется высокая точность и такая ошибка допустима, то применение модели оправдано. В противном случае ее следует заменить более сложной моделью, учитывающей сопротивление воздуха.

В заключение подчеркнем, то наша оценка рассмотренной математической модели не является абсолютной: модель годится или не годится. Проведенный анализ дал возможность установить для нее условия применимости, связав их с диапазоном изменения параметров задачи и требуемой точностью. Он также позволил наметить путь уточнения модели в случае, когда условия ее применимости перестают выполняться.

§ 3. Развитие и уточнение математической модели

Исследование прикладных задач обычно начинается с построения и анализа простейшей, наиболее грубой математической модели рассматриваемого объекта. (Характерным примером может служить модель параболической траектории полета тела, получившего на поверхности Земли начальную скорость $\boldsymbol{v}_0$.) Однако в дальнейшем часто возникает необходимость уточнить модель, сделать ее соответствие объекту более полным. Это может быть обусловлено разными причинами: требованием более высокой точности, появлением новой информации об объекте, которую нужно отразить в математической модели, расширением диапазона параметров, выводящим за пределы применимости исходной модели, и т. д. При построении новой модели полезно максимально полно использовать опыт и результаты, полученные на первом этапе. Часто процесс последовательного развития и уточнения модели повторяется неоднократно.

Для иллюстрации этих соображений вернемся снова к задаче о движении тела, брошенного с поверхности Земли, и рассмотрим ее применительно к *внешней баллистике*. Так называют науку о движении снаряда, вылетевшего из ствола орудия. Мы выбрали баллистику не только потому, что ее задачи интересны с математической и важны с практической точек зрения. Не менее существенна другая сторона вопроса: на этом примере мы сможем наглядно показать процесс постепенного совершенствования и уточнения математической модели рассматриваемого явления, который исторически продолжался более 300 лет.

Древние воины, использовавшие катапульты для разрушения неприятельских укреплений, не знали законов механики и не могли теоретически рассчитать траекторию полета камня даже в рамках очень простой модели. Впрочем, это и не было нужно. Положение изменилось с изобретением пороха и появлением артиллерии, существенно увеличившей дальность, интенсивность и эффективность обстрела.

Исследования по внешней баллистике были начаты в XVII в. Г. Галилеем, который разработал теорию параболического движения снаряда, рассмотренную в § 1. Следующий шаг связан с именем И. Ньютона. В своем основном труде «Математические начала натуральной философии» (1687 г.) он предпринял попытку решить задачу баллистики с учетом сопротивления воздуха. Этот шаг был совершенно необходим: из результатов § 2 ясно, что, несмотря на несовершенство орудий XVII в., воздух должен был оказывать заметное влияние на движение ядер, вызывая отклонение их траектории от параболы (3). Действительно, проведем с помощью (12) оценку роли сопротивления воздуха для чугунного ядра, полагая $l=1$ км$=10^3$ м, $R=0{,}07$ м, $\rho_0=7\cdot 10^3$ кг/м^3 (плотность чугуна), $C=0{,}15$. При указанных значениях параметров $F_л/P=0{,}15$, т. е. пренебрежение сопротивлением воздуха приводит к ошибке в определении дальности l порядка 15% ($\sim$150 м).

Рассмотрим математическую модель баллистики пушечного ядра, учитывающую сопротивление воздуха. При ее построении мы по-прежнему будем считать справедливыми первые три предположения предыдущей модели, а четвертое предположение переформулируем следующим образом:

4) воздух действует на ядро при его движении с силой $\boldsymbol{F}_л$, ее модуль определяется формулой (11), а направление противоположно направлению скорости.

Учет сопротивления воздуха существенно усложняет задачу. Теперь уже она не допускает решения в виде простых аналитических формул и может быть решена только с помощью численных методов. На рис. 2, 3 в качестве примера приведены результаты расчетов двух траекторий.

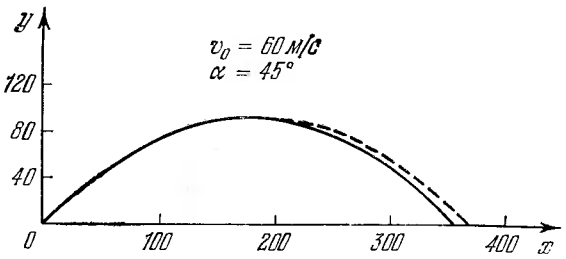

Рис. 2. Влияние сопротивления воздуха на движение пушечного ядра, вылетающего с начальной скоростью 60 м/с. Сплошная линия — траектория, рассчитанная с учетом сопротивления воздуха, штриховая — без учета.

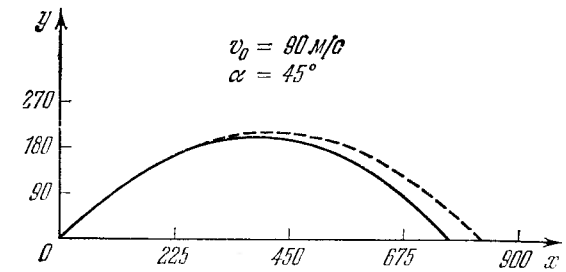

Рис. 3. Влияние сопротивления воздуха на движение пушечного ядра, вылетающего с начальной скоростью 90 м/с. Сплошная линия — траектория, рассчитанная с учетом сопротивления воздуха, штриховая — без учета.

В обоих случаях предполагалось, что чугунное ядро радиуса $R=0{,}07$ м вылетает под углом $\alpha=45°$ к поверхности Земли. Начальная скорость v_0 на рис. 2 принималась равной 60 м/с, на рис. 3—90 м/с. Штриховыми линиями даны параболические траектории (3) в модели § 1. Сравнение сплошных и штриховых линий наглядно показывает влияние сопротивления воздуха на движение ядра при различных начальных скоростях.

Переход в XIX в. от гладкоствольного к нарезному оружию позволил существенно увеличить дальность и точность стрельбы. Появилась возможность вести огонь с закрытых

позиций с помощью приборов, не видя непосредственно цели. Это потребовало совершенствования прицельных приспособлений, поставило новые задачи перед баллистикой и привело к дальнейшему уточнению используемых математических моделей.

Наиболее характерной особенностью баллистики снарядов нарезного оружия является сверхзвуковая скорость их движения. При таких скоростях около снаряда образуется волна сильного сжатия, которая расходится от него в виде конуса (так называемого *конуса Маха*) с углом раствора $\varphi=\arcsin(c/v)$, где v — скорость снаряда, c — скорость звука (при нормальных условиях $t=15°C$, $p=760$ мм рт. ст., $c=340$ м/с). Эта волна хорошо видна на рис. 4, где приведено изображение снаряда, движущегося со скоростью $v=2{,}48\ c$.

Лобовое сопротивление при сверхзвуковом движении является, в первую очередь, волновым. Оно связано с тем, что снаряд должен непрерывно расходовать свою кинетическую энергию на образование волны. Для его уменьшения снаряду придают специальную форму вытянутого тела вращения с заостренным носом (см. рис. 4). Модуль лобового сопротивления, как и в предыдущем случае, удобно представить в виде (6). Однако для сверхзвуковых скоростей ($v>c$) коэффициент C не будет постоянным. Он оказывается функцией скорости $C=C(v)$, резко возрастающей при увеличении скорости. Это сильно усложняет задачу расчета траектории снаряда.

Рис. 4. Волна сжатия вокруг снаряда, движущегося со сверхзвуковой скоростью.

До сих пор мы предполагали, что все величины, которые входят в условия задачи баллистики, известны точно. На самом деле это не так. Снаряды нельзя изготовить абсолютно одинаковыми: они немного отличаются друг от друга по весу, по величине порохового заряда и при выстреле получают разные начальные скорости. Невозможно дважды выстрелить под одним и тем же углом возвышения α. Эти и целый ряд других неконтролируемых факторов приводят к тому, что два снаряда, выпущенные из орудия при одинаковых, на наш взгляд, условиях, никогда не попадут в одну и ту

же точку: из-за действия случайных факторов снаряды рассеиваются. Это означает, что любое предсказание баллистического расчета носит не строго определенный, детерминированный характер, а является вероятностным. В конце XIX — начале XX в. в математических моделях баллистики начали использовать методы теории вероятности.

Сложность задач баллистики делает невозможным их решение непосредственно на поле боя. В помощь артиллеристам стали создаваться специальные таблицы, получившие название «таблиц стрельбы». В них для определенного типа орудий и снарядов (т. е. для заданной начальной скорости v_0) приводились основные характеристики траекторий при разных углах возвышения α: дальность l, максимальная высота h, время полета снаряда T. Такие таблицы были уже широко распространены к началу первой мировой войны.

Дальнейшее увеличение скорости снарядов и дальности стрельбы привело к необходимости внести в математическую модель баллистики еще одно уточнение — учет вращения Земли вокруг своей оси. Тем самым пришлось отказаться от предположения, что Земля — инерциальная система отсчета. Анализ показывает, что из-за вращения Земли всякое тело должно отклоняться от направления своего движения в Северном полушарии немного вправо, в Южном — влево. С этим эффектом связаны, в частности, два хорошо известных явления: у наших рек правый берег обычно подмывается сильнее левого, а на двухколейных железных дорогах быстрее снашивается правый рельс по ходу движения поезда.

Мы не будем останавливаться на деталях математического описания данного эффекта и его включения в общую модель баллистики снаряда. Приведем только один поучительный эпизод из истории первой мировой войны.

В 1914 г. около Фолклендских островов, которые расположены в Атлантическом океане вблизи побережья Южной Америки (51° S, 60° W), произошел морской бой между английской и немецкой эскадрами, закончившийся победой англичан. Однако в начале боя залпы английских кораблей систематически ложились метров на 100 левее цели и потребовалась специальная корректировка в установках прицела, чтобы устранить отклонение. Произошло это по следующей причине. Прицельные приспособления на английских кораблях автоматически вносили поправку на вращение Земли. Однако она была рассчитана на средние широты северного полушария (50° N), бой же произошел на той же

широте Южного полушария, где отклонение снарядов из-за вращения Земли должно иметь противоположный знак. В результате коррекция, вносимая прицельным устройством, не компенсировала смещение, а удваивала его, и сдвиг достиг величины порядка 100 м.

Заканчивая знакомство с математическими задачами баллистики, остановимся еще на одном событии первой мировой войны. В ходе войны немцы сумели подойти к Парижу на расстояние меньше 100 км, но у них не хватило сил овладеть столицей Франции. Тогда командование германской армии приняло решение подвергнуть Париж артиллерийскому обстрелу. Для этой цели специально были изготовлены сверхдальнобойные орудия «Колоссаль». Они представляли собой огромные установки (длина ствола — 34 м, калибр — 210 мм), которые монтировались на специальной платформе и имели дальность стрельбы свыше 100 км. Снаряд посылался под фиксированным углом возвышения $\alpha=52{,}5°$, круто уходил вверх и основную часть пути летел на высоте, превышающей 20 км, практически не испытывая сопротивления воздуха. Дальность стрельбы регулировалась размером порохового заряда, масса которого доходила до 200 кг. Ствол из-за своей огромной длины легко деформировался и выдерживал небольшое число выстрелов.

Стрельба на такие большие расстояния потребовала сложных баллистических расчетов на основе усовершенствованной математической модели. В ней учитывались зависимость плотности воздуха и ускорения свободного падения от высоты, топогеодезические данные о местности, включающие кривизну земной поверхности (Земля в задачах баллистики перестала быть «плоской»), метеорологические данные о скорости и направлении ветра, давлении воздуха и т. д. Несмотря на тщательную подготовку каждого выстрела, наблюдалось большое рассеивание снарядов: из 303 снарядов, выпущенных за время войны по Парижу, 120 упало за его пределами. Существенного влияния на ход первой мировой войны обстрел Парижа не оказал.

Подведем итоги обсуждения математических моделей задач баллистики. Оно началось с простейшей модели Галилея, которая основана на предположениях, сильно упрощающих задачу. Такая модель справедлива в очень узком диапазоне начальных скоростей: $v_0 \leqslant 30$ м/с.

Затем была рассмотрена модель, учитывающая сопротивление воздуха. При этом предполагалось, что коэффициент, лобового сопротивления в формуле (6) является по-

стоянным ($C \approx 0{,}15$). Эта модель справедлива в диапазоне дозвуковых скоростей: $v_0 \leqslant 250$—300 м/с.

Следующая модель позволила перейти к сверхзвуковым скоростям. Однако для ее реализации необходимо знать зависимость коэффициента лобового сопротивления рассматриваемого тела от скорости.

Далее в модель были также включены эффект вращения Земли, учет сферической формы ее поверхности и зависимости g от высоты, метеорологических данных, случайных факторов, которые приводят к рассеянию снарядов и придают изучаемым закономерностям статистический характер. В результате была построена модель баллистики, применимая во всем диапазоне скоростей, которые были достигнуты с помощью огнестрельного оружия. Подчеркнем, что в ней оказались пересмотренными все исходные упрощающие предположения модели Галилея.

Каждая модель, применявшаяся в баллистике, проходила проверку практикой. С появлением нарезного оружия, существенно увеличившего дальность и точность стрельбы, начали играть важную роль полигонные испытания. На них проверялись и оружие, и баллистические расчеты.

В 50-х годах с появлением ракет различного назначения возникли задачи ракетной баллистики. Запуск 4 октября 1957 г. первого искусственного спутника Земли и выход 2 января 1959 г. станции Луна-1 в межпланетное пространство потребовали разработки специфических проблем космической баллистики.

На рис. 5 в качестве примера приведена траектория советской автоматической межпланетной станции Луна-2, которая позволила заглянуть в неизвестное: сделать первые снимки обратной стороны Луны и передать их на Землю. Станция была запущена 4 октября 1959 г. Она имела только микродвигатели, предназначенные для ее ориентации и наведения фотообъективов на Луну. Коррекция траектории не проводилась. Осуществление такого уникального для своего времени космического полета потребовало высочайшей точности вывода станции на орбиту.

Дальнейший прогресс в космических исследованиях был основан на применении автоматических и пилотируемых аппаратов, снабженных собственными бортовыми двигателями. Благодаря им космические аппараты, выведенные на орбиту ракетой-носителем, перестали быть пассивными «пленниками» гравитационных сил: появилась возможность целенаправленного изменения их траекторий.

Всякое управляемое изменение траектории принято называть в космической баллистике *маневром*. Примерами маневров могут служить переход с одной орбиты искусственного спутника Земли на другую, стыковка, возвращение

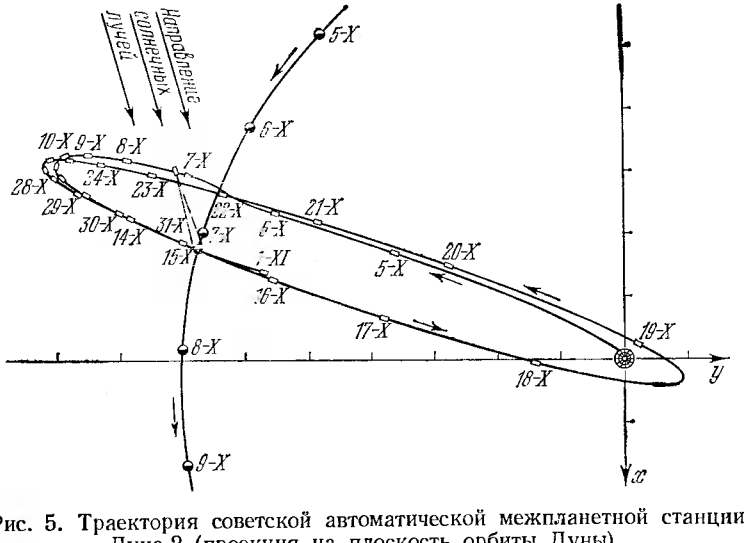

Рис. 5. Траектория советской автоматической межпланетной станции Луна-2 (проекция на плоскость орбиты Луны).

аппарата с орбиты на Землю, коррекция траектории при полете к другим планетам, изменение траектории при подлете к какой-нибудь планете с целью перехода на орбиту ее искусственного спутника, мягкая посадка. Маневр — это сложная и ответственная операция, от успеха которой обычно существенно зависит выполнение всей намеченной программы полета. Расчет маневра и управление его осуществлением выполняется с помощью ЭВМ. Космические исследования базируются на новейших достижениях во многих областях человеческих знаний. В частности, они были бы невозможны без современных вычислительных машин.

В заключение подчеркнем еще раз, что математические модели позволяют свести исследование реального «нематематического» объекта к решению математической задачи, открывая тем самым возможность использования для его изучения хорошо разработанного математического аппарата в сочетании с мощной вычислительной техникой. На этом основано применение математики для познания законов реального мира и их использования в практике.

ГЛАВА 2

ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

Построение модели рассматриваемого объекта позволяет поставить задачу его изучения как математическую. После этого наступает второй этап исследования — поиск метода решения сформулированной математической задачи. Следует иметь в виду, что в прикладных работах нас, как правило, интересуют количественные значения искомых величин, т. е. ответ должен быть доведен «до числа». Все расчеты проводятся с числами, записанными в виде конечных десятичных дробей, поэтому результаты вычислений всегда носят приближенный характер. Стало быть, важно добиться того, чтобы ошибки укладывались в рамки требуемой точности.

§ 1. Понятие алгоритма

В большинстве задач, с которыми вы встречались до сих пор в математике, ответ давался в виде формулы. Формула определяла последовательность математических операций, которую нужно выполнить для вычисления искомой величины. Например, формула корней квадратного уравнения позволяет найти их по значениям коэффициентов этого уравнения, формула Герона выражает площадь треугольника через длины его сторон и т. д.

Однако вам известны задачи, для которых ответ легко может быть найден, хотя он и не записывается в виде формулы. Вспомните младшие классы, когда вы учили целые числа и арифметические операции над ними. Можно ли назвать «формулой» правило вычисления суммы нескольких чисел с помощью поразрядного сложения столбиком? В то же время это правило полностью решает поставленную задачу: оно определяет последовательность математических операций, которую нужно выполнить для вычисления искомой величины.

Рассмотрим еще один известный вам пример — задачу отыскания наибольшего общего делителя (НОД) двух целых чисел n_1, n_2 (для определенности предположим, что $n_1 > n_2$). Общей формулы для решения этой задачи, которая выражала бы НОД через заданные числа n_1, n_2, не существует. Однако можно указать универсальные методы, которые позволяют найти НОД любых двух целых чисел.

Один из таких методов заключается в последовательном переборе чисел n_2, n_2-1, n_2-2 и т. д. Процесс продолжается до тех пор, пока мы не обнаружим число, являющееся одновременно делителем n_1 и n_2. Такой подход всегда приведет к решению поставленной задачи, хотя вряд ли нужно доказывать, что он является трудоемким и неэффективным.

Рассмотрим другой, более интересный метод решения той же задачи. Разделим n_1 на n_2 в целых числах с остатком. Пусть при таком делении получены частное r_1 и остаток n_3 ($0 \leqslant n_3 < n_2$). Это означает, что число n_1 представимо в виде

$$n_1 = r_1 n_2 + n_3. \tag{1}$$

Если остаток n_3 равен нулю, то задача решена: число n_2 — НОД пары чисел n_1, n_2. Если $n_3 \neq 0$, из (1) вытекает следующее утверждение: всякий общий делитель чисел n_1, n_2 является одновременно общим делителем чисел n_2, n_3, и наоборот. Отсюда, в частности, следует, что НОД чисел n_1, n_2 равен НОД чисел n_2, n_3.

Будем теперь искать НОД чисел n_2, n_3. Для этого снова разделим n_2 на n_3 в целых числах с остатком. Пусть при этом получился остаток n_4 ($0 \leqslant n_4 < n_3 < n_2$). Если он равен нулю, то искомый НОД равен n_3. В противном случае заменим пару чисел n_2, n_3 на пару чисел n_3, n_4 и продолжим процесс. В результате после конечного числа шагов, не превышающего n_2, мы придем к решению рассматриваемой задачи.

Итак, мы видим, что для решения математической задачи важно указать систему правил, которая задает строго определенную последовательность математических операций, приводящих к искомому ответу. Такую систему правил называют *алгоритмом*. Понятие алгоритма в его общем виде относится к числу основных понятий математики.

В простейшем случае последовательность математических операций, с помощью которых можно вычислить искомые величины, определяется формулами. Так, формула Герона является алгоритмом вычисления площади треуголь-

ника по его сторонам. Однако описанный выше метод нахождения НОД дает пример алгоритма, который не сводится к формуле. Его называют *алгоритмом Евклида*.

Алгоритмы решения многих математических задач, для которых не удается получить ответ в виде формулы, основаны на следующей процедуре: строится бесконечный процесс, сходящийся к искомому решению. Он обрывается на некотором шаге (вычисления нельзя продолжать бесконечно), и полученная таким образом величина приближенно принимается за решение рассматриваемой задачи. Сходимость процесса гарантирует, что для любой заданной точности $\varepsilon > 0$ найдется такой номер шага N, что на этом шаге ошибка в определении решения задачи не превысит ε.

Таблица 1

k	$n=6\cdot 2^k$	p_n	q_n
0	6	3, 000 000 000 000	3, 464 101 615 138
1	12	3, 105 828 541 230	3, 630 002 002 236
2	24	3, 132 628 613 281	3, 245 155 564 194
3	48	3, 139 350 203 047	3, 166 557 423 678
4	96	3, 141 031 950 890	3, 147 778 817 495
5	192	3, 141 452 472 285	3, 143 135 797 312
6	384	3, 141 557 607 912	3, 141 978 227 840
7	768	3, 141 583 892 148	3, 141 689 033 932
8	1 536	3, 141 590 463 228	3, 141 616 747 849
9	3 072	3, 141 592 105 999	3, 141 598 677 103
10	6 144	3, 141 592 516 692	3, 141 594 159 465
11	12 288	3, 141 592 619 365	3, 141 593 030 058
12	24 576	3, 141 592 645 034	3, 141 592 747 706
13	49 152	3, 141 592 651 034	3, 141 592 677 119
14	98 304	3, 141 592 653 055	3, 141 592 659 472
15	196 608	3, 141 592 653 456	3, 141 592 655 060
16	393 216	3, 141 592 653 556	3, 141 592 653 957

Для иллюстрации этого подхода к решению математических задач рассмотрим два простых примера. Первый пример связан с задачей о длине окружности и вычислении числа π. На подробном разборе задачи мы останавливаться не будем, поскольку она хорошо известна из средней школы. Приведем лишь табл. 1, в которой приводятся результаты расчетов периметров правильных многоугольников, вписанных в окружность с диаметром $d=1$ и описанных вокруг нее. Таблица начинается с известных периметров вписанного

и описанного шестиугольников: $p_6 = 3$, $q_6 = 2\sqrt{3}$. Далее периметры многоугольников рассчитывались по этим данным с помощью формулы удвоения. Последняя строка таблицы соответствует вписанному и описанному многоугольникам с числом сторон $n = 6 \cdot 2^{16} = 393\,216$ (результат 16-кратного удвоения числа сторон).

С ростом n периметры p_n вписанных многоугольников растут, а периметры q_n описанных многоугольников убывают, стремясь в пределе к длине окружности:

$$\lim_{n \to \infty} p_n = \lim_{n \to \infty} q_n = \pi.$$

Таким образом, периметры p_n определяют число π с недостатком, q_n — с избытком:

$$p_n < \pi < q_n. \tag{2}$$

Двусторонняя оценка (2) позволяет легко контролировать точность на каждом шаге вычислений. Составляя разность периметров q_n и p_n, приведенных в последней строке табл. 1, получим

$$\varepsilon_n < \Delta_n = q_n - p_n = 0{,}000\,000\,000\,401.$$

Первые 10 знаков чисел p_n и q_n в этой строке совпадают. Они дают первые 10 знаков числа

$$\pi = 3{,}141\,592\,653\ldots$$

Заканчивая обсуждение этого вопроса, приведем некоторые факты из истории вычисления числа π. Великий древнегреческий ученый Архимед, используя формулу удвоения, дошел до правильного 96-угольника и получил следующую двустороннюю оценку π:

$$3\frac{10}{71} < \pi < 3\frac{1}{7}$$

$$\left(3\frac{10}{71} = 3{,}14084\ldots,\ 3\frac{1}{7} = 3{,}14285\ldots,\ \Delta \approx 0{,}002\right).$$

Чтобы правильно понять значение этого результата, нужно иметь в виду, что в то время не было привычной для нас позиционной записи чисел, десятичных дробей и хорошо разработанной техники извлечения квадратных корней, хотя такую операцию Архимеду приходилось выполнять на каждом шаге два раза.

В первой половине XV в. в знаменитой обсерватории хана Улугбека под Самаркандом его придворный астроном

Аль-Каши вычислил π с 17 знаками после запятой. Он сделал 27 удвоений числа сторон и дошел до $6 \cdot 2^{27}$-угольника. Это был уникальный для своего времени расчет. Вычисления Аль-Каши были связаны с составлением таблицы синусов с шагом в $1'$, нужной для астрономических наблюдений. Для сравнения укажем, что в Европе французский математик Ф. Виет (вы знаете его теорему о корнях квадратного уравнения) в 1593 г., т. е. спустя 150 лет после Аль-Каши, нашел лишь 9 цифр числа π после запятой с помощью периметра $6 \cdot 2^{16}$-угольника (16 удвоений числа сторон). Только на рубеже XVI и XVII в. результат Аль-Каши был повторен, а затем и превзойден.

К концу XIX в. английский математик В. Шенкс вычислил 707 знаков числа π, затратив на это более 20 лет. Такой результат по праву получил славу рекорда вычислений XIX в. Однако в 1945 г. было обнаружено, что В. Шенкс допустил ошибку в 520-м знаке и все его последующие вычисления пошли насмарку. В настоящее время с помощью ЭВМ число π вычислено с фантастической точностью — более 500 тыс. знаков. Продолжительность расчетов подобного типа зависит от используемого алгоритма и быстродействия ЭЗМ. Обычно она измеряется несколькими часами.

В качестве второго примера рассмотрим задачу об извлечении квадратного корня из произвольного положительного числа a. Эта задача может быть решена разными способами. Мы опишем алгоритм, основанный на построении монотонной последовательности, сходящейся к $\sqrt{a}$.

Выберем в качестве x_0 произвольное положительное число и рассмотрим последовательность $\{x_n\}$, определенную с помощью *рекуррентной формулы* *):

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \quad n = 0, 1, 2, \ldots, \tag{3}$$

где a — положительное число, из которого нужно извлечь

---

*) *Рекуррентная формула* (от латинского recurrens — возвращающийся) — формула, позволяющая выразить $(n+1)$-й член последовательности через значения ее первых n членов. В данном случае мы имеем простейшую рекуррентную формулу, в которой $(n+1)$-й член выражается прямо через n-й. При наличии рекуррентной формулы последовательность полностью определяется выбором нулевого члена x_0. Способ задания последовательностей с помощью рекуррентных формул является очень распространенным. Он прост и удобен для расчетов на ЭВМ.

квадратный корень. Анализ этой последовательности позволяет установить следующие свойства.

1. Члены последовательности $\{x_n\}$ при $n=1, 2, \ldots$ удовлетворяют неравенству

$$x_n \geqslant \sqrt{a}. \tag{4}$$

Перепишем (3) в виде

$$x_n = \sqrt{a}\,\frac{1}{2}\left(\frac{x_{n-1}}{\sqrt{a}} + \frac{\sqrt{a}}{x_{n-1}}\right) = \sqrt{a}\,\frac{1}{2}\left(t_{n-1} + \frac{1}{t_{n-1}}\right),$$

$$n=1, 2, \ldots,$$

где $t_{n-1} = \frac{x_{n-1}}{\sqrt{a}}$. Функция $f(t) = \frac{1}{2}\left(t + \frac{1}{t}\right)$ при любом положительном t удовлетворяет неравенству $f(t) \geqslant 1$. Отсюда получаем (4).

2. Члены последовательности $\{x_n\}$ при $n=1, 2, \ldots$ не возрастают, т. е.

$$x_{n+1} \leqslant x_n, \qquad n=1, 2, \ldots \tag{5}$$

Разделив (3) на x_n и воспользовавшись неравенством $a/x_n^2 \leqslant 1$, получим

$$\frac{x_{n+1}}{x_n} = \frac{1}{2}\left(1 + \frac{a}{x_n^2}\right) \leqslant 1,$$

т. е. неравенство (5).

Итак, мы видим, что при $n=1, 2, \ldots$ рекуррентная последовательность (3) — монотонно невозрастающая (5) и ограниченная снизу (4). Следовательно, по теореме Вейерштрасса о монотонных последовательностях она имеет предел:

$$\lim_{n\to\infty} x_n = c. \tag{6}$$

Учитывая это, перейдем в (3) к пределу при $n\to\infty$. В результате получим $c = \frac{1}{2}\left(c + \frac{a}{c}\right)$, или

$$c = \sqrt{a}. \tag{7}$$

Итак, мы доказали, что при выборе в качестве нулевого приближения x_0 любого положительного числа рекуррентная последовательность (3) сходится к $\sqrt{a}$:

$$\lim_{n\to\infty} x_n = \sqrt{a}, \tag{8}$$

монотонно приближаясь к своему пределу сверху.

Сходимость (8) означает, что для любой заданной точности $\varepsilon>0$ можно указать такой номер N, что член последовательности x_N, а также все дальнейшие члены удовлетворяют неравенству

$$0 \leqslant x_N - \sqrt{a} < \varepsilon, \quad \text{или} \quad \sqrt{a} \leqslant x_N < \sqrt{a} + \varepsilon,$$

т. е. x_N определяет $c=\sqrt{a}$ с ошибкой меньше ε.

О достигнутой точности на n-м шаге можно судить по следующей оценке:

$$x_n - \sqrt{a} = \frac{x_n^2 - a}{x_n + \sqrt{a}} < \frac{x_n^2 - a}{2c_0}, \tag{9}$$

где c_0 — некоторое число, удовлетворяющее неравенству $0 < c_0 \leqslant c = \sqrt{a}$. Практически за c_0 удобно взять грубое приближенное значение $\sqrt{a}$ по недостатку.

Число *итераций**) N, необходимое для достижения точности ε, зависит как от требуемой точности, так и от близости нулевого приближения к искомому корню c.

Для оценки скорости сходимости метода положим $x_n = \sqrt{a} + \delta_n$ ($\delta_n = x_n - \sqrt{a} \geqslant 0$ при $n=1, 2, \ldots$) и подставим в (3):

$$\sqrt{a} + \delta_{n+1} = \frac{1}{2}\left(\sqrt{a} + \delta_n + \frac{a}{\sqrt{a} + \delta_n}\right),$$

или

$$\delta_{n+1} = \frac{\delta_n^2}{2(\sqrt{a} + \delta_n)} \leqslant \frac{\delta_n^2}{2\sqrt{a}}. \tag{10}$$

Такая рекуррентная оценка погрешности говорит о высокой скорости сходимости процесса.

Для иллюстрации данного метода вычислим $\sqrt{81725{,}3}$. За нулевое приближение примем $x_0=300$. Это — значение корня с избытком, которое получается при замене подкоренного числа на 90000.

Нулевое приближение выбрано достаточно близким к корню: его погрешность δ_0 не превышает 15. По формуле (10) легко получить оценки погрешностей следующих при-

---

*) *Итерация* (от латинского iteratio — повторение) — результат неоднократного применения какой-нибудь математической операции, в данном случае вычислений по рекуррентной формуле (3).

ближений:

$$\delta_1 < \frac{225}{2\cdot 250} < 0{,}5, \qquad \delta_2 < \frac{0{,}25}{2\cdot 250} = 0{,}0005$$

(для упрощения оценок $\sqrt{a}$ в знаменателе формулы (10) заменим на меньшее число: 250). Мы видим, что уже второе приближение дает весьма высокую точность.

После этого анализа выпишем несколько первых приближений:

$$x_1 = 286{,}203\,833\,334, \qquad x_4 = 285{,}876\,371\,881,$$
$$x_2 = 285{,}876\,564\,976, \qquad x_5 = 285{,}876\,371\,881.$$
$$x_3 = 285{,}876\,371\,881,$$

Обратите внимание на то, что после третьего шага числа x_n перестали изменяться, процесс «останавливается». В этом проявляется принципиальная особенность вычислений с конечным числом значащих цифр. Когда на третьем шаге мы достигли погрешности, не превышающей 10^{-9}, то при расчетах с девятью знаками после запятой становится невозможно уловить разницу между x_{n+1} и x_n, лежащую за пределами ошибки округления. Чтобы продолжить расчет дальше и получить значение корня с более высокой степенью точности, нужно перейти к вычислениям с большим числом значащих цифр.

Мы рассмотрели два примера, которые показывают, как бесконечный сходящийся процесс может быть использован для приближенного решения математической задачи. Пусть слова «приближенное решение» вас не пугают, не воспринимайте их как «решение второго сорта». Если вы найдете в какой-нибудь задаче ответ в виде формулы и захотите подсчитать по ней численное значение искомой величины, то из-за представления чисел при вычислениях конечными десятичными дробями все равно получите для нее приближенное значение. Напомним, наконец, что в прикладных исследованиях решение математической задачи принципиально дает лишь приближенную информацию относительно изучаемого объекта, о чем подробно говорилось в главе 1.

Проблема применения алгоритмов, использующих бесконечный сходящийся процесс,— не в приближенном характере ответа, а в большом объеме необходимых вычислений. Не случайно такие алгоритмы принято называть *вычислительными алгоритмами*, а основанные на них методы решения математических задач — *численными методами*. Широкое применение вычислительных алгоритмов стало возмож-

ным благодаря ЭВМ. До их появления численные методы использовались редко и только в сравнительно простых случаях в силу чрезвычайной трудоемкости вычислений вручную.

В заключение сделаем несколько замечаний общего характера:

1) При разработке вычислительных алгоритмов особенное внимание уделяется тому, чтобы они были удобны для машинного счета.

2) Опыт показывает, что гораздо выгоднее развивать универсальные алгоритмы для решения широкого класса типичных математических задач, чем строить частные алгоритмы для решения каждой задачи в отдельности.

3) Изучение объектов самой различной природы часто приводит к одним и тем же математическим задачам. Поэтому имеется благоприятная возможность выделить задачи, которые часто встречаются в приложениях, изучить их особенности, разработать эффективные алгоритмы и реализовать эти алгоритмы в виде стандартных программ для ЭВМ.

В следующих параграфах этой главы мы продолжим знакомство с вычислительными алгоритмами, рассматривая хорошо известную из средней школы задачу — решение уравнений. Уравнения сыграли важную роль в истории математики, в развитии ее идей и методов. В то же время они и сегодня представляют большой интерес, поскольку часто встречаются в теоретических и прикладных задачах. Выбранные для обсуждения алгоритмы решения уравнений основаны на разных идеях, каждый из них обладает определенными достоинствами. Поэтому в конце главы будет интересно сравнить их между собой и обсудить на этом материале ряд общих вопросов, связанных с численным решением математических задач.

§ 2. Уравнения. Теорема о существовании корня непрерывной функции

В школьном курсе математики изучаются линейные и квадратные уравнения, корни которых могут быть найдены по известным формулам. Существуют также формулы для решения уравнений третьей и четвертой степеней, однако они очень сложны и неудобны для практического применения. Мы не будем приводить этих формул и останавливаться на их обсуждении, чтобы не отвлекаться от основной цели разговора.

Методы решения линейных и квадратных уравнений были известны еще древним грекам. Решение уравнений третьей и четвертой степеней было получено усилиями итальянских математиков Ш. Ферро, Н. Тартальи, Дж. Кардано, Л. Феррари в XV в. в эпоху Возрождения. Затем наступила пора поиска формул для корней уравнений пятой и более высоких степеней. В них принимали участие многие крупнейшие математики. Настойчивые, но безрезультатные попытки продолжались около 300 лет и завершились в 20-х годах XIX в. благодаря работам норвежского математика Н. Абеля. Он доказал, что общее уравнение пятой и более высоких степеней неразрешимо в радикалах. Решения общего уравнения n-й степени

$$a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n=0, \quad a_0\neq 0, \qquad (11)$$

при $n\geqslant 5$ нельзя выразить через коэффициенты с помощью арифметических действий и извлечения корней.

Если мы будем рассматривать неалгебраические уравнения, то задача еще больше усложнится. В этом случае найти для корней явные выражения, за редким исключением, не удается.

Возьмем в качестве примера очень простое уравнение

$$x=\cos x. \qquad (12)$$

Построим графики функций, стоящих в левой и правой части. Как видно из рис. 6, они пересекаются при $x=c$ $(0<c<1)$. Число c является корнем уравнения (12), однако получить для него формулу невозможно.

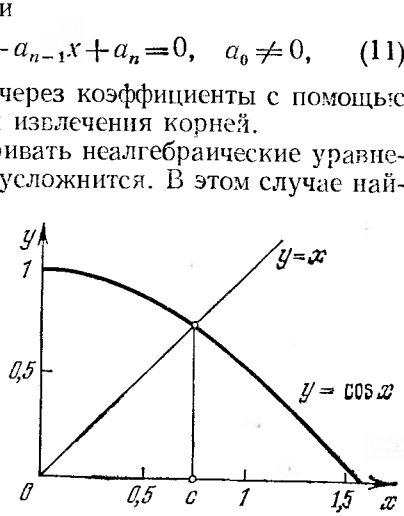

Рис. 6. Графическое решение уравнения $y=\cos x$.

В условиях, когда формулы «не работают», когда рассчитывать на них можно только в самых простейших случаях, важное значение приобретают универсальные вычислительные алгоритмы. Известен целый ряд алгоритмов решения рассматриваемой математической задачи. Если записать уравнение в виде

$$f(x)=0, \qquad (13)$$

то эти алгоритмы обычно не накладывают никаких ограни-

чений на конкретный вид функции $f(x)$, а предполагают только, что она обладает некоторыми свойствами типа непрерывности, дифференцируемости и т. д. Мы познакомимся с тремя алгоритмами. Однако прежде чем перейти к их описанию и обоснованию, рассмотрим некоторые общие вопросы качественного исследования уравнений.

Часто при решении уравнений важно знать заранее, имеет ли оно корни, и если имеет, то где они, примерно, располагаются. Рассмотрим квадратное уравнение

$$ax^2+bx+c=0. \tag{14}$$

Если подсчитать его дискриминант $\delta=b^2-4ac$ и убедиться, что он положителен, то можно сделать следующий вывод: уравнение (14) имеет два действительных корня: один из них лежит левее точки $x_0=-b/(2a)$, другой — правее.

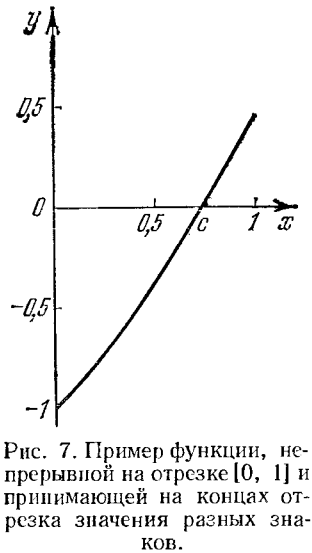

Рис. 7. Пример функции, непрерывной на отрезке [0, 1] и принимающей на концах отрезка значения разных знаков.

Этот случай тривиален — наш вывод основан на формуле для корней, т. е. на известном решении задачи. Гораздо важнее научиться проводить исследование уравнений, не имея под рукой готового ответа.

Посмотрите на рис. 7. На нем изображен график некоторой функции $f(x)$, непрерывной на отрезке $[0, 1]$ и принимающей на концах отрезка значения разных знаков: $f(0)<0$, $f(1)>0$. График является непрерывной линией, которую можно нарисовать, не отрывая карандаша от бумаги. Линия должна перейти из нижней полуплоскости $y<0$ в верхнюю $y>0$. При этом она не может «перепрыгнуть» через ось x, а должна ее обязательно пересечь в некоторой точке $x=c$. В этой точке функция $f(x)$ обращается в нуль, т. е. число c является корнем уравнения (13).

Мы рассуждали на интуитивном уровне, теперь сформулируем результат в виде теоремы.

Теорема о существовании корня непрерывной функции. *Если функция $f(x)$ непрерывна на отрезке $[a, b]$ и принимает на его концах значения*

разных знаков, то на этом отрезке существует по крайней мере один корень уравнения (13).

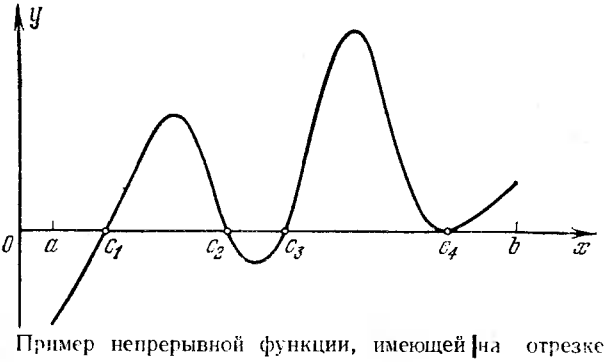

Рис. 8. Пример непрерывной функции, имеющей на отрезке $[a, b]$ четыре корня.

Обратите внимание на то, что, гарантируя существование решения уравнения, теорема не позволяет определить число его корней. На рис. 8 в качестве примера приведен

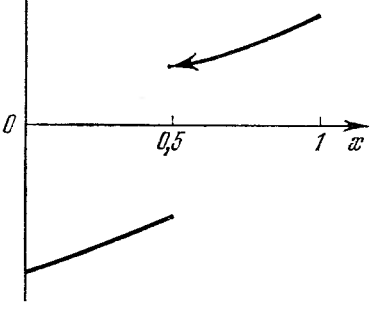

Рис. 9. Пример разрывной функции, принимающей на концах отрезка $[a, b]$ значения разных знаков, но не имеющей на этом отрезке корней.

график функции, удовлетворяющей условиям теоремы и имеющей на рассматриваемом отрезке четыре корня.

Требование непрерывности функции $f(x)$ во всех точках отрезка $[a, b]$ существенно. При наличии хотя бы одной точки разрыва утверждение теоремы становится неверным. Соответствующий пример показан на рис. 9. Мы видим на нем график разрывной функции, которая принимает на концах отрезка $[a, b]$ значения разных знаков, но не имеет корней.

Сформулированная теорема относится к числу так называемых *теорем существования*. Таких теорем, устанавливающ-

щих условия разрешимости различных математических задач, очень много. С некоторыми из них мы познакомимся в этой книге.

Доказательства теорем существования можно разделить на два типа. Бывают конструктивные доказательства, основу которых составляет метод фактического построения искомого решения. Они играют особенно важную роль в прикладной математике, в которой всегда требуется получить решение рассматриваемой задачи.

Наряду с этим весьма часто встречаются неконструктивные доказательства теорем существования. Как правило, они основаны на рассуждениях от противного: цепь логических заключений показывает, что решение обязано существовать, ибо в противном случае получится противоречие. В качестве характерного примера теоремы с неконструктивным доказательством можно привести основную теорему высшей алгебры. Она утверждает, что всякое алгебраическое уравнение (11) имеет по крайней мере один корень (вообще говоря, комплексный).

В следующем параграфе мы приведем конструктивное доказательство теоремы существования корня у непрерывной функции, которое нам даст один из самых простых и эффективных алгоритмов численного решения уравнений. Сейчас же рассмотрим несколько примеров, показывающих, как эта теорема применяется при исследовании уравнений.

В качестве первого примера обратимся еще раз к уравнению (12), которое предварительно перепишем в виде (13):

$$f(x) = x - \cos x = 0.$$

Функция $f(x) = x - \cos x$ непрерывна на отрезке $[0, 1]$, а ее значения на концах отрезка имеют разные знаки:

$$f(0) = -1 < 0, \quad f(1) = 1 - \cos 1 > 0.$$

Отсюда сразу следует существование на отрезке $[0, 1]$ по крайней мере одного корня уравнения (12). Раньше мы пришли к этому выводу с помощью наглядных, но математически нестрогих геометрических соображений. Теперь этот вывод — прямое следствие сформулированной теоремы.

Мы уже говорили, что теорема не позволяет определить общего числа корней. Однако в данном случае это легко сделать с помощью дополнительного исследования. Вычислим производную функции $f(x)$:

$$f'(x) = 1 + \sin x.$$

В интересующей нас области изменения переменной x: $x \in [0, 1]$ она положительна. Следовательно, функция $f(x)$ на отрезке $[0, 1]$ монотонно возрастает и может иметь только один корень.

В качестве второго примера рассмотрим алгебраическое уравнение (11) произвольной нечетной степени n. Обозначим многочлен, стоящий в левой части уравнения, через $P_n(x)$ и отметим, что функция $P_n(x)$ непрерывна на всей числовой прямой. Знак многочлена при достаточно больших по модулю значениях x совпадает со знаком его старшего члена a_0x^n. В силу нечетности n он различен для отрицательных и положительных x. Это позволяет утверждать, что всякое алгебраическое уравнение нечетной степени имеет по крайней мере один действительный корень.

На уравнения четной степени вывод не распространяется, в чем легко убедиться на примере простейшего уравнения

$$x^2 + 1 = 0.$$

Однако для них с помощью данной теоремы можно установить другой результат: если в алгебраическом уравнении произвольной четной степени n знаки коэффициентов a_0 и a_n противоположны, то это уравнение имеет по крайней мере один отрицательный и один положительный корень.

Предположим для определенности, что $a_0 > 0$, $a_n < 0$. Тогда при больших по модулю значениях x многочлен $P_n(x)$, как и его старший член a_0x^n, принимает положительное значение. В то же время при $x = 0$ он принимает отрицательное значение: $P_n(0) = a_n < 0$. Отсюда сразу следует требуемое утверждение.

Познакомившись на этих примерах с методом предварительного качественного исследования уравнений, перейдем теперь к обсуждению вычислительных алгоритмов для нахождения их корней. Первый из них, который мы рассмотрим в следующем параграфе, будет одновременно и методом доказательства теоремы о корне непрерывной функции.

§ 3. Метод вилки

В артиллерии существует следующий метод пристрелки: один снаряд посылают с недолетом, другой — с перелетом, и при этом говорят, что цель взята в «вилку». Послав следующий снаряд со средним значением прицела между двумя предыдущими, смотрят, как он упадет — с недолетом или

перелетом. В результате «вилка» сужается. Такая корректировка прицела продолжается до тех пор, пока снаряды не накроют цель.

Доказательство теоремы о корне непрерывной функции, которое мы приведем в этом параграфе, основано на идее «артиллерийской вилки». Строится последовательность вложенных друг в друга отрезков $[a_n, b_n]$. Их концы образуют монотонные последовательности, одна из которых $\{a_n\}$ («недолеты») сходится к некоторой точке $x=c$ снизу ($a_n \leqslant c$), вторая $\{b_n\}$ («перелеты») — сверху ($b_n \geqslant c$). Доказывается, что при выполнении условий теоремы предельная точка $x=c$ является корнем уравнения (13). В результате устанавливается существование решения этого уравнения на отрезке $[a, b]$. Процесс построения последовательности вложенных отрезков, содержащих искомый корень $x=c$, позволяет найти его приближенно с любой точностью ε (подобно вычислению π по периметрам правильных вписанных и описанных многоугольников p_n и q_n).

После этих вводных замечаний перейдем к доказательству теоремы. Предположим для определенности, что функция $f(x)$ принимает на левом конце отрезка $[a, b]$ отрицательное значение, на правом — положительное:

$$f(a) < 0, \qquad f(b) > 0.$$

Возьмем среднюю точку отрезка $[a, b]$ $\xi=(a+b)/2$ и вычислим в ней значение функции $f(x)$. Если $f(\xi)=0$, то утверждение теоремы доказано: мы нашли на отрезке $[a, b]$ точку $c=\xi$, в которой наша функция обращается в нуль. Если $f(\xi)\neq 0$, поступим следующим образом: рассмотрим два отрезка $[a, \xi]$ и $[\xi, b]$ и выберем один из них, исходя из условия, чтобы функция $f(x)$ принимала на его концах значения разных знаков. Выбранный отрезок обозначим $[a_1, b_1]$. По построению

$$f(a_1) < 0, \qquad f(b_1) > 0.$$

Затем возьмем среднюю точку отрезка $[a_1, b_1]$ $\xi_1 = (a_1+b_1)/2$ и вычислим в ней значение функции $f(\xi_1)$. Если $f(\xi_1)=0$, то доказательство теоремы закончено. Если же $f(\xi_1)\neq 0$, снова рассмотрим два отрезка $[a_1, \xi_1]$, $[\xi_1, b_1]$ и выберем тот, на концах которого функция $f(x)$ принимает значения разных знаков. Выбранный отрезок обозначим $[a_2, b_2]$. По построению

$$f(a_2) < 0, \qquad f(b_2) > 0.$$

Будем продолжать этот процесс. В результате он либо оборвется на некотором шаге n в силу того, что $f(\xi_n)=0$, либо будет продолжаться неограниченно. В первом случае вопрос о существовании корня уравнения (13) решен, поэтому нам нужно рассмотреть второй случай.

Неограниченное продолжение процесса дает последовательность отрезков $[a, b]$, $[a_1, b_1]$, $[a_2, b_2]$, . . . Эти отрезки вложены друг в друга — каждый последующий отрезок принадлежит всем предыдущим:

$$a_n \leqslant a_{n+1} < b_{n+1} \leqslant b_n, \tag{15}$$

причем

$$f(a_n) < 0, \qquad f(b_n) > 0.$$

Длины отрезков с возрастанием номера n стремятся к нулю:

$$\lim_{n\to\infty}(b_n - a_n) = \lim_{n\to\infty}\frac{b-a}{2^n} = 0.$$

Рассмотрим левые концы отрезков. Согласно (15) они образуют монотонно неубывающую ограниченную последовательность $\{a_n\}$. Такая последовательность имеет предел, который мы обозначим через c_1:

$$\lim_{n\to\infty} a_n = c_1. \tag{16}$$

Согласно (15) и теореме о переходе к пределу в неравенствах имеем

$$c_1 \leqslant b_n. \tag{17}$$

Теперь рассмотрим правые концы отрезков. Они образуют монотонно невозрастающую ограниченную последовательность $\{b_n\}$, которая тоже имеет предел. Обозначим этот предел через c_2:

$$\lim_{n\to\infty} b_n = c_2. \tag{18}$$

Согласно неравенству (17) пределы c_1 и c_2 удовлетворяют неравенству $c_1 \leqslant c_2$.

Итак,

$$a_n \leqslant c_1 \leqslant c_2 \leqslant b_n$$

и, следовательно,

$$c_2 - c_1 \leqslant b_n - a_n = \frac{b-a}{2^n}.$$

Таким образом, разность $c_2 - c_1$ меньше любого наперед

заданного положительного числа. Это означает, что $c_2 - c_1 = 0$, т. е.

$$c_1 = c_2 = c. \tag{19}$$

Найденная точка c интересна тем, что она является единственной общей точкой для всех отрезков построенной последовательности. Используя непрерывность функции $f(x)$, докажем, что она является корнем уравнения (13).

Мы знаем, что $f(a_n) < 0$. Согласно определению непрерывности и возможности предельного перехода в неравенствах имеем

$$f(c) = \lim_{n \to \infty} f(a_n) \leqslant 0. \tag{20}$$

Аналогично, учитывая, что $f(b_n) > 0$, получаем

$$f(c) = \lim_{n \to \infty} f(b_n) \geqslant 0. \tag{21}$$

Из (20) и (21) следует, что

$$f(c) = 0, \tag{22}$$

т. е. c — корень уравнения (13). Теорема доказана.

Процесс построения последовательности вложенных стягивающихся отрезков методом вилки является эффективным вычислительным алгоритмом решения уравнения (13). На n-м шаге процесса получаем

$$a_n \leqslant c \leqslant b_n. \tag{23}$$

Это двойное неравенство показывает, что число a_n определяет искомый корень c с недостатком, а число b_n — с избытком с ошибкой, не превышающей длину отрезка $\Delta_n = b_n - a_n = (b-a)/2^n$. При увеличении n ошибка стремится к нулю по закону геометрической прогрессии со знаменателем $q = 1/2$. Если задана требуемая точность $\varepsilon > 0$, то чтобы ее достигнуть, достаточно сделать число шагов N, удовлетворяющее условию

$$N > \log_2 \frac{b-a}{\varepsilon}. \tag{24}$$

В качестве примера применим метод вилки к решению уравнения (12), записанному в виде (13):

$$f(x) = x - \cos x = 0.$$

Результаты расчетов, связанных с 12-кратным делением исходного отрезка [0, 1] пополам, даны в табл. 2. Они определяют корень c с точностью $\varepsilon < (1/2)^{12} < 0{,}00025$.

Таблица 2

n	a_n	b_n	$\xi_n = \frac{a_n + b_n}{2}$	$f(\xi_n)$
0	0,000 000 000 000	1,000 000 000 000	0,500 000 000 000	—0,377 582
1	0,500 000 000 000	1,000 000 000 000	0,750 000 000 000	+0,018 311
2	0,500 000 000 000	0,750 000 000 000	0,625 000 000 000	—0,185 963
3	0,625 000 000 000	0,750 000 000 000	0,687 500 000 000	—0,085 335
4	0,687 500 000 000	0,750 000 000 000	0,718 750 000 000	—0,033 879
5	0,718 750 000 000	0,750 000 000 000	0,734 375 000 000	—0,007 875
6	0,734 375 000 000	0,750 000 000 000	0,742 187 500 000	+0,005 196
7	0,734 375 000 000	0,742 187 500 000	0,738 281 250 000	—0,001 345
8	0,738 281 250 000	0,742 187 500 000	0,740 234 375 000	+0,001 924
9	0,738 281 250 000	0,740 234 375 000	0,739 257 812 500	+0,000 289
10	0,738 281 250 000	0,739 257 812 500	0,738 769 531 250	—0,000 528
11	0,738 769 531 250	0,739 257 812 500	0,739 013 671 875	—0,000 120
12	0,739 013 671 875	0,739 257 812 500		

Итак, мы можем утверждать, что искомый корень c принадлежит отрезку [0,739 013 671 875, 0,739 257 812 500]. Отбрасывая десятичные знаки, лежащие за пределами достигнутой точности, получим

$$0{,}73901 < c < 0{,}73926. \tag{25}$$

§ 4. Метод итераций (метод последовательных приближений)

В этом параграфе мы познакомимся еще с одним численным методом решения уравнений. Предположим, что уравнение можно записать в виде

$$x = \varphi(x). \tag{26}$$

Возьмем произвольное значение x_0 из области определения функции $\varphi(x)$ и будем строить последовательность чисел $\{x_n\}$, определенных с помощью рекуррентной формулы

$$x_{n+1} = \varphi(x_n), \quad n = 0, 1, 2, \ldots \tag{27}$$

Последовательность $\{x_n\}$ называется *итерационной последовательностью*. При ее изучении встают два вопроса:

1) Можно ли процесс вычисления чисел x_n продолжать неограниченно, т. е. будут ли числа x_n принадлежать области определения функции $\varphi(x)$?

2) Если итерационный процесс (27) бесконечен, то как ведут себя числа x_n при $n \to \infty$?

Исследование этих вопросов показывает, что при определенных ограничениях на функцию $\varphi(x)$ итерационная последовательность является бесконечной и сходится к корню уравнения (26):

$$\lim_{n \to \infty} x_n = c, \qquad c = \varphi(c). \tag{28}$$

Однако для того, чтобы провести это исследование, нам нужно ввести одно новое понятие.

Говорят, что функция $f(x)$ удовлетворяет на отрезке $[a, b]$ *условию Липшица*, если существует такая постоянная α, что для любых x_1, x_2, принадлежащих отрезку $[a, b]$, имеет место неравенство

$$|f(x_1) - f(x_2)| \leqslant \alpha |x_1 - x_2|. \tag{29}$$

Величину α в этом случае называют *постоянной Липшица*.

Если функция $f(x)$, удовлетворяет на отрезке $[a, b]$ условию Липшица, то она непрерывна на нем. Действительно, пусть x_0 — произвольная точка отрезка. Рассмотрим приращение функции $f(x)$ в этой точке:

$$\Delta f = f(x_0 + \Delta x) - f(x_0)$$

и оценим его с помощью неравенства (29):

$$|\Delta f| \leqslant \alpha |\Delta x|.$$

Таким образом, $\lim_{\Delta x \to 0} \Delta f = 0$, что означает непрерывность функции $f(x)$.

Условие Липшица имеет простой геометрический смысл. Возьмем на графике функции $y = f(x)$ две произвольные точки: M_1 с координатами $(x_1, f(x_1))$ и M_2 с координатами $(x_2, f(x_2))$ (рис. 10). Напишем уравнение прямой линии, проходящей через эти точки:

$$y = f(x_1) + k(x - x_1),$$

где k — тангенс угла наклона прямой к оси x — определяется формулой

$$k = \frac{f(x_2) - f(x_1)}{x_2 - x_1}.$$

Если функция $f(x)$ удовлетворяет на отрезке $[a, b]$ условию Липшица (29), то при произвольном выборе точек M_1 и M_2 имеем $|k| \leqslant \alpha$. Таким образом, с геомет ниче-

ской точки зрения условие Липшица означает ограниченность тангенса угла наклона секущих, проведенных через всевозможные пары точек графика функции $y=f(x)$.

Рис. 10. Геометрическая иллюстрация условия Липшица.

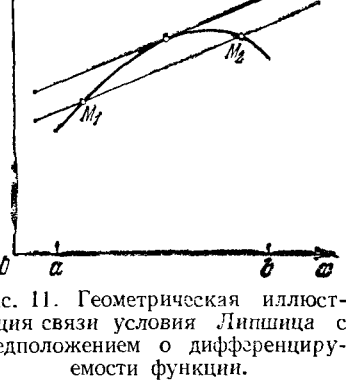

Рис. 11. Геометрическая иллюстрация связи условия Липшица с предположением о дифференцируемости функции.

Предположим, что функция $f(x)$ имеет на отрезке $[a, b]$ ограниченную производную. $|f'(x)|\leqslant m$; тогда она удовлетворяет условию Липшица с постоянной $\alpha=m$. Для доказательства этого утверждения воспользуемся *формулой конечных приращений Лагранжа*:

$$f(x_2)-f(x_1)=f'(\xi)(x_2-x_1), \qquad (30)$$

где x_1, x_2 — произвольные точки отрезка $[a, b]$, ξ — некоторая точка отрезка $[x_1, x_2]$. Возьмем модуль обеих частей равенства (30) и заменим в правой части $|f'(\xi)|$ на m. В результате получим неравенство (29) с $\alpha=m$.

Рис. 11 дает геометрическую иллюстрацию установленного свойства. Согласно формуле Лагранжа (30) каждой секущей графика функции $y=f(x)$ можно поставить в соответствие параллельную ей касательную. Поэтому наибольший тангенс угла наклона секущих не превосходит наибольшего тангенса угла наклона касательных, и его можно оценить той же константой m : $|k|\leqslant m$.

Познакомившись с условием Липшица, перейдем к изучению итерационной последовательности, предполагая, что уравнение (26) имеет корень $x=c$. Существование этого корня можно установить с помощью предварительного качественного исследования уравнения с применением теоремы § 2.

Теорема о сходимости итерационной последовательности. *Пусть c — корень уравнения* (26) *и пусть функция* $\varphi(x)$ *удовлетворяет на некотором отрезке* $[c-\delta, c+\delta]$ $(\delta>0)$ *условию Липшица с постоянной* $\alpha<1$. *Тогда при любом выборе* x_0 *на отрезке* $[c-\delta, c+\delta]$ *существует бесконечная итерационная последовательность* $\{x_n\}$ (27) *и эта последовательность сходится к корню* $x=c$, *который является единственным решением уравнения* (26) *на отрезке* $[c-\delta, c+\delta]$.

Сформулированная теорема имеет очень простой смысл. Будем говорить, что функция φ осуществляет отображение точки x на точку $y=\varphi(x)$. Тогда условие Липшица с постоянной $\alpha<1$ означает, что отображение φ является сжимающим: расстояние между точками x_1 и x_2 больше, чем расстояние между их изображениями $y_1=\varphi(x_1)$ и $y_2=\varphi(x_2)$.

Корень c является неподвижной точкой отображения φ, он преобразуется сам в себя: $c=\varphi(c)$. Поэтому каждый шаг в итерационном процессе (27), сжимая расстояния, должен приближать члены последовательности $\{x_n\}$ к неподвижной точке c.

После этих соображений, поясняющих смысл теоремы, перейдем к ее доказательству. Возьмем произвольную точку x_0 на отрезке $[c-\delta, c+\delta]$, она отстоит от точки c не больше, чем на δ: $|c-x_0|\leqslant\delta$.

Вычислим x_1: $x_1=\varphi(x_0)$, при этом $x_1-c=\varphi(x_0)-\varphi(c)$. Разность $\varphi(x_0)-\varphi(c)$ можно оценить с помощью условия Липшица:

$$|x_1-c|=|\varphi(x_0)-\varphi(c)|\leqslant\alpha|x_0-c|\leqslant\alpha\delta. \qquad (31)$$

Неравенство (31) показывает, что x_1 принадлежит отрезку $[c-\delta, c+\delta]$ и расположена ближе к точке c, чем x_0.

Продолжим построение итерационной последовательности. Вычислим x_2 : $x_2=\varphi(x_1)$, при этом

$$|x_2-c|=|\varphi(x_2)-\varphi(c)|\leqslant\alpha|x_1-c|\leqslant\alpha^2|x_0-c|\leqslant\alpha^2\delta.$$

Точка x_2 опять принадлежит отрезку $[c-\delta, c+\delta]$ и расположена ближе к точке c, чем точка x_1, т. е. мы приблизились к c.

По индукции легко доказать, что последующие итерации также существуют и удовлетворяют неравенствам

$$|x_n-c|\leqslant\alpha^n|x_0-c|\leqslant\alpha^n\delta. \qquad (32)$$

Отсюда следует, что

$$\lim_{n\to\infty}(x_n-c)=0, \quad \text{т. е.} \quad \lim_{n\to\infty}x_n=c. \qquad (33)$$

Нам остается доказать, что корень $x=c$ является единственным решением уравнения (26) на отрезке $[c-\delta,\ c+\delta]$. Действительно, допустим, что существует еще один корень $x=c_1$:

$$c_1=\varphi(c_1), \qquad c_1\in[c-\delta,\ c+\delta]. \tag{34}$$

Примем c_1 за нулевое приближение и будем строить итерационную последовательность (27). Тогда с учетом (34) получим $x_n=c_1$ $(n=0,\ 1,\ 2,\ \ldots)$. С другой стороны, по доказанному $\lim\limits_{n\to\infty} x_n=c$, т. е. $c_1=c$. Никаких других решений уравнение (26) на отрезке $[c-\delta,\ c+\delta]$ иметь не может.

Сходимость итерационной последовательности к корню уравнения (26) может быть использована для приближенного определения этого корня с любой степенью точности. Для этого нужно только провести достаточное число итераций.

В качестве примера, иллюстрирующего данный метод, рассмотрим еще раз уравнение $x=\cos x$. Роль функции $\varphi(x)$ в нем играет $\cos x$. Это — дифференцируемая функция, производная которой равна $-\sin x$. На отрезке $[0,\ 1]$

$$|\varphi'(x)|=\sin x\leqslant\sin 1. \tag{35}$$

Таким образом, функция $\varphi(x)=\cos x$ удовлетворяет на отрезке $[0,\ 1]$ условию Липшица с постоянной $\alpha=\sin 1<1$.

Таблица 3

n	x_{2n}	x_{2n+1}
0	0,500 000 000 000	0,877 582 561 890
1	0,639 012 494 166	0,802 685 100 681
2	0,694 778 026 789	0,768 195 831 281
3	0,719 165 445 942	0,752 355 759 420
4	0,730 081 063 138	0,745 120 341 349
5	0,735 006 309 016	0,741 826 522 642
6	0,737 235 725 443	0,740 329 651 877
7	0,738 246 238 333	0,739 649 062 768
8	0,738 704 539 357	0,739 341 452 279
9	0,738 912 449 332	0,739 201 444 135

Результаты вычислений по рекуррентной формуле (27), которая в случае уравнения (12) принимает вид $x_{n+1}=\cos x_n$, даны в табл. 3. За нулевое приближение была

выбрана средняя точка отрезка $x_0=0{,}5$. Для удобства анализа итерационной последовательности ее члены расположены по два в строке. В результате образовались столбцы членов с четными и нечетными номерами. Сравнивая их между собой, мы видим, что четные члены меньше нечетных: итерационная последовательность «скачет» то вверх, то вниз. С возрастанием номера четные члены возрастают, а нечетные — убывают, приближаясь друг к другу. Такое поведение последовательности означает, что корень уравнения (12) лежит между четными и нечетными итерациями; первые дают его значение с недостатком, вторые — с избытком. Это позволяет легко контролировать точность, достигнутую после любого числа итераций: погрешность не превышает разности между последними вычисленными нечетным и четным членами.

Например, мы остановили процесс вычислений на 19-й итерации и можем написать для корня c двойное неравенство:

$$0{,}738\,912\,449\,332 = x_{18} < c < x_{19} = 0{,}739\,201\,444\,135, \quad (36)$$

т. е. члены итерационной последовательности x_{18} и x_{19} определяют c как с недостатком, так и с избытком, с погрешностью, которая не превышает разности $x_{19}-x_{18}$.

$$\varepsilon < \Delta_{19} = x_{19} - x_{18} < 0{,}0003.$$

Точность, которую мы достигли после 19 итераций, примерно соответствует точности 12 шагов в методе вилки. Причина такого различия ясна. В обоих методах погрешность убывает по закону геометрической прогрессии. Для метода вилки знаменатель прогрессии равен 1/2, он не зависит от вида функции $f(x)$. Для метода итераций знаменатель равен α — постоянной Липшица функции $\varphi(x)$. В рассматриваемом примере $\alpha>1/2$, поэтому сходимость метода итераций медленнее сходимости метода вилки. Это означает, что метод итераций имеет преимущество перед методом вилки с точки зрения скорости сходимости только в том случае, когда $\alpha<1/2$.

§ 5. Метод касательных (метод Ньютона)

Метод касательных, связанный с именем И. Ньютона, является одним из наиболее эффективных численных методов решения уравнений. Идея метода очень проста. Предположим, что функция $f(x)$, имеющая корень c на отрезке a, b, дифференцируема на этом отрезке и ее производная

$f'(x)$ не обращается на нем в нуль. Возьмем произвольную точку x_0 и запишем в ней уравнение касательной к графику функции $f(x)$:

$$y = f(x_0) + f'(x_0)(x - x_0). \tag{37}$$

Графики функции $f(x)$ и ее касательной близки около точки касания, поэтому естественно ожидать, что точка x_1 пересечения касательной с осью x будет расположена недалеко от корня c (рис. 12).

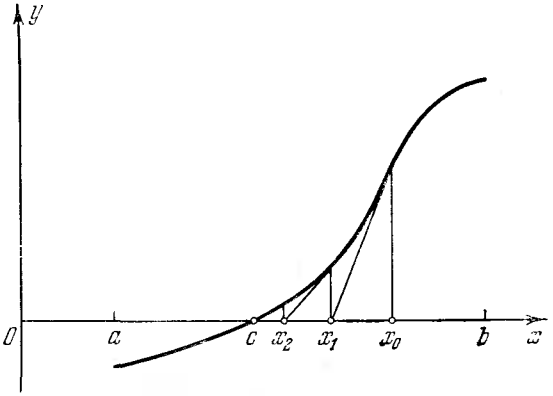

Рис. 12. Построение последовательности $\{x_n\}$ по методу касательных.

Для определения точки x_1 имеем уравнение

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0.$$

Таким образом,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{38}$$

Повторим проделанную процедуру: напишем уравнение касательной к графику функции $f(x)$ при $x = x_1$ и найдем для нее точку пересечения x_2 с осью x (см. рис. 12):

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}.$$

Продолжая этот процесс, получим последовательность $\{x_n\}$, определенную с помощью рекуррентной формулы

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, 2, \ldots \tag{39}$$

При исследовании этой последовательности, как и последовательности (27) метода итераций, встают два вопроса:

1) Можно ли процесс вычисления чисел x_n продолжать неограниченно, т. е. будут ли числа x_n принадлежать отрезку $[a, b]$?

2) Если процесс (39) бесконечен, то как ведет себя последовательность $\{x_n\}$ при $n\to\infty$?

При анализе этих вопросов предположим, что корень $x=c$ является внутренней точкой отрезка $[a, b]$ $(a<c<b)$, а функция $f(x)$ дважды дифференцируема на данном отрезке, причем ее производные удовлетворяют неравенствам

$$|f'(x)|\geqslant m>0, \quad |f''(x)|\leqslant M, \quad x\in[a, b], \tag{40}$$

и докажем следующую теорему.

Теорема о сходимости метода касательных. *Если функция $f(x)$ удовлетворяет сформулированным условиям, то найдется такое* δ: $0<\delta\leqslant\min(c-a, b-c)$, *что при любом выборе начального приближения на отрезке* $[c-\delta, c+\delta]\subset[a, b]$ *существует бесконечная итерационная последовательность* (39) *и эта последовательность сходится к корню* c.

Доказательство. В силу предположения о дифференцируемости функции $f(x)$ и не равенстве нулю ее производной $f'(x)$ уравнение (13) эквивалентно на отрезке $[a, b]$ уравнению

$$x=\varphi(x), \qquad \varphi(x)=x-\frac{f(x)}{f'(x)}, \tag{41}$$

так что корень $x=c$ исходного уравнения является одновременно корнем уравнения (41). Исследуем возможность отыскания этого корня с помощью метода итераций.

Вычислим производную функции $\varphi(x)$:

$$\varphi'(x)=1-\frac{(f'(x))^2-f(x)f''(x)}{(f'(x))^2}=\frac{f(x)f''(x)}{(f'(x))^2} \tag{42}$$

и оценим полученное выражение. Согласно неравенствам (40)

$$|\varphi'(x)|\leqslant\frac{M}{m^2}|f(x)|. \tag{43}$$

Для дальнейшей оценки $|\varphi'(x)|$ воспользуемся непрерывностью функции $f(x)$ и равенством ее нулю в точке $x=c$:

$$\lim_{x\to c} f(x)=f(c)=0. \tag{44}$$

Положим $\varepsilon = m^2/(2M)$; тогда в силу (44) для данного ε можно указать такое δ: $0 < \delta \leqslant \min(c-a, b-c)$, что для всех $x \in [c-\delta, c+\delta]$ выполняется неравенство

$$|f(x)-f(c)| = |f(x)| \leqslant \varepsilon = \frac{m^2}{2M}. \tag{45}$$

Учитывая это, получим

$$|\varphi'(x)| \leqslant \frac{M}{m^2} \cdot \frac{m^2}{2M} = \frac{1}{2}. \tag{46}$$

Таким образом, функция $\varphi(x)$ удовлетворяет на отрезке $[c-\delta, c+\delta] \subset [a, b]$ условию Липшица с постоянной $\alpha = 1/2 < 1$. Это означает, что уравнение (41) можно решать методом итераций: при любом выборе нулевого приближения x_0 на отрезке $[c-\delta, c+\delta]$ существует бесконечная последовательность $\{x_n\}$ (27), сходящаяся к корню $x=c$.

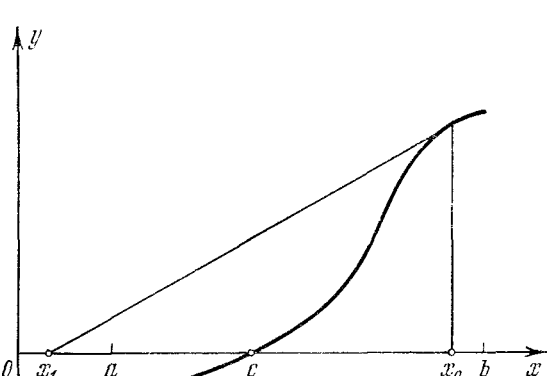

Рис. 13. Случай, когда процесс построения последовательности $\{x_n\}$ обрывается из-за плохого выбора нулевого приближения.

Теперь нам остается заметить, что итерационной последовательностью для уравнения (41), сходимость которой мы только что установили, является последовательность (39) метода касательных. Теорема доказана.

Требование близости нулевого приближения x_0 к искомому корню c является существенным для метода касательных. Так, на рис. 13 изображен график той же функции $f(x)$, что и на рис. 12, однако x_0 выбрано дальше от корня c, чем в первом случае. В результате после первого же шага получается точка x_1, которая не принадлежит исходному

отрезку $[a, b]$, и на этом процесс построения рекуррентной последовательности метода касательных обрывается.

Таким образом, до начала расчетов по данному методу для выбора нулевого приближения x_0 нужно знать область локализации искомого корня $x=c$. Если известен в общих чертах график функции $f(x)$, то ее легко определить по этому графику. В случае необходимости можно сделать несколько шагов по методу вилки. Затруднения, связанные с предварительным исследованием уравнения, вполне окупаются высокой скоростью сходимости метода.

В качестве первого примера применения метода касательных рассмотрим задачу извлечения квадратного корня из произвольного положительного числа a. Будем искать его как решение уравнения

$$f(x)=x^2-a=0. \tag{47}$$

Рекуррентная формула метода Ньютона (39) в данном случае принимает вид:

$$x_{n+1}=x_n-\frac{x_n^2-a}{2x_n}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right). \tag{48}$$

Она совпадает с формулой (3). Таким образом, алгоритм вычисления $\sqrt{a}$ основан на решении уравнения (47) методом Ньютона. Мы уже знаем, что процесс сходится к $\sqrt{a}$ при любом выборе начального приближения $x_0 \in (0, +\infty)$, причем сходимость весьма быстрая.

Второй пример снова будет связан с уравнением $x=\cos x$ (мы уже пользовались этим уравнением для иллюстрации двух предыдущих методов). Запишем для него рекуррентную формулу (39) (помня, что $f(x)=x-\cos x$):

$$x_{n+1}=x_n-\frac{x_n-\cos x_n}{1+\sin x_n}, \qquad n=0,\ 1,\ 2,\ \ldots \tag{49}$$

Выберем, как и для метода итераций, в качестве нулевого приближения $x_0=0{,}5$ и вычислим несколько следующих приближений по формуле (49):

$$x_1=0{,}755\,222\,320\,557, \qquad x_4=0{,}739\,085\,078\,239,$$
$$x_2=0{,}739\,141\,702\,652, \qquad x_5=0{,}739\,085\,078\,239.$$
$$x_3=0{,}739\,085\,197\,449,$$

Мы видим, что, начиная с номера $n=1$, последовательность $\{x_n\}$ убывает и приближается к корню $x=c$ сверху. После четвертого шага процесс «останавливается». Мы уже встречались с таким явлением, когда обсуждали алгоритм

вычисления квадратного корня. Остановка связана с тем, что расчеты ведутся с 12 знаками, и после достижения погрешности, не превышающей 10^{-12}, становится невозможно уловить разницу между x_{n+1} и x_n, лежащую за пределами ошибки округления. Если есть необходимость повысить точность, то нужно перейти к расчетам с бо́льшим числом знаков. ЭВМ допускают такую возможность.

Примеры вычисления корня (47) и решения уравнения (12) показывают очень высокую скорость сходимости метода касательных. Для уравнения (12) после двух шагов была достигнута точность 10^{-4}, после четырех — 10^{-12}. Для сравнения укажем, что точность 10^{-4} метод вилки обеспечивает на 15-м шаге, метод итераций — на 22-м шаге.

§ 6. Заключительные замечания

Мы познакомились с тремя методами численного решения уравнений. Наряду с ними существуют еще несколько методов, на которых мы не останавливались. Ситуация, когда одну и ту же математическую задачу можно решать с помощью разных методов, является довольно типичной. В таких случаях естественно возникает необходимость сравнения их между собой.

При оценке эффективности численных методов существенное значение имеют различные свойства:

1) универсальность;

2) простота организации вычислительного процесса и контроля за точностью;

3) скорость сходимости.

Посмотрим с этой точки зрения на разработанные методы решения уравнений.

1) Наиболее универсальным является метод вилки: он требует только непрерывности функции $f(x)$. Два других метода накладывают более сильные ограничения. Во многих случаях это преимущество метода вилки может оказаться существенным.

2) С точки зрения организации вычислительного процесса все три метода очень просты. Однако и здесь метод вилки обладает определенным преимуществом. Вычисления можно начинать с любого отрезка $[a, b]$, на концах которого непрерывная функция $f(x)$ принимает значения разных знаков. Процесс будет сходиться к корню уравнения $f(x)=0$, причем на каждом шаге он дает для корня двустороннюю оценку, по которой легко определить достигнутую точность.

Сходимость же метода итераций или касательных зависит от того, насколько удачно выбрано нулевое приближение.

3) Наибольшей скоростью сходимости обладает метод касательных. В случае, когда подсчет значений функции $f(x)$ сложен и требует больших затрат машинного времени, это преимущество становится определяющим.

Итак, мы видим, что ответ на вопрос о наилучшем численном методе решения уравнений не однозначен. Он существенно зависит от того, какую дополнительную информацию о функции $f(x)$ мы имеем и, в соответствии с этим, каким свойствам метода придаем наибольшее значение.

При обосновании метода итераций и метода Ньютона на функции $\varphi(x)$ и $f(x)$, а также на выбор начального приближения x_0 накладывались определенные ограничения. Однако при решении конкретных задач проверить их выполнение часто бывает трудно и даже практически невозможно. Функция может не задаваться в виде простой формулы, а находиться в результате численного решения некоторой математической задачи, получаться из измерений и т. д. В таких случаях применимость метода приходится проверять «экспериментально»: начинают расчет и следят за поведением первых членов последовательности $\{x_n\}$. Если по ним видно, что процесс сходится, то расчет продолжают, пока не достигнут нужной точности. В противном случае вычисления прекращают и анализируют полученные данные, пытаясь установить причину расходимости и, в соответствии с ней, выбрать другой метод решения задачи.

Такая организация работы не связана с конкретной темой нашего разговора — численными методами решения уравнений. Она носит общий характер. В прикладной математике существует много численных методов, особенно относящихся к сложным задачам, которые пока не получили строгого обоснования, но успешно применяются на практике. Именно этот факт — основной аргумент в их пользу.

В заключение отметим, что центральная идея метода итераций — сжимающие отображения — является весьма общей. При незначительной модификации она может быть использована для изучения гораздо более сложных математических задач, чем уравнение (13). Для многих типов нелинейных уравнений принцип сжимающих отображений является, по существу, единственным методом их исследования и решения. Существенные обобщения допускает также метод Ньютона. Оба метода в своей общей форме играют важную роль в современной математике.

ГЛАВА 3

ЭЛЕКТРОННО-ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

§ 1. От 10 пальцев к ЭВМ

Применение математических методов для решения практических задач неизбежно связано с проведением расчетов, с доведением ответа «до числа». Поэтому проблема упрощения и ускорения вычислений всегда имела первостепенное значение. Один из путей ее решения был связан с усовершенствованием методов счета. Самым важным результатом здесь явился переход от аддитивной *) (например римской) системы записи чисел к принятой в настоящее время позиционной системе.

В римском счислении MCXXXVI означает пятьсот + +сто + десять + десять + десять + пять + один. В ней с увеличением изображаемых чисел нужно неограниченно увеличивать число используемых символов. Однако главный недостаток аддитивной системы заключается в сложности выполнения арифметических операций при такой форме записи чисел. Это «искусство» требовало специальной профессиональной подготовки, которую в то время могли получить лишь немногие.

Позиционная система обладает по сравнению с аддитивной двумя существенными преимуществами.

Во-первых, в ней любое число записывается с помощью небольшого числа символов (в общепринятой десятичной системе ими являются десять арабских цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; в двоичной системе, которая используется в большинстве вычислительных машин, такими символами являются 0 и 1). В десятичной системе число MCXXXVI представляется в виде

$$636 = 6 \cdot 10^2 + 3 \cdot 10 + 6,$$

*) Аддитивный (от латинского additivus — придаточный) — полученный путем сложения.

при этом один и тот же символ — цифра 6 — в зависимости от своего положения имеет разное значение: в одном случае он означает число сотен, в другом — число единиц.

Во-вторых, для позиционной системы правила выполнения арифметических операций намного проще, чем для аддитивной. Они сводятся к запоминанию таблиц сложения и умножения однозначных чисел.

Важное значение для упрощения вычислений сыграло также изобретение логарифмов. Вот что писал по этому поводу французский математик и астроном П. Лаплас, которому приходилось проводить большие расчеты в связи с астрономическими исследованиями: «Изобретение логарифмов, сокращая вычисления нескольких месяцев в труд нескольких дней, словно удваивает жизнь астронома».

Наряду с упрощением методов счета постоянно велись поиски путей ускорения процесса вычислений. Самым первым «техническим средством» в этой области стали 10 пальцев на руках человека. Они породили способ счета по пальцам и десятичную систему счисления. Многовековую историю имеют обычные канцелярские счеты, которые до сравнительно недавнего времени были обязательным атрибутом на столе у любого бухгалтера и кассира. Несколько поколений инженеров и научных работников широко пользовалось логарифмической линейкой для проведения простых, не требующих высокой точности расчетов.

В XVII в. начали появляться механические вычислительные машины, главная особенность которых состояла в том, что алгоритмы выполнения арифметических операций закладывались в само устройство. Их конструкция непрерывно совершенствовалась. В первой половине XX в. для проведения вычислений с многоразрядными числами широко использовались созданные по этому принципу ручные арифмометры и настольные клавишные машины, которые приводились в действие электромотором. При работе на таких машинах вычислителю было нужно набирать исходные данные на клавиатуре, записывать полученные результаты в специально приготовленную таблицу и следить за порядком выполнения арифметических операций, который определялся алгоритмом решения задачи. Сами же операции выполнялись машиной. Машина в несколько раз увеличивала скорость счета и существенно снижала нервное напряжение работника по сравнению со счетом вручную. Грубо это можно сравнить с ездой на лошади: и быстрее, чем пешком, и не так утомительно.

Однако развитие науки и техники в первой половине XX в., особенно в 30-е—40-е годы, существенно расширило круг прикладных задач, поставленных перед математикой. Их сложность не позволяла, как правило, получить ответ в явном виде и требовала применения численных методов. Для проведения расчетов стали создаваться специальные группы вычислителей. Объем вычислений и потребность в вычислителях резко нарастали. Стало ясно, что идти дальше по экстенсивному *) пути нельзя. Необходим был принципиально новый шаг, и этот шаг был сделан: появились ЭВМ.

Создание ЭВМ можно сравнить с самыми выдающимися достижениями — такими, как изобретение колеса, освоение выплавки металлов, создание паровой машины, освоение электричества, использование атомной энергии. Хорошо известна роль так называемых великих открытий в судьбе человечества: за сравнительно короткий срок они существенно изменяли производительные силы общества, оказывая большое влияние на условия его жизни.

Однако в этом ряду ЭВМ занимает особое место: если обычные машины расширяли физические возможности людей, то ЭВМ существенно повысили их интеллектуальный потенциал. Они способствовали развитию новых эффективных методов познания и использования законов природы, явились одним из наиболее важных факторов научно-технического прогресса последних десятилетий.

Далее мы коротко расскажем о принципах работы и основных этапах развития ЭВМ.

§ 2. Как работают ЭВМ

Для того чтобы ответить на этот вопрос, обсудим сначала, как работали вычислители, вооруженные клавишными машинами типа арифмометров, до появления ЭВМ.

Пусть нужно вычислить $\sqrt{a}=\sqrt{81\,725{,}3}$ (см. § 1 гл. 2). Вычислитель решает воспользоваться методом, основанным на рекуррентной формуле (2.3) **), выбрав за нулевое приближение $x_0=300$. После этого он заготовляет табл. 1.

Выполняя расчеты, вычислитель заполняет в таблице

*) Экстенсивный (от латинского extensivus — расширяющий, удлиняющий) — в противоположность интенсивному означает не качественное, а лишь количественное изменение.

**) Ссылка на формулу вида (2.3) означает, что ее надо искать в гл. 2 под номером (3).

строчку за строчкой, пока не получит результата требуемой точности (об этом можно судить либо по неравенству (2.9), либо просто по установлению нужного числа десятичных знаков у последовательности $\{x_n\}$).

Таблица 1

n	$\frac{a}{x_{n-1}}$	$x_{n-1}+\frac{a}{x_{n-1}}$	$x_n=x_{n-1}+\frac{a}{x_{n-1}}$
0			300
1			
2			
3			
4			

Приведем табл. 1 в заполненном виде после завершения вычислений (табл. 2). (В отличие от приведенных на с. 33, здесь результаты содержат не 12, а только 8 значащих цифр в соответствии с разрядностью клавишных машин.)

Таблица 2

n	$\frac{a}{x_{n-1}}$	$x_{n-1}+\frac{a}{x_{n-1}}$	$x_n=x_{n-1}+\frac{a}{x_{n-1}}$
0			300
1	272,41766	572,41766	286,20883
2	285,54429	571,75312	285,87656
3	285,87618	571,75274	285,87637
4	285,87637	571,75274	285,87637

Клавишные машины — это механические устройства. Считали они медленно: на выполнение одного действия уходило несколько секунд. Однако самым медленным звеном в вычислительном процессе была не машина, а человек. Основное время уходило не на счет, а на набор с помощью клавиш данных для очередной операции и на запись результатов в таблицу.

Принципиально новый шаг, который был сделан при создании ЭВМ, состоял в полной автоматизации вычислений, в их проведении без участия человека. Для этого нужно заранее описать в «понятной» для машины форме

всю последовательность операций, необходимую для решения задачи (такое описание называется *программой*), и задать исходные данные.

Чтобы программу и исходные данные можно было ввести в машину, выполнить необходимые вычисления и вывести полученные результаты, любая ЭВМ, независимо от ее конкретных конструктивных особенностей, должна иметь следующие узлы.

1) *Устройство управления* (УУ): управляет порядком выполнения операций, координирует работу всех узлов машины.

2) *Арифметико-логическое устройство* (АЛУ): служит для выполнения арифметических и логических операций.

3) *Запоминающее устройство* (ЗУ) или просто *память*: предназначено для хранения программы, исходных данных и результатов вычислений.

4) *Устройство ввода-вывода* (УВВ): используется для ввода в машину программы, исходных данных и вывода результатов счета, т. е. для обмена информацией между человеком и машиной.

Память ЭВМ разбита на отдельные перенумерованные ячейки. В каждую ячейку можно записать одно число. Номер ячейки называется ее *адресом*, хранимое в ней число — *машинным словом*. Одна часть машинных слов, находящихся во время работы ЭВМ в ее памяти, представляет собой «настоящие» числа, другая — команды программы. В условной, закодированной форме числа-команды определяют, из каких ячеек нужно взять данные, какую операцию над ними выполнить, в какую ячейку записать результат. С одним из вариантов записи команд мы познакомимся в конце параграфа.

Ячейки памяти состоят из элементов, в них помещаются отдельные разряды хранимого числа. Количество разрядов в ячейке задается конструктивными особенностями машины и одинаково для всех ячеек. Оно определяет длину машинного слова данной ЭВМ. Обычно в ЭВМ используется двоичная система счисления. В соответствии с этим элементы ячеек памяти могут находиться в одном из двух состояний (намагничено — не намагничено, есть ток — нет тока и т. д.). Одно из них интерпретируется как цифра 0, второе — как цифра 1. Проанализировав состояние всех элементов ячейки, можно прочитать помещенное в нее машинное слово. Меняя состояние элементов, мы заменим одно машинное слово другим.

Различают *оперативную (внутреннюю)* и *внешнюю память*. Разница между ними состоит в том, что УУ и АЛУ могут непосредственно получать и обрабатывать информацию, находящуюся только в оперативной памяти. Для работы с информацией, хранящейся во внешней памяти, ее нужно сначала поместить в оперативную память. Это осуществляется с помощью специальных команд.

Внешняя память современных ЭВМ состоит из магнитных лент, дисков, барабанов. Объем этих носителей в тысячи, десятки тысяч раз больше объема оперативной памяти. Однако для того, чтобы получить доступ к нужному элементу данных, расположенному во внешней памяти, его предварительно надо переслать в оперативную память, а эта операция занимает намного больше времени, чем прямое обращение в оперативную память.

Ввод программы и исходных данных в ЭВМ первоначально осуществлялся с помощью перфокарт. В настоящее время для этого используются также другие методы, о которых мы коротко расскажем в дальнейшем.

Перфокарты — это специальные картонные карты стандартных размеров и формы. Нужная информация наносится (перфорируется *)) на них в виде набора отверстий. Перфорация осуществляется заранее с помощью специального устройства — перфоратора, не связанного непосредственно с ЭВМ. Подготовленные перфокарты с отперфорированной на них программой и исходными данными ставятся в читающее устройство ЭВМ, которое просматривает их одну за другой, прочитывает закодированную на них информацию и записывает ее в оперативную память. После того как программа и исходные данные введены, машина может приступать к вычислениям. Результаты расчетов, предписанные программой, печатаются на бумаге специальным печатающим устройством.

Отметим, что числа вводятся в ЭВМ и выводятся из нее на печать в привычной десятичной форме. Машина сама переводит их при вводе из десятичной системы в двоичную, а при выдаче на печать — из двоичной в десятичную. Пользователь ЭВМ может вообще ничего не знать ни о двоичной системе счисления, ни о том, что она используется для проведения вычислений в ЭВМ. Это никак не отразится на его работе.

---

*) Перфорировать (от латинского perforare — буравить) — пробивать отверстия, просверливать.

Устройство управления и арифметико-логическое устройство образуют единый комплекс, который называется *центральным процессором*. Центральный процессор и оперативная память составляют ядро ЭВМ. Все остальные элементы машины относятся к разряду *внешних устройств* или *периферийного оборудования*.

Теперь, когда мы познакомились с основными элементами ЭВМ и их назначением, приведем в качестве примера программу вычисления квадратного корня из положительного числа a с заданной точностью ε. При этом, как и в § 1 гл. 2, будем обозначать через x_0 нулевое приближение (за него можно принять, как мы знаем, любое положительное число) и через c_0— какое-нибудь число, удовлетворяющее условию $0<c_0\leqslant\sqrt{a}$. Оно нужно для оценки достигнутой точности с помощью неравенства (2.9).

Программа вычисления $\sqrt{a}$

1) Введи числа a, ε, x_0, c_0 и переходи к следующей команде.

2) Присвой величине x значение x_0 и переходи к следующей команде.

3) Присвой величине y значение a/x и переходи к следующей команде.

4) Присвой величине y значение $x+y$ и переходи к следующей команде.

5) Присвой величине x значение $(1/2)y$ и переходи к следующей команде.

6) Присвой величине y значение x^2 и переходи к следующей команде.

7) Присвой величине y значение $y-a$ и переходи к следующей команде.

8) Присвой величине y значение y/c_0 и переходи к следующей команде.

9) Присвой величине δ значение $(1/2)y$ и переходи к следующей команде.

10) Сравни δ и ε. Если $\delta>\varepsilon$, то переходи к команде 3), иначе переходи к следующей команде.

11) Напечатай числа x, a, ε и переходи к следующей команде.

12) Прекрати вычисления.

Поясним составленную программу. Команда 2) помещает значение начального приближения x_0 в ячейку памяти, в которой хранятся значения переменной величины x (на каждом этапе вычислений в этой ячейке хранится зна-

чение x, равное значению одного из членов рекуррентной последовательности x_n).

Команды 3)—5) вычисляют по числу x число $(1/2)(x+a/x)$, т. е. делают очередную итерацию. Найденное число помещается в ячейку памяти, в которой хранится значение величин x, при этом старое значение переменной величины x теряется безвозвратно. Это значит, что если бы нам понадобилось старое значение переменной величины x, то для его восстановления нам бы пришлось провести все вычисления заново.

Команды 6)—9) вычисляют величину

$$\delta = \frac{x^2 - a}{2c_0},$$

с помощью которой оценивается сверху разность $x - \sqrt{a}$ (см. (2.9)).

Важное значение имеет команда 10). По этой команде не производятся вычисления, а сравниваются между собой вычисленное значение δ и заданная точность ε. По результату сравнения УУ «принимает решение», что делать дальше. Если $\delta > \varepsilon$, то УУ вернет вычислительный процесс к команде 3) и заставит делать еще одну итерацию. В противном случае, когда требуемая точность достигнута, машина напечатает полученный результат и прекратит вычисления по данной программе.

Заканчивая комментарии к программе, обратим ваше внимание на переменную y. Ей мы присваиваем результаты промежуточных вычислений, которые нужны только для выполнения следующей операции. Мы их не храним долго в памяти машины, а, использовав, тут же «затираем» результатом следующих вычислений. Такой подход позволяет экономить память машины, не загромождая ее ненужной информацией. Для данной задачи это не важно: она слишком проста, но для больших задач экономное распределение памяти машины имеет принципиальное значение, иначе задача в ней может не поместиться.

Теперь перейдем к обсуждению самого главного. Выше мы уже говорили, что программа должна определять последовательность вычислений в «понятной» для машины форме. Данная программа не удовлетворяет этому требованию: она записана в форме, понятной человеку, а не машине. Каждая машина имеет свой язык. Он определяется *системой команд*, т. е. набором операций, которые может выполнять машина, и способом их кодировки. Для каждой ЭВМ свой

ма команд своя, так что *машинные языки* разных ЭВМ отличаются друг от друга. Однако общие принципы построения таких языков одинаковы, поэтому для первого знакомства достаточно рассмотреть язык конкретной ЭВМ.

Чтобы не касаться несущественных деталей, присущих конкретным ЭВМ, возьмем в качестве примера некоторую условную машину со следующей формой записи команд:

КОП A1 A2 A3

Здесь A1, A2 — адреса ячеек памяти, где хранятся числа, над которыми выполняется операция, A3 — адрес ячейки, в которую должен быть записан результат, КОП — код операции, выполняемой согласно данной команде. Все операции, которые «умеет» выполнять ЭВМ, перенумерованы. Числа, выражающие эти номера, называются *кодами операций*.

Приведем таблицу кодов некоторых операций нашей условной машины, нужных для записи программы вычисления $\sqrt{a}$ на машинном языке.

Таблица 3

КОП	Операция
00	Пересылка числа из ячейки A1 в ячейку A3, адрес A2 не используется
01	Сложение
02	Вычитание
03	Умножение
04	Деление
15	Сравнение чисел из ячеек A1 и A2 и передача управления на команду, расположенную в ячейке A3, если первое число больше второго (условная передача управления)

Согласно этой таблице команда

04 050 101 062

означает следующий приказ ЭВМ: возьми число из ячейки с адресом 050, раздели его на число из ячейки с адресом 101 и помести полученный результат в ячейку 062. Предположим, что эта команда хранится в ячейке 425; тогда, выполнив ее, машина перейдет к выполнению команды, которая хранится в ячейке 426.

Процедура последовательного выполнения команд программы конструктивно заложена в ЭВМ. Однако ее можно нарушить с помощью специальных операций передачи управления. В нашей машине такую возможность дает операция 15.

Рассмотрим в качестве примера команду

15 061 051 003

Она выполняется следующим образом: машина сравнивает между собой числа, которые хранятся в ячейках 061 и 051. Если первое число больше второго, то она переходит к выполнению команды из ячейки 003, в противном случае она будет выполнять команду из следующей ячейки.

После этих пояснений приведем программу вычисления $\sqrt{a}$ на языке нашей условной машины. Ради простоты мы не будем в ней явно описывать ввод в машину и вывод на печать нужных чисел, а ограничимся лишь основной частью программы, состоящей из команд 2) — 10). Дело в том, что ввод и вывод, которые должны сопровождаться переводом чисел из одной системы счисления в другую, представляют собой не элементарные операции, а достаточно сложные действия. Они описываются подпрограммами, состоящими из многих команд, и обсуждение этих подпрограмм увело бы нас в сторону от основных вопросов.

Итак, предположим, что числа a, ε, x_0, c_0 уже введены в машину, переведены в двоичную систему счисления и записаны в ячейки 050, 051, 052, 053. Предположим также, что в ячейке 060 записано двоичное представление числа 0,5. Значения переменных δ, y, x, которые вычисляются в процессе исполнения программы, будем помещать в ячейки 061, 062, 101.

Программа, которая вводится в ЭВМ и исполняется ею, приведена на следующей странице в среднем столбце, а левый и правый столбцы представляют собой лишь комментарий к ней. В левом столбце указаны номера ячеек, хранящих соответствующие команды, в правом — в символической форме описаны выполняемые операции.

В заключение отметим, что реально в памяти ЭВМ коды операций и адреса ячеек, которые фигурируют в командах, представлены в двоичной системе. Однако чтобы не усложнять чтение трудно воспринимаемыми двоичными числами, состоящими из длинных цепочек нулей и единиц, мы воспользовались привычной десятичной записью. Для понимания программы это не принципиально.

Программа вычисления $\sqrt{a}$, записанная на машинном языке

Адрес	Команда				Пояснения
		...			Ввод и перевод в двоичную систему счисления чисел a, ε, x_0, c_0
002	00	052	000	101	$x_0 \Rightarrow x$
003	04	050	101	062	$a/x \Rightarrow y$
004	01	101	062	062	$x+y \Rightarrow y$
005	03	060	062	101	$0{,}5 \cdot y \Rightarrow x$
006	03	101	101	062	$x^2 \Rightarrow y$
007	02	062	050	062	$y-a \Rightarrow y$
008	04	062	053	062	$y/c_0 \Rightarrow y$
009	03	060	062	061	$0{,}5 \cdot y \Rightarrow \delta$
010	15	061	051	003	Если $\delta > \varepsilon$, то перейти на команду из ячейки 003
		...			Перевод в десятичную систему счисления и печать чисел x, a, ε
		...			Останов

Составленная программа элементарна, однако она позволяет обсудить ряд специфических особенностей работы на ЭВМ.

1. Всякая ЭВМ, наряду с арифметическими операциями, может выполнять множество других операций. В частности, мы хотели бы обратить внимание на операцию условной передачи управления. Благодаря ей оказалось возможным оборвать бесконечный сходящийся процесс после достижения заданной точности. Такие операции имеются у любой ЭВМ. Они необходимы для решения задач, для которых алгоритм не может заранее однозначно предопределить весь ход вычислений. Вычислительный процесс допускает разветвления, причем выбор нужной ветви приходится «делать» самой машине во время исполнения программы на основании анализа получающихся результатов.

2. Благодаря тому, что алгоритм вычисления сводится к многократному применению рекуррентной формулы, число команд в программе намного меньше числа фактически выполняемых операций. Формула описана в программе только один раз. Это очень важное обстоятельство. Если бы нам пришлось писать программы, в которых число команд равно числу выполняемых операций, то применение ЭВМ оказалось бы практически бессмысленным: время, затраченное на создание и отладку программы, превышало

бы время решения задачи вручную. Рекуррентные формулы, часто встречающиеся в алгоритмах решения различных задач, очень удобны для расчетов на ЭВМ.

3. Команды программы хранятся в памяти машины как обычные числа. Это дает возможность выполнять над ними различные операции, т. е. в ходе решения задачи программа может сама себя перестраивать: изменять в командах коды операций, адреса ячеек, из которых берутся или в которые засылаются данные, убирать одни и добавлять другие команды, менять команды местами и т. д. Все это с учетом возможностей команд передачи управления делает процедуру счета на ЭВМ очень гибкой. Не случайно в 50-х годах при появлении первых ЭВМ их называли вычислительными машинами с гибким программным управлением. Такое название правильно передает наиболее существенную особенность этих замечательных устройств.

§ 3. Поколения ЭВМ

Во введении и первых параграфах этой главы мы подчеркивали, что появление ЭВМ — не дело случая, а результат настойчивого поиска, стимулированного развитием науки и техники, потребностями практики.

Поиск средств автоматизации вычислений велся постоянно, так что многие идеи и принципы, использованные в современных ЭВМ, были высказаны и частично реализованы раньше в других вычислительных устройствах. Поэтому прежде чем обсуждать эволюцию ЭВМ, пути их совершенствования и перспективы дальнейшего развития, полезно, хотя бы очень коротко, рассказать предысторию. Идея создания вычислительных машин с программным управлением была высказана Ч. Бэббиджем (1791—1871 г.), профессором математики Кембриджского университета. Ч. Бэббидж занимался составлением навигационных таблиц, имевших важное значение для такой морской державы, как Великобритания. Внимательное изучение процедур вычислений привело его к мысли о возможности их автоматизации. Ч. Бэббидж разработал проект механического вычислительного автомата, названного им аналитической машиной, у которого были все основные узлы ЭВМ, включая принцип управления с помощью запоминаемой программы. Этот грандиозный проект, опередивший свое время, остался нереализованным.

Была создана только небольшая «разностная машина» со сравнительно простой и жестко заданной программой. Труды Ч. Бэббиджа были опубликованы в 1888 г. после его смерти, и о них забыли. По достоинству его идеи были оценены значительно позже.

В 80-х годах прошлого века Г. Холлериз (США) для проведения переписи населения 1890 г. сконструировал машину, автоматизировавшую процесс обработки данных, и использовал в качестве носителей информации перфокарты. В 1896 г. он основал фирму по выпуску перфокарт и счетно-перфорационных машин. В дальнейшем она была преобразована в фирму IBM (International Business Mashines), которая является в настоящее время крупнейшим производителем ЭВМ.

В 1937 г. профессор университета штата Айова Дж. Атанасов с группой сотрудников начал работу по созданию ЭВМ. Принципиально новая идея Дж. Атанасова состояла в том, что вычислитель (арифметическое устройство) работал в двоичной системе. Были созданы специальные электромеханические блоки для перевода чисел из десятичной системы в двоичную и наоборот. Вторая мировая война не дала довести работу над проектом до конца.

Параллельно с Дж. Атанасовым работу над вычислительным автоматом вел в Гарвардском университете Г. Айткен. В 1937 г. им был предложен проект релейной электромеханической машины. Машина была построена фирмой IBM в 1944 г. и названа Марк-1. Она еще не имела гибко изменяющейся программы и по современным представлениям была очень медленной (операция умножения выполнялась за 3 с). Однако возможность создания автомата, состоящего из многих тысяч логических элементов, была доказана.

Примерно в то же время в Германии была создана вычислительная электромеханическая машина с программным управлением (машина К. Цузе).

В 1945 г. сотрудники Пенсильванского университета П. Эккерт и Дж. Моучли построили машину ENIAK. Это была ламповая машина, содержащая блоки на электронных реле. В отличие от Марк-1, ее можно назвать машиной с автоматическим программным управлением, хотя она еще не имела внутренней памяти.

В 1949 г. в Великобритании была построена машина EDSAC, которая уже обладала всеми необходимыми компонентами современных ЭВМ.

В 1947 г. была начата, а в 1951 г. завершена работа над первой советской ЭВМ, названной МЭСМ. Руководил ее созданием академик С. А. Лебедев.

Наступила эра ЭВМ. В 1954 г. во всем мире насчитывалось около 100 ЭВМ. В последующие годы их число нарастало стремительными темпами: 1965 г. — 40 тыс., 1974 г.— 215 тыс., 1978 г.— 580 тыс., 1983 г.— около 2 млн. Параллельно с ростом числа машин не менее быстро шел процесс их совершенствования. В зависимости от элементной базы центрального процессора и оперативной памяти, технических характеристик (быстродействия, объема памяти, системы команд) и сложности архитектуры принято делить ЭВМ на поколения.

В настоящее время насчитывается 4 поколения машин и уже просматриваются черты следующего, пятого поколения. Различие в технических возможностях ЭВМ разных поколений оказывало прямое воздействие на программное обеспечение и формы общения человека с вычислительной техникой. В настоящем параграфе мы расскажем о поколениях ЭВМ, не затрагивая вопросов их программного обеспечения, которым посвящена следующая глава книги.

Машины первого поколения. В качестве элементной базы центрального процессора ЭВМ первого поколения использовались электронные лампы, общее число которых доходило до нескольких десятков тысяч. Оперативная память строилась на блоках ферритовых сердечников. Использовались также для этой цели, особенно в первых ЭВМ, электронно-лучевые трубки, ртутные линии задержки и т. д. Такие ЭВМ потребляли значительную мощность — до 100 и более киловатт. Из серийных советских машин к первому поколению относятся Стрела (1953 г.), Урал (1954 г.), М-20 (1959 г.), Минск-1 (1960 г.), ряд машин серии БЭСМ.

Основными характеристиками ЭВМ являются быстродействие и объем оперативной памяти. Быстродействие оценивается как среднее число машинных операций, выполняемых за одну секунду. Объем оперативной памяти определяется числом машинных слов, длина машинного слова — числом двоичных разрядов или *битов*. Приведем в качестве примера характеристики одной из наиболее производительных машин первого поколения — машины М-20: быстродействие — 20 тыс. операций в секунду, длина машинного слова — 45 двоичных разрядов (45 бит), объем

оперативной памяти — 4К слов, где К=2^{10}=1024 — «двоичная тысяча».

Машина М-20 имела также внешнюю память, которая состояла из промежуточной памяти в виде 3 магнитных барабанов по 4К слов и 4 магнитофонов с емкостью магнитных лент по 75К слов. Скорость обмена информацией оперативной памяти с магнитными барабанами была сравнительно высокой, с магнитными лентами — медленной: требовалось несколько десятков секунд, а в «неблагоприятном» случае, когда для поиска нужной информации приходилось перематывать почти всю магнитную ленту,— несколько минут.

Быстродействие машины М-20 превышало скорость работы вычислителя с настольной клавишной машиной, равную примерно 2 операциям в минуту, в 600 000 раз. Чтобы получить представление об объеме ее памяти, воспользуемся следующими соображениями. В русском алфавите 32=2^5 букв. Все буквы можно закодировать с помощью 5-разрядных двоичных чисел. В одном 45-разрядном машинном слове можно разместить 9 букв, а вся оперативная память вмещает 9·4096=36 864 буквы или 18 с лишним страниц текста, содержащего 2000 букв на странице (40 строк по 50 букв). Если дополнительно принять во внимание емкость 3 магнитных барабанов, то это число нужно учетверить.

Машины первого поколения имели относительно простую структуру и выполняли программы строго последовательно: центральный процессор переходил к новой команде только после того, как полностью оканчивал выполнение предыдущей. При таком режиме скорость работы ЭВМ существенно снижали «медленные» команды, связанные с обращением к внешним устройствам: магнитным лентам, печатающему устройству, вводу-выводу перфокарт и т. д. Скорость работы этого оборудования была на несколько порядков ниже скорости работы центрального процессора. Например, за время, в течение которого вводилась одна перфокарта, процессор мог выполнить около 10 тыс. арифметических и логических команд.

При работе на машинах первого поколения программист писал программу непосредственно на языке машины, самостоятельно распределяя ячейки оперативной памяти под программу, исходные данные, результаты счета. Разобраться в чужой программе без подробных пояснений автора об ее структуре и расп еделении памяти было практически

невозможно. Это сильно затрудняло обмен программами и особенно модификацию чужих программ.

Для машин первого поколения был характерен *открытый режим* их использования: программист приходил в отведенное ему время в машинный зал, садился за пульт ЭВМ и сам «пропускал» свою программу. Очень много времени и сил занимали «отладки» новых программ, т. е. выискивание и исправление ошибок, проверка программ с помощью пробных тестовых расчетов. Хотя результаты этих расчетов никакого практического интереса, как правило, не представляли (расчеты делались для контроля программы), они «съедали» до 50% дефицитного машинного времени.

В эпоху машин первого поколения, охватившую 50-е годы, началась разработка стандартных и типовых программ, составление библиотек программ на внешних носителях памяти с инструкциями по их использованию. Столкнувшись с типичной задачей или типичным элементом в большой задаче, математик мог прямо воспользоваться готовой программой.

Машины второго поколения. Переход к машинам второго поколения характеризовался прежде всего изменением элементной базы центрального процессора: на смену электронным лампам пришли транзисторы. Они работали более надежно, имели небольшие размеры, потребляли мало энергии и поэтому не так нагревались. Процессор стал более компактным.

Оперативная память по-прежнему строилась на ферритовых сердечниках, однако их размеры удалось существенно уменьшить (наружный диаметр сердечника был доведен до 0,45—1,0 мм). В результате скорость срабатывания устройства, зависящая от массы сердечника, возросла.

Технические возможности новой элементной базы в сочетании с опытом проектирования и эксплуатации машин первого поколения позволили сделать важный шаг в развитии вычислительной техники, повышении ее производительности. Расширилась система команд ЭВМ, усложнилась архитектура. Появилась возможность совмещать по времени некоторые операции, например работу центрального процессора и устройств ввода-вывода.

По своим параметрам машины второго поколения существенно превосходили машины первого поколения: они имели быстродействие порядка 10^4—10^5 операций в секунду и объем оперативной памяти (16—64)К слов. Улучши-

лись технические характеристики внешних запоминающих устройств, расширился парк устройств ввода-вывода.

Различных типов машин второго поколения было очень много. Из серийных отечественных ЭВМ к ним, например, относятся машины МИР, Минск-23, М-220, БЭСМ-4, Минск-32 и др. Ко второму поколению ЭВМ по своей элементной базе относится также машина высокой производительности БЭСМ-6, хотя ее архитектура уже отвечает следующему, третьему поколению ЭВМ.

Рассмотрим на примере ЭВМ БЭСМ-6, какие возможности скрывались за данной выше общей характеристикой машин второго поколения. БЭСМ-6, серийный выпуск которой начался в 1967 г., является одной из самых больших и совершенных машин второго поколения в мире. Ее быстродействие — 1 млн. операций в секунду (оно превышает указанные выше цифры «среднего» быстродействия машин второго поколения), оперативная память — 128К слов, длина слова — 48 бит. Таким образом, БЭСМ-6 в 50 раз быстрее машины М-20, а ее оперативная память примерно соответствует 1 200 000 буквам или книге в 600 страниц. Объем промежуточной памяти на магнитных барабанах — 512К, т. е. в 4 раза больше оперативной памяти. К центральному процессору могут быть подключены 32 магнитофона с емкостью каждой ленты до миллиона машинных слов. На такую ленту можно записать в закодированном виде 5 тыс. страниц текста, т. е. 10-томное издание. С 1972 г. в качестве промежуточной памяти стали также использовать магнитные диски, емкость которых значительно превышала емкость барабанов.

Емкость устройств внешней памяти принято измерять в машинных словах стандартной длины, содержащих 8 двоичных разрядов. Такую единицу называют *байтом*: 1 байт = =8 бит. Используются также более крупные единицы — *килобайт* (кб) и *мегабайт* (Мб): 1 кб=1024 байт, 1 Мб= =1024 кб. Измерение емкости памяти в стандартных единицах дает абсолютную характеристику устройства, не связанную с длиной машинного слова конкретной машины.

Емкость магнитных дисков в эпоху ЭВМ второго поколения составляла 1—30 Мб. На магнитном диске емкостью в 30 Мб можно хранить информацию, соответствующую примерно 15 тыс. страниц текста. На первый взгляд это очень много. Однако в вычислительной практике постоянно встречались задачи, для которых не хватало именно памяти.

Такие задачи стимулировали настойчивые поиски путей повышения емкости запоминающих устройств.

Машины второго поколения обладали более развитой и совершенной системой ввода-вывода. Появились быстродействующие читающие устройства, способные пропускать до 1000 перфокарт в минуту, алфавитно-цифровые печатающие устройства (АЦПУ), графопостроители. АЦПУ дали возможность гибко менять форму выдачи результатов, например, печатать их в виде таблиц со словесным описанием приведенных величин. Все это существенно облегчило обработку результатов расчетов.

Повышение технических характеристик машин второго поколения, особенно таких как БЭСМ-6, расширило возможности их применения, позволило решать большие и сложные задачи. Однако усложнение задач неизбежно связано с усложнением программ. Программирование и отладка грозили стать самым «узким» местом, снижающим эффективность использования ЭВМ. Возникла проблема дальнейшей автоматизации, включения в нее не только вычислительного процесса, но также процесса создания и отладки программ. Важное значение для решения этой проблемы имел переход от программирования на языке машин к программированию на *алгоритмических языках*.

Первые алгоритмические языки появились в конце 50-х — начале 60-х годов. В качестве примера можно привести алгол-60, название которого является сокращением английских слов Algorithmic Language (алгоритмический язык), а число 60 указывает год его появления — 1960. В отличие от машинных языков ЭВМ с их трудно воспринимаемой цифровой формой записи команд, алгоритмические языки более наглядны. Они используют привычную математическую символику и другие легко воспринимаемые изобразительные средства. Фразы этих языков состоят из нужных формул, записанных в обычном, понятном любому математику виде, и из нескольких стандартных терминов на английском языке. Важным достоинством алгоритмических языков является их универсальность и наличие международного стандарта, они совершенно не зависят от конкретного типа машины, для которой предназначена написанная программа. Программист, работающий на алгоритмическом языке, может вообще не знать систему команд машины, ему не нужно переучиваться при переходе с одной машины на другую.

Для того чтобы программа, написанная на алгоритмическом языке, могла быть выполнена ЭВМ, она должна быть сначала переведена с этого универсального языка на собственный язык машины. Делает это сама ЭВМ с помощью специальной программы — *транслятора* *). Транслятор проводит логический анализ программ, написанных на алгоритмическом языке, и осуществляет их перевод на язык машины. Транслятор — очень сложная программа, его создание — большая работа, которую проводит группа высококвалифицированных специалистов по системному программированию. Однако важно подчеркнуть, что обычные пользователи ЭВМ могут ничего не знать ни о самом трансляторе, ни о принципах его работы. Это не помешает им писать программы на алгоритмическом языке и проводить по ним расчеты. Более подробно об алгоритмических языках и трансляторах будет рассказано в § 2 следующей главы.

Итак, в эпоху ЭВМ второго поколения в развитии вычислительной техники, в повышении ее быстродействия и эффективности был достигнут значительный прогресс. Он сопровождался быстрым увеличением общего числа ЭВМ. Однако рост потребностей в вычислительной технике шел еще более быстрыми темпами, опережая возможности ее производства. Вызвано это было расширением сферы применения ЭВМ, увеличением числа пользователей, существенным усложнением решаемых задач. Машинное время, которое было дефицитным в эпоху машин первого поколения, стало еще более дефицитным.

Одно из слабых мест ЭВМ первого и отчасти второго поколения состояло в том, что при вводе и выводе данных, т. е. при «общении» машины с человеком, она не считала, ее «мозг» — центральный процессор — бездействовал, а работало только периферийное оборудование. Механические устройства ввода и вывода, несмотря на существенные усовершенствования, не могли угнаться за быстродействием электронного центрального процессора, и потери его рабочего времени от простоев при вводе и выводе данных были значительными.

Жертвой этого конфликта в условиях острого и постоянно растущего дефицита машинного времени, его высокой стоимости, оказался человек. Началась жесткая борьба за сокращение «контактов» пользователей с машиной, в которой на первое место ставились «интересы» машины, ее

*) Транслятор (по-английски — translator) — переводчик.

центрального процессора, а не удобства пользователей. Одним из ее проявлений стал переход от открытого режима работы на ЭВМ к *закрытому режиму*: математиков-вычислителей «отлучили» от ЭВМ и перестали пускать в машинный зал. Задачи теперь пропускали операторы по инструкциям, составленным авторами программ.

Это новшество внедрялось в вычислительных центрах, как картошка на Руси, при яростном сопротивлении пользователей ЭВМ. Они доказывали, что без них задача считаться не будет, что операторы все перепутают (так, действительно, иногда бывало), что в инструкции нельзя оговорить всех тонкостей и т. д. Однако закрытый режим позволил более эффективно использовать вычислительную технику, существенно сократить непроизводительные потери машинного времени, и это предопределило исход борьбы: сопротивление пользователей было сломлено.

Преимущество закрытого режима состояло в том, что теперь на машине постоянно работали профессиональные операторы. Они знали пульт гораздо лучше большинства математиков-вычислителей, действовали более четко и грамотно, допускали меньше «операторского» брака. Пользователи, лишенные возможности присутствовать при прохождении своих задач через машину и вносить прямо на месте коррективы и исправления в программы, были вынуждены более внимательно готовить задания, заранее предусматривать возможные осложнения и отражать их в инструкциях операторам. Это также давало значительную экономию машинного времени.

Скоро инструкции операторам заменили инструкции самой машине в виде набора стандартных перфокарт, которые добавлялись к программе. Был введен режим *пакетной обработки*: программы собирались одна за другой и ставились в читающее устройство ЭВМ. Машина, закончив очередные расчеты, сама вводила следующую программу, границы которой выделялись специальными картами, и приступала к ее решению. Операторам оставалось только контролировать работу ЭВМ и следить за тем, чтобы ее читающее устройство не пустовало.

Этот комплекс технических и организационных мер существенно повысил эффективность использования вычислительной техники.

Машины третьего поколения. Совершенствование технологии производства полупроводников привело к созданию микроэлектронных устройств, получивших название

интегральных схем. Интегральная схема представляет собой тонкую пластинку химически чистого кристаллического полупроводникового материала (чаще всего кремния) размером около 1 см². На поверхности такого кристалла в результате очень сложного многоэтапного процесса создается многослойная миниатюрная электронная схема, содержащая все необходимые транзисторы, диоды, сопротивления и соединения между ними. Схема помещается в пластмассовый или керамический защитный корпус, снабженный несколькими десятками тонких металлических ножек, так что вся конструкция по внешнему виду напоминает многоногого паука.

В зависимости от числа электронных компонент, смонтированных на одном кристалле, различают малые интегральные схемы (до 100 элементов на кристалл), средние (до 1000 элементов на кристалл) и большие интегральные схемы (БИС), которые могут насчитывать несколько десятков тысяч электронных компонент. В последнее время все большее распространение получают сверхбольшие интегральные схемы (СБИС), на кристалле которых размещены сотни тысяч сверхминиатюрных электронных приборов и соединений между ними. Под микроскопом кристалл СБИС напоминает план большого города с домами, улицами и площадями.

Использование интегральных схем вместо отдельных транзисторов позволило значительно уменьшить габариты узлов ЭВМ, повысить их экономичность и надежность. Достоинства интегральных схем при конструировании сложных электронных устройств стимулировали их непрерывное совершенствование. За последние 20 лет ежегодно происходило примерное удвоение числа электронных компонент, которые размещались на одном кристалле. Соответственно уменьшались размеры отдельных транзисторов и соединений между ними. Сейчас эти размеры удалось довести до нескольких микрон.

Интегральные схемы стали элементной базой центральных процессоров машин третьего поколения. Они используются также в этих машинах в качестве элементной базы оперативной памяти, вытесняя постепенно память на ферритовых сердечниках. Широкое применение в вычислительной технике интегральных схем открыло новые возможности для ее совершенствования, повышения быстродействия до 10^5—10^7 операций в секунду, расширения оперативной памяти до нескольких мегабайт.

В эпоху ЭВМ второго поколения появилось очень много различных типов машин, которые имели близкие характеристики, но отличались системой команд и способом их кодировки. Для каждой из них приходилось разрабатывать свое математическое обеспечение: трансляторы с алгоритмических языков, библиотеку стандартных программ и т. д. Стоимость математического обеспечения быстро росла и нередко стала превышать стоимость самих машин. Это явилось одной из причин, по которой машины третьего поколения, обладающие еще более сложным математическим обеспечением, стали разрабатывать не поодиночке, а семействами. ЭВМ одного семейства могли отличаться быстродействием, объемом памяти, однако все они являлись конструктивно, программно, информационно совместимыми и обладали одинаковым математическим обеспечением. Это существенно снижало расходы на разработку математического обеспечения и предоставляло широкие возможности для наиболее экономного, эффективного использования вычислительной техники в зависимости от характера решаемых задач.

Одним из примеров такого семейства является единая система электронных вычислительных машин (ЕС ЭВМ) третьего поколения, которая создается и выпускается в рамках международного сотрудничества социалистических стран — участниц СЭВ по многостороннему соглашению, подписанному в декабре 1969 г. Выпуск машин единой системы начался в 1972 г. Их вместе с периферийным оборудованием производили и поставляли друг другу страны — участницы соглашения. Осуществление в короткий срок такого крупного проекта явилось претворением в жизнь принципов экономической интеграции социалистических стран, лежащей в основе деятельности СЭВ.

В последующие годы начался выпуск модифицированных моделей ЕС ЭВМ. В настоящее время в рамках единой системы насчитывается около 20 ЭВМ и свыше 200 различных устройств периферийного оборудования, каждое из которых совместимо с любой машиной системы. В единой системе приняты международные стандарты на технические характеристики всех устройств и узлов ЭВМ, на систему кодов, операций, средств программирования. ЭВМ единой системы совместимы с моделями ЭВМ ряда капиталистических стран, например крупнейшей американской фирмы IBM. Это открывает возможности для широкого международного сотрудничества в области использо-

вания вычислительной техники и обмена программным обеспечением.

При разработке машин третьего поколения был учтен богатый опыт эксплуатации ЭВМ второго поколения со всеми их достоинствами и недостатками. Достоинства старались развить дальше, а недостатки — устранить. В результате новые машины отличались от своих предшественниц не только элементной базой, быстродействием, объемом памяти. Они предоставляли пользователям гораздо больше возможностей и удобств в работе, существенно изменяли стиль использования вычислительной техники.

Мы уже отмечали, что для машин первого поколения был характерен открытый режим эксплуатации. В эпоху машин второго поколения он сменился закрытым режимом. Для машин третьего поколения типичной стала работа в *мультипрограммном режиме с разделением времени*, при котором ЭВМ одновременно обслуживала несколько пользователей. Достигнуто это было как за счет конструктивных особенностей машин, так и благодаря созданию специального математического обеспечения.

К конструктивным особенностям прежде всего нужно отнести то, что в третьем поколении ЭВМ было окончательно закреплено разделение машин на отдельные независимые модули: центральный процессор и специальные процессоры для управления устройствами ввода и вывода, названные *каналами ввода-вывода*. Такое разделение является основой мультипрограммного режима, при котором машина работает сразу с несколькими программами и наиболее полно использует возможности своего центрального процессора. Пока центральный процессор проводит расчеты по одной из программ, каналы и подключенные к ним периферийные устройства печатают полученные ранее результаты, вводят и подготавливают к исполнению следующие программы. Закончив очередной этап расчетов, центральный процессор сразу переключается на следующую программу, передав выдачу результатов одному из свободных каналов выдачи.

Режим разделения времени — это особая форма мультипрограммного режима, при котором ЭВМ работает сразу с несколькими пользователями, считая их задачи по очереди малыми порциями. Возможность организации такого обслуживания потребовала существенного усложнения аппаратуры и математического обеспечения ЭВМ. Пришлось решать вопросы распределения оперативной памяти между

различными программами, защиты каждой из них от «вмешательства» со стороны остальных программ, создания системы прерывания, которая осуществляет автоматическое переключение машины с одной программы на другую. Были разработаны специальные операционные системы, обеспечивающие согласованные действия всех модулей ЭВМ. Коротко о них будет рассказано в § 4 следующей главы.

Обычно при работе в режиме разделения времени пользователи связываются с ЭВМ с выносных терминалов *), которые стоят либо в их рабочих комнатах, либо в специально оборудованных помещениях. Вооруженный терминалом, пользователь может ввести в машину свою программу или (это бывает гораздо чаще) вызвать ее с внешнего носителя памяти и произвести нужные расчеты. Получив результаты и ознакомившись с ними, он имеет возможность внести изменения в программу, в исходные данные и продолжить вычисления. При таком режиме пользователь работает с ЭВМ в форме диалога. Если бы он ничего не знал о существовании других терминалов, за которыми одновременно с ним ведут расчеты другие пользователи, то у него была бы иллюзия того, что машина находится в его личном распоряжении.

Так спираль развития вычислительной техники и ее использования человеком завершила очередной виток: математики-вычислители, попавшие «в опалу» в эпоху машин второго поколения, были реабилитированы. Они снова получили прямой доступ к ЭВМ, только теперь и сами машины, и форма связи с ними существенно изменились. Математику не нужно было больше идти в машинный зал, машина сама пришла в его рабочее помещение через выносной терминал. Существенное расширение контактов с машиной и изменение их формы потребовали создания нового периферийного оборудования.

Широкое распространение получили телетайпы, которыми начали пользоваться еще в эпоху машин второго поколения. Они применяются в машинных залах для связи операторов с ЭВМ и в качестве выносных терминалов для пользователей. Телетайп — это электромеханическое устройство, аналогичное телеграфному аппарату, который можно увидеть в любом почтовом отделении. Буквенный и

*) *Выносной терминал* (от латинского terminus — конец, предел) устройство ввода-вывода, расположенное вне машинного зала.

цифровой текст набирают на клавиатуре пишущей машинки телетайпа. Он печатается на бумаге и одновременно в закодированном виде (как телеграмма) посылается по кабелю в ЭВМ. Машина может отвечать на вопросы, заданные ей через телетайп, выдавать результаты расчетов. Ее ответы будут восприняты устройством, декодированы и отпечатаны пишущей машинкой.

Большие удобства пользователям предоставляет терминальное устройство с электронно-лучевой трубкой, получившее название дисплея *). Дисплеи имеют клавиатуру, и с них, как и с телетайпов, можно вводить программу и данные в ЭВМ, только в этом случае набранный текст не печатается, а высвечивается на экране. Информация, которая выводится из машины, также подается на экран. Она может иметь форму либо алфавитно-цифрового текста, либо графического материала. Это делает работу с дисплеем особенно удобной, поскольку позволяет представить результаты расчетов не только в виде таблиц, но и в виде графиков, наглядность которых существенно упрощает последующий анализ.

При моделировании на ЭВМ сложного объекта появляется возможность увидеть на экране дисплея его поведение и, в случае необходимости, снять с экрана на кинопленку. Такие кинофильмы стали одним из распространенных способов представления результатов расчета.

Комбинация фотоэлемента и электронно-лучевой трубки дисплея используется в качестве прямого устройства ввода в ЭВМ информации, зафиксированной на фотопленке. Пленка помещается между трубкой и фотоэлементом. Когда луч пробегает экран, фотоэлемент фиксирует точки экрана, которые оказались скрытыми непрозрачными участками пленки. Информация о почернении пленки преобразуется в числовую информацию и вводится в ЭВМ. Такие устройства оказались спасением для физиков: они позволили автоматизировать обработку огромного числа (десятки, сотни тысяч) фотографий треков частиц в пузырьковых камерах.

Дисплеи предоставляют очень широкие возможности для работы с машиной в форме диалога. Они непрерывно совершенствуются: появились цветные дисплеи, а также дисплеи с собственным процессором и памятью, получившие название «интеллектуальных» дисплеев.

---

*) Дисплей (по-английски — display) — показ, выставление напоказ, выставка.

Машины четвертого поколения. Машины четвертого поколения — это новый шаг в развитии вычислительной техники. Их быстродействие доведено до 10^7—10^8 операций в секунду. Такая высокая скорость работы достигнута в первую очередь за счет распараллеливания вычислений: центральный процессор этих ЭВМ состоит из нескольких независимых блоков, которые могут одновременно выполнять большое число операций. Еще более высокую производительность имеют специализированные процессоры, приспособленные для выполнения векторных и матричных операций над массивами чисел. В задачах, содержащих много операций подобного типа, быстродействие специализированных процессоров может доходить до 10^9 операций в секунду.

Элементной базой машин четвертого поколения являются БИС и СБИС. Их применение позволило создать мощные, наделенные широкими возможностями процессоры, увеличить объем оперативной памяти и в то же время уменьшить размеры машин. Последнее обстоятельство имеет важное значение не только для удобного, компактного размещения вычислительной аппаратуры, но и для ее быстродействия. Действительно, сигнал, распространяющийся со скоростью света, проходит 3 м за время $t=10^{-8}$ с. Если бы машина выполняла все операции строго последовательно и после каждой операции пересылала информацию на расстояние порядка 3 м, то поднять ее быстродействие выше 10^8 операций в секунду было бы принципиально невозможно. По-видимому, в недалеком будущем центральные процессоры мощных, сверхбыстрых ЭВМ будут иметь размеры порядка десятка сантиметров.

Большие успехи достигнуты в области совершенствования периферийного оборудования, особенно внешних запоминающих устройств большой емкости. Созданы магнитные диски емкостью в тысячи мегабайт. Поперечная плотность записи на них доведена до нескольких десятков дорожек на 1 мм. Для сравнения укажем, что в кассетных магнитофонах на ленте шириной порядка 4 мм располагается от 2 до 4 дорожек, на долгоиграющих пластинках записывается около 10 дорожек на 1 мм.

Существенные изменения претерпевает программирование для машин четвертого поколения. Новые входные языки предоставляют возможности для описания сложных структур данных, для параллельной обработки информации. Много внимания уделяется повышению производительности

труда программистов. Разрабатываются специальные комплексы по производству программного продукта по принципам промышленного производства. Создаются ЭВМ с нетрадиционной структурой и схемой работы: потоковые ЭВМ, однородные вычислительные среды, ЭВМ с программно-перестраиваемой структурой и др.

Машины четвертого поколения принесли новую технологию использования вычислительной техники: ЭВМ стали объединять в комплексы и сети. *Вычислительный комплекс* строится на базе нескольких территориально близко расположенных ЭВМ. Он может состоять как из одинаковых, так и из разных машин. В первом случае говорят об *однородном* вычислительном комплексе, во втором — о *неоднородном*.

Создание однородных вычислительных комплексов повышает мощность и надежность: в этом случае выход из строя отдельных машин снижает производительность комплекса, но не лишает его работоспособности. Обычно ЭВМ, включенные в комплекс, имеют либо общую оперативную память *(многопроцессорные системы)*, либо общее поле внешних устройств.

Неоднородные вычислительные комплексы создаются на базе разных ЭВМ. Каждая из них имеет свою архитектуру, систему команд и выполняет в рамках комплекса определенные функции. Главные ЭВМ обеспечивают исполнение программ пользователей, процессоры ввода-вывода отвечают за работу внешних устройств, связные процессоры осуществляют связь комплекса с более широким объединением — сетью ЭВМ. В состав комплекса могут также входить специализированные ЭВМ, ориентированные на быстрое выполнение операций над векторами и матрицами.

На вычислительных комплексах реализуется распределение обработки данных: несколько программ обрабатывается параллельно, и каждая из них одновременно загружает работой несколько машин и устройств ввода-вывода.

Вычислительные комплексы и отдельные машины объединяются в более крупные системы — *сети ЭВМ*. Сети могут быть очень разветвленными и охватывать машины, отстоящие друг от друга на тысячи километров. В узлах сетей находятся мощные комплексы, предназначенные для организации «коммунального» обслуживания пользователей. Узлы соединяются между собой высокоскоростными ли-

ниями связи, на пересечении которых находятся связные ЭВМ. Пользователи входят в сеть через специальные ЭВМ — концентраторы терминалов. К каждому такому концентратору подключаются их терминалы или персональные компьютеры. Когда абоненту вычислительной сети не хватает производительности своих вычислительных средств, он посылает программу для проведения расчетов в мощные узловые вычислительные комплексы.

Кроме обслуживания вычислительными мощностями, сеть предоставляет своим абонентам еще один вид обслуживания — информационный. Абонент получает доступ к банкам данных, содержащим сведения самого различного характера, которые было бы невозможно собрать в каком-нибудь одном вычислительном центре.

Развитие системы коммунального обслуживания вычислительными и информационными услугами требует не только совершенствования аппаратурного и программного обеспечения ЭВМ, но и решения многих других проблем. Некоторые из них выходят на общегосударственный уровень.

К таким проблемам в первую очередь относится развитие средств связи, пригодных для использования в вычислительном деле: телефонных и телеграфных каналов, спутниковой связи, волоконнооптических линий и т. д. Решение проблемы связи необходимо для еще более широкого и эффективного использования вычислительной техники в различных областях человеческой деятельности: в науке, на производстве, в культуре и образовании.

§ 4. Мини- и микрокомпьютеры

Первоначально развитие вычислительной техники шло только в направлении неуклонного увеличения мощности и сложности машин. Однако в середине 60-х годов, когда стало абсолютно ясно, что область возможного применения ЭВМ далеко выходит за рамки быстродействующего вычислительного автомата, в развитии вычислительной техники появилось новое направление. Дело в том, что использование ЭВМ в «невычислительной» сфере (управление производственными процессами или сложными научными экспериментами, контроль качества продукции, обработка текстов и т. д.) предъявляет к ним специфические требования. Во многих случаях на первое место выдвигается не быстродействие, объем памяти, длина машинного слова, опреде-

ляющая точность, а надежность, простота технического обслуживания, низкая стоимость. Потребность в вычислительной технике специального типа привела к появлению *миникомпьютеров* — ЭВМ с укороченным машинным словом, ограниченным набором команд и небольшой оперативной памятью.

Чтобы характер этого новшества в развитии вычислительной техники стал более ясным, приведем некоторые данные первого миникомпьютера PDP-5. Он был разработан в 1963 г. американской фирмой DEC (Digital Equipment Corporation) для управления ядерными реакторами и имел длину машинного слова 12 бит, оперативную память — 4К=4096 слов, простую систему команд, которая включала единственную арифметическую команду — сложение целых чисел. Ценой таких жестких ограничений удалось создать небольшую настольную машину, экономичную в отношении энергопотребления, высоконадежную, дешевую, с минимальными требованиями к техническому обслуживанию (на уровне бытового телевизора). В то же время необходимо подчеркнуть, что, в отличие от специализированных управляющих устройств с «запаянной» логикой, это была программируемая ЭВМ. Поэтому сфера ее применения не ограничилась реакторами, для которых она создавалась. Машина получила широкое распространение и использовалась в качестве управляющей и контролирующей в самых различных областях.

Миникомпьютер PDP-5 и созданная на его основе модель PDP-8 (1965 г.) положили начало серийному производству и массовому распространению новых машин. Их выпуск нарастал лавинообразно. По мере роста производства увеличивалась степень его автоматизации на базе самих же миникомпьютеров, улучшались технические характеристики, снижалась цена. Низкая стоимость, простота эксплуатации позволили использовать миникомпьютеры там, где применение обычных макрокомпьютеров было бы либо невозможно, либо экономически нецелесообразно.

В СССР серийно выпускаются миникомпьютеры СМ-3 и СМ-4. Они имеют 16-разрядное машинное слово, что является в настоящее время общепринятым стандартом для миникомпьютеров. Быстродействие машины СМ-4 измеряется сотнями тысяч операций в секунду, объем оперативной памяти — 124К слов. В качестве внешней памяти к СМ-4 может быть подключено до 8 магнитных дисков емкостью

по 5 Мб. Машины СМ-3 и СМ-4 программно совместимы как между собой, так и с получившими широкое распространение машинами семейства PDP-11 фирмы DEC.

Миникомпьютеры хорошо адаптируются к работе в самых разных условиях, и сегодня можно указать множество «освоенных ими специальностей». Они используются в качестве встроенных управляющих машин в сложных приборах, механизмах, станках, на автоматических производственных линиях. Миникомпьютерами оснащаются научные лаборатории и медицинские учреждения. Они находятся на борту самолетов и космических ракет. Миникомпьютеры работают в автоматизированных системах обработки текстов, в обучающих и контролирующих системах. Устройство миникомпьютеров позволяет в случае необходимости подключать дополнительное оборудование, наращивать мощность, переходить от единичных машин к миникомпьютерным системам. Введение дисков и терминалов создает хорошие условия для интерактивной обработки информации. Добавление аналого-цифровых преобразователей и мультиплексеров обеспечивает возможность сбора данных в режиме реального времени в автоматизированных экспериментах. Доукомплектование блоком «расширенной арифметики» и дополнительными блоками памяти позволяет успешно использовать миникомпьютеры для решения вычислительных задач.

По мере расширения сферы использования миникомпьютеров создавалось соответствующее математическое обеспечение: операционные системы, системы программирования, библиотеки и пакеты прикладных программ. Обычно типовое математическое обеспечение миникомпьютера включает несколько операционных систем (дисковую, разделения времени, реального времени), системы программирования на языках ассемблер, бейсик, фортран, паскаль, а также на некоторых специальных языках, приспособленных для управления процессами в режиме реального времени и для планово-экономических расчетов. Развитые библиотеки и пакеты для миникомпьютеров содержат сотни прикладных программ разного назначения.

Элементной базой миникомпьютеров являются интегральные схемы со степенью интеграции от 10—20 до 100—200 транзисторов на одном кристалле. Переход к БИС позволил сделать следующий шаг в развитии миниатюрной вычислительной техники: появилась возможность выполнить в виде одной БИС целый процессор. Такие процессоры

получили название *микропроцессоров*, а созданные на их основе ЭВМ — *микрокомпьютеров*. Остальные элементы микрокомпьютеров (оперативная память, устройства сопряжения с периферийным оборудованием) также реализуются в виде БИС. Все это размещается на одной или нескольких печатных платах. Имеются даже *однокристальные микрокомпьютеры*, выполненные целиком в виде одной БИС. Микрокомпьютер вместе с подключенным к нему периферийным оборудованием образует *микрокомпьютерную систему*.

Микрокомпьютеры в своем развитии повторили путь миникомпьютеров, только сделали это в более быстром темпе. Первые микрокомпьютеры появились в начале 70-х годов на базе 4-разрядного микропроцессора. Вскоре после этого были созданы 8-разрядные микрокомпьютеры. К началу 80-х годов выпуск 8-разрядных микрокомпьютеров исчислялся миллионами штук в год, 4-разрядных — десятками миллионов. Прогрессирующая технология производства БИС позволила довести число транзисторов на одном кристалле до десятков тысяч. В результате появилась возможность создать 16-разрядные микрокомпьютеры, которые по всем параметрам превосходят миникомпьютеры при меньшей стоимости, большей надежности и практически неограниченной возможности тиражирования. В настоящее время идет процесс постепенного вытеснения миникомпьютеров микрокомпьютерами. В СССР выпускаются серии микрокомпьютеров Электроника-60, Электроника НЦ-80 и др. Они программно совместимы между собой и с миникомпьютерами СМ-3, СМ-4.

Успехи новой техники небывало расширили применение ЭВМ как эффективного средства управления и автоматизации. Особенно широко мини- и микрокомпьютеры используются в качестве элементов, встроенных для этой цели в сложные приборы, механизмы, станки. Однако в сфере решения вычислительных задач они пока коренных изменений не принесли. Причина заключается в том, что вычислительные мощности малых машин ограничены, а операционные системы не могут обеспечить тех удобств, которые предоставляют пользователям операционные системы больших ЭВМ.

В середине 70-х годов в развитии микрокомпьютеров наметилась новая тенденция, связанная с появлением так называемых *персональных компьютеров*. Персональный компьютер — это микрокомпьютерная система, оформленная

в виде единого, компактного прибора. Он предоставляет своему хозяину широкий комплекс технических средств, необходимых для разработки, отладки и выполнения программ. Ориентация на одного пользователя существенно упрощает операционную систему и другое программное оснащение, обеспечивает достаточно высокий уровень вычислительного сервиса.

В настоящее время имеется несколько разновидностей персональных компьютеров, различающихся по назначению, возможностям, цене. Простейшие, так называемые «домашние» компьютеры, предназначены в первую очередь для программируемых игр. Они оформлены в виде прибора с клавиатурой и небольшим дисплеем, имеют оперативную память 4—16 кб. В качестве внешней памяти часто используется обычный кассетный магнитофон. Более мощную разновидность представляет собой учебный компьютер. Он имеет оперативную память 48—64 кб, встроенную внешнюю память на гибких дисках и небольшое печатающее устройство.

Наиболее совершенная разновидность персональных компьютеров рассчитана на профессионалов. У таких компьютеров оперативная память достигает сотен килобайт, внешняя — десятков мегабайт. Они имеют богатый набор периферийного оборудования, развитое математическое обеспечение и по уровню вычислительного сервиса приближаются к большим ЭВМ. Кроме того, как мы уже говорили в предыдущем параграфе, персональные компьютеры могут быть использованы в качестве терминальных устройств для вхождения в сеть ЭВМ. Подключаясь с помощью своего персонального компьютера через концентратор терминалов к сети, пользователь получает доступ к вычислительным мощностям главных ЭВМ и к информации, хранящейся в банках данных сети.

В заключение сделаем некоторые общие замечания о путях дальнейшего развития вычислительной техники. Не будем говорить о совершенствовании элементной базы, увеличении быстродействия, расширении памяти — эти тенденции действуют постоянно, и пока их ослабления не заметно.

Наиболее существенное отличие машин следующего, пятого поколения от сегодняшних будет связано с повышением «интеллектуальных возможностей». Многие функции, которые в машинах четвертого поколения реализуются с помощью сложных программных комплексов, будут

заложены непосредственно в аппаратуру ЭВМ. В свою очередь их программное обеспечение должно быть направлено на развитие эвристических способностей. Нужно «научить» машины делать логические выводы из имеющейся информации, самостоятельно разрабатывать программы решения сложных задач по описанию их постановки на языке, близком к естественному языку человека. Широкое распространение получат системы речевого ввода и вывода, которые дадут возможность вести с машиной диалог в буквальном смысле слова. Все эти усовершенствования позволят ЭВМ играть еще более важную роль во многих областях интеллектуальной деятельности человека.

ГЛАВА 4

ПРОГРАММИРОВАНИЕ. МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ЭВМ

§ 1. Прикладное и системное программирование

Первые ЭВМ поставлялись в вычислительные центры заводами-изготовителями в «голом» виде, без какого-либо вспомогательного программного обеспечения. Поэтому все операции, которые нужно было выполнить ЭВМ для решения задачи, математику-прикладнику приходилось полностью описывать в своей программе. Кроме того, при открытом режиме эксплуатации вычислительной техники он, как правило, сам работал за пультом в качестве оператора ЭВМ, пропуская программу в процессе ее отладки или счета. Все это занимало массу непроизводительно затраченного времени и требовало широкой специальной подготовки, которой в ту пору обладали немногие.

Однако очень скоро было осознано, что значительную часть вспомогательной, рутинной работы, связанной с использованием ЭВМ, можно переложить на машины. Для этого нужно заранее написать соответствующие программы, а затем использовать их каждый раз, когда в этом возникает необходимость. Например, можно составить программы для перевода чисел из десятичной системы счисления в двоичную и наоборот с тем, чтобы использовать их при вводе и выводе чисел. В результате пользователь ЭВМ получит возможность иметь дело только с числами, записанными в привычной десятичной системе, совершенно не «соприкасаясь» с двоичными числами, с которыми оперирует ЭВМ. Можно разработать программу, считывающую и воспринимающую тексты-приказы с операторского терминала. Такая программа позволит перейти от управления работой ЭВМ с помощью кнопок и тумблеров на пульте к более простой и удобной форме управления с помощью словесных

приказов. При наличии большого развитого комплекса таких вспомогательных программ работа на ЭВМ существенно упрощается как для программистов, так и для персонала, обслуживающего машину.

Мы уже говорили в предыдущей главе о том, что открытый режим эксплуатации ЭВМ просуществовал сравнительно недолго. На смену ему пришел закрытый режим: пользователей перестали пускать в машинные залы, а всю работу на ЭВМ поручили специальным группам операторов. Что касается программирования, то его развитие пошло дальше по двум направлениям. Первое из них — *прикладное программирование*, связанное с разработкой программ для решения конкретных научно-технических задач (моделирование физического процесса, вычисление траектории спутника, расчет инженерной конструкции, управление технологическим процессом и т. д.). Людей, разрабатывающих такие программы, обычно называют *прикладными программистами*. Среди них можно встретить как профессиональных специалистов по вычислительной математике, так и физиков, инженеров, экономистов, которые обладают навыками программирования и знакомы с использованием ЭВМ в своей области знаний.

Второе направление в программировании принято называть *системным программированием*. В его задачу входит разработка специальных программ, автоматизирующих процесс написания и отладки прикладных программ, обеспечивающих эффективное использование ЭВМ при их исполнении. Совокупность таких вспомогательных программ называется *математическим* или *программным обеспечением ЭВМ*, а разработчики этих программ — *системными программистами*.

Прикладные программы являются конечной целью программирования. Если сравнить их создание с изготовлением товаров народного потребления (продуктов питания, одежды, обуви, мебели), то системное программирование будет играть роль производства средств производства (машин, станков, инструментов и т. д.).

В настоящее время ни одна ЭВМ не поставляется заводом-изготовителем без математического обеспечения. При этом в комплексе «аппаратура + математическое обеспечение», называемом *вычислительной системой*, непрерывно возрастает ведущая роль второго компонента. Именно в математическом обеспечении заключается главное богатство современных ЭВМ. Характерно, что затраты на его

разработку составляют, как правило, свыше 70% общих затрат на создание вычислительной системы.

В математическом обеспечении современных ЭВМ можно выделить три основных элемента.

1. *Системы программирования.* Они включают языки программирования, трансляторы с них и некоторые вспомогательные средства, упрощающие отладку и модификацию программ.

2. *Пакеты прикладных программ.* Пакетом называют набор программ, предоставляющих пользователю практически готовые средства решения задач из какой-нибудь определенной области знаний.

3. *Операционные системы.* Это комплекс программ, управляющий выполнением всех других программ и обеспечивающий эффективность использования аппаратуры ЭВМ.

В следующих параграфах мы расскажем о перечисленных элементах математического обеспечения ЭВМ с точки зрения тех возможностей, которые они предоставляют для прикладного программирования и расчетов на ЭВМ.

§ 2. Языки программирования. Трансляторы

В § 2 предыдущей главы была приведена программа вычисления $\sqrt{a}$, написанная на машинном языке. На этом элементарном примере хорошо видны недостатки машинного языка как языка программирования. Цифровая форма записи команд, необходимость разбивать алгоритм на мелкие операции делают программу не наглядной и громоздкой, затрудняют ее отладку. «Индивидуальный характер» языков ЭВМ исключает прямой перенос программы с машины одного типа на машину другого типа. Процесс программирования на машинном языке сложен и трудоемок, он требует тщательности, большого внимания, хорошего знания особенностей ЭВМ, на которой предстоит проводить расчеты. Выполнить качественно такую работу могут только специально подготовленные люди. Не случайно в эпоху первых ЭВМ основную массу пользователей составляли профессиональные математики-программисты. Лишь немногие представители других специальностей (физики, инженеры) решались овладеть программированием на машинном языке, остальные были вынуждены идти со своими задачами «на поклон» к профессионалам.

Эти трудности в использовании ЭВМ были преодолены благодаря разработке специальных *языков программирова-*

ния (алгоритмических языков). Они существенно упростили процесс программирования, сделали его не зависящим от конкретной ЭВМ, способствовали тому, что прямой доступ к вычислительной технике смогли получить представители самых разных областей знаний.

Первый шаг в разработке языков программирования состоял в простой замене цифровой формы записи команд на более удобную для восприятия — символьную. Такой язык получил название *автокода*. При записи программы на автокоде вместо цифровых кодов операций указывают их названия (обычно в сокращенном виде). Например, вместо кодов 00, 02, 15 записывают ПЕРЕС (пересылка), ВЫЧ (вычитание), ПБОЛ (переход «по больше»). Вместо же адресов ячеек, в которых хранятся числа, указывают символы соответствующих переменных: X, DELTA, EPSILON и т. д. (Печатающие и читающие устройства, которые используются при работе с вычислительной техникой, не «знают» греческого алфавита, так что вместо греческих букв приходится употреблять другие обозначения, например названия этих букв.) Перед каждой командой, на которую возможен переход из другого места программы, записывается какое-нибудь слово — *метка*. При этом в самой команде перехода указывается метка, а не цифровой адрес соответствующей команды. Символьные обозначения переменных и метки, служащие для выделения нужных команд, называются *именами*.

Программы на автокоде гораздо более просты и наглядны, чем программы на машинном языке: по содержательным символам легче установить, какие операции и над какими величинами выполняются, куда и в каком случае передается управление и т. д. Приведем в качестве примера программу вычисления $\sqrt{a}$, записанную на автокоде (см. с. 92), и сравним ее с программой на машинном языке (см. с. 65). Чтобы идентичность двух программ была полной, мы и здесь не будем описывать операций ввода и вывода чисел.

Заканчивая обсуждение автокодов, подчеркнем, что при всех своих несомненных достоинствах они являются лишь трансформированными копиями машинных языков с их индивидуальными особенностями и зависимостью от ЭВМ. В связи с этим автокоды называют *машинно-ориентированными языками*.

Следующим шагом в развитии языков программирования было появление *проблемно-ориентированных языков*. В этом названии нашел отражение тот факт, что при их

разработке идут не «от машины», а «от задачи»: в языке стремятся максимально полно учесть специфику класса задач, для решения которых его предполагается использовать. Например, для многих научно-технических задач характерны большие расчеты по сложным формулам, поэтому в ориентированные на такие задачи языки вводят удобные средства для их записи. Использование понятий, терминов, символов, привычных для специалистов соответствующей области знаний, облегчает им изучение языка, упрощает процесс составления и отладки программ.

Программа вычисления $\sqrt{a}$, записанная на автокоде

Команды			Примечания
Названия операций и		переменных	
	...		Ввод и перевод в двоичную систему счисления чисел a, ε, x_0, c_0
	ПЕРЕС	X0, 0, X	$x_0 \Rightarrow x$
ЦИКЛ	ДЕЛ	A, X, Y	$a/x \Rightarrow y$
	СЛОЖ	X, Y, Y	$x+y \Rightarrow y$
	УМН	HALF, Y, X	$0{,}5 \cdot y \Rightarrow x$
	УМН	X, X, Y	$x^2 \Rightarrow y$
	ВЫЧ	Y, A, Y	$y-a \Rightarrow y$
	ДЕЛ	Y, C0, Y	$y/c_0 \Rightarrow y$
	УМН	HALF, Y, DELTA	$0{,}5 \cdot y \Rightarrow \delta$
	ПБОЛ	DELTA, EPSILON, ЦИКЛ	Если $\delta > \varepsilon$, то перейти на команду ЦИКЛ
	...		Перевод в десятичную систему счисления и печать чисел x, a, ε
	...		Останов

Проблемно-ориентированные языки обычно называют *алгоритмическими языками высокого уровня*. Уровень языка характеризует степень его близости к естественному, человеческому языку. Машинный язык не похож на человеческий. Он крайне беден в своих изобразительных средствах. Поэтому при записи на нем программ приходится пользоваться длинными и внешне однообразными фразами, а алгоритм разбивать на множество мелких, элементарных операций. Средства записи программ на алгоритмических языках высокого уровня более выразительны и привычны для человека. Например, алгоритм вычисления по сложной формуле не разбивается на отдельные операции, а записы-

вается компактно, в виде одного предложения с использованием привычной математической символики. Составить свою или понять чужую программу на таком языке гораздо проще.

Важным преимуществом алгоритмических языков высокого уровня по сравнению с машинным языком или автокодом является их универсальность, независимость от ЭВМ. Программа, написанная на таком языке, может выполняться на разных машинах. Составителю программы не нужно знать систему команд ЭВМ, на которой он предполагает проводить вычисления. При переходе на другую ЭВМ программа не требует переделки.

Алгоритмические языки высокого уровня в силу своей лаконичности и строгости правил построения предложений дисциплинируют мышление. Логическое несовершенство метода, скрытые изъяны сравнительно легко обнаруживаются при попытке его описания на алгоритмическом языке. Такие языки — не только средство общения человека с машиной, но и людей между собой. Программа, написанная на алгоритмическом языке, легко может быть понята любым специалистом, который знает язык и характер задачи. В сочетании с фактором машинной независимости это существенно упростило обмен программами между различными научными центрами.

В качестве примера алгоритмического языка высокого уровня рассмотрим алгол-60, уже упоминавшийся в предыдущей главе. Программы на алголе записываются с помощью нескольких служебных слов на английском языке и символов (букв, цифр, математических знаков), которые используются для обозначения переменных и записи формул. Служебными словами в алголе являются слова: **begin** — начало, **end** — конец, **real** — вещественный, **go to** — перейти к, **if** — если, **then** — тогда и некоторые другие. В текстах программ они выделяются: в печатном издании набираются полужирным шрифтом, в рукописном тексте подчеркиваются. Формулы записываются в виде, близком к общепринятому, с учетом двух правил:

1) формулы «вытягиваются» в строку, «многоэтажные» записи формул не допускаются;

2) для каждой операции используется только один знак; для сложения, вычитания, умножения, деления и возведения в степень такими знаками являются $+$, $-$, $\times$, $/$, $\uparrow$.

Первое правило связано с тем, что любую информацию вводят в ЭВМ последовательно символ за символом. Второе правило касается в первую очередь умножения и деления, которые мы привыкли в повседневной практике обозначать по-разному. В алголе этого делать нельзя. Не разрешается употреблять в качестве знака умножения точку, в качестве знака деления — двоеточие или горизонтальную черту, не разрешается также опускать знак умножения. При использовании знака возведения в степень ↑ слева от него указывается основание степени, справа — показатель. Например, вместо a^3, d^2 пишут $a \uparrow 3$, $d \uparrow 2$. Такая форма записи обусловлена первым правилом, которое не допускает выхода за пределы строки. С учетом сделанных замечаний формула

$$\frac{a^3+bc}{b-d^2}$$

запишется на алголе следующим образом:

$$(a\uparrow 3+b\times c)/(b-d\uparrow 2).$$

Каждая программа на алголе начинается и заканчивается служебными словами **begin** и **end**. Между ними записывается ее текст. Он состоит из предложений, которые по своему характеру делятся на *описания* и *операторы*. Назначение описаний — дать информацию о свойствах фигурирующих в программе величин. С помощью операторов указываются действия, которые должны быть над ними выполнены. Различные действия задаются операторами разного типа: операторами присваивания, условными операторами и т. д. Их смысл легко понять на конкретном примере, в качестве которого снова возьмем программу вычисления $\sqrt{a}$. Запишем ее теперь на алголе-60:

```
          begin
          real a, epsilon, x, x0, c0;
          input (a, epsilon, x0, c0);
          x := x0;
cycle:    x := (a/x+x)/2;
          if(x↑2-a)/(2×c0) > epsilon
          then go to cycle;
          output (x, a, epsilon)
          end
```

Прокомментируем эту программу. Служебные слова **begin** и **end** указывают ее начало и конец.

Первое предложение программы, начинающееся служебным словом **real**, является описанием. В нем перечисляются названия всех использованных переменных *) и указывается, что их значениями будут вещественные числа.

Второе предложение — оператор ввода переменных *a*, *epsilon*, *x*0, *c*0 (*input* — ввести). Он не только вводит соответствующие числа в машину, но и переводит их из десятичной системы счисления в двоичную.

Третье и четвертое предложения являются операторами присваивания. Справа от знака этого оператора := записывается формула, по которой должны проводиться вычисления, слева указывается переменная, которой присваивается полученный результат. Отметим, что в нашей программе при выполнении первого из этих операторов никаких вычислений фактически проводить не нужно, просто переменной *x* присваивается значение переменной *x*0. Перед вторым оператором присваивания поставлена метка *cycle* (цикл). Она позволяет ссылаться на данный оператор из других мест программы.

Пятое предложение программы, занимающее две строки, начинается словом **if**. Оно представляет собой условный оператор. Принцип действия этого оператора состоит в следующем: проверяется условие, сформулированное между словами **if** и **then**

$$\frac{x^2 - a}{2c_0} > \varepsilon.$$

Если неравенство верно, то нужно выполнить оператор, который стоит после слова **then**. В программе это оператор перехода **go to**. Он отправляет к предложению с меткой *cycle*, чтобы вычислить новую итерацию. В противном случае, когда неравенство не выполняется, машина переходит к следующему шестому предложению. В нем стоит оператор вывода на печать переменных *x*, *a*, *epsilon* (*output* — вывести) с переводом их из двоичной системы счисления в десятичную. На этом выполнение программы заканчивается.

Итак, знакомясь с программированием, мы привели четыре варианта программы вычисления $\sqrt{a}$ (см. с. 61, 65, 92, 94). Самой простой, наглядной и компактной является

*) Обратим внимание на то, что для переменных x_0, c_0 использованы обозначения *x*0, *c*0. Мы уже отмечали выше, что применять обозначения с подстрочными и надстрочными индексами в алголе нельзя. Для обозначения переменной ε, как и в автокоде, использовано название той греческой буквы *epsilon*.

программа на алголе-60. Она выигрывает не только у программ на машинном языке и автокоде, но даже у самой первой программы на описательном «человеческом» языке.

Выше уже отмечалось, что каждый проблемно-ориентированный язык предназначен для программирования задач определенного класса. Специализация обусловлена существенными различиями в задачах, решаемых с помощью вычислительной техники. Разнообразие классов задач привело к тому, что на сегодняшний день разработано несколько сотен алгоритмических языков. Правда, широкое распространение и международное признание получили лишь 10—15 языков. Среди них в первую очередь нужно отметить фортран и алгол-60, предназначенные для решения научно-технических задач, кобол — для решения экономических задач, симула — для решения задач моделирования, бейсик — для решения небольших вычислительных задач в диалоговом режиме работы ЭВМ, лисп — для решения задач обработки символьной информации.

В принципе каждый из этих языков можно использовать для решения задач не своего класса. Однако, как правило, применение оказывается неудобным. В то же время математикам-прикладникам нередко приходится менять сферу своей деятельности. Такой переход может потребовать изучения нового языка, что создает определенные трудности. В связи с этим в середине 60-х годов начали разрабатывать алгоритмические языки широкой ориентации. Обычно они строились по принципу объединения возможностей узкоориентированных языков. Среди них наиболее известны ПЛ/1, паскаль, ада. Однако, как любое универсальное средство, широкоориентированные языки во многих конкретных задачах оказываются менее эффективными. Поэтому наряду с тенденцией к универсальности имеется тенденция к специализации.

В последние годы начата разработка языков нового типа — так называемых *непроцедурных языков* или *языков спецификаций* (описаний). Они принципиально отличаются от упоминавшихся языков тем, что предназначены не для записи алгоритма решения, а лишь для описания постановки задачи (что дано, что требуется получить). При этом поиск решения задачи возлагается непосредственно на ЭВМ. Разработка непроцедурных языков связана с появлением пакетов прикладных программ, о которых будет рассказано в следующем параграфе. Начаты также исследования, направленные на разработку средств общения человека с

ЭВМ на естественном языке (русском, английском и т. д.). Пока они далеко не продвинулись, так как сталкиваются с огромными трудностями. Суть проблемы не в том, чтобы ввести в машину текст на естественном языке (это делается давно), а в том, чтобы научить ее такой текст «понимать».

Обсудим, наконец, последний вопрос. Мы уже знаем из предыдущей главы, что ЭВМ может работать только по программе, которая написана на ее машинном языке. Алгоритмические же языки являются для ЭВМ «непонятными», и программа, составленная на одном из таких языков, для своего исполнения должна быть сначала оттранслирована, т. е. переведена на ее машинный язык.

Наиболее просто проблема трансляции решается для автокода, который очень близок к машинному языку. Чтобы понять принцип трансляции с автокода, предположим сначала, что эту работу должен выполнить человек. В начале своей работы он составляет две таблицы. Одна устанавливает взаимно однозначное соответствие между символьными обозначениями операций и их цифровыми кодами, вторая — между встречающимися в программе именами и адресами ячеек памяти, выделенными для хранения соответствующей информации: команд, снабженных метками, и численных значений переменных. Вооружившись этими таблицами, человек должен просмотреть всю программу команду за командой и заменить символьные обозначения автокода цифровыми. В результате он получит программу, эквивалентную исходной, но записанную на машинном языке.

Описанная процедура трансляции с автокода представляет собой хорошо алгоритмизованный процесс, так что ее может осуществить не только человек, но и ЭВМ. Для этого нужно написать на машинном языке специальную программу, воспроизводящую действия человека-переводчика. Такие программы были созданы и получили название *ассемблеров* (сборщиков). В связи с этим автокод часто называют *языком ассемблера*.

Для каждого типа ЭВМ ассемблер разрабатывается один раз, затем его можно многократно использовать для трансляции различных автокодных программ. Опишем коротко принцип работы ассемблера. Обычно эта программа вместе с таблицей соответствия двух форм записи операций в виде названий и в виде цифровых кодов хранится во внешней памяти ЭВМ. Вводя в оперативную память машины автокодную программу, предназначенную для трансляции, про-

граммист вызывает из внешней памяти ассемблер с таблицей и передает ему управление. Ассемблер проводит анализ программы, распределяет память, закрепляя за переменными и командами ячейки, в которых они будут храниться, формирует таблицу имен и сопоставленных им адресов ячеек памяти. Если таблица соответствия двух форм записи операций является универсальной, то вторая таблица для каждой программы индивидуальна. Она составляется непосредственно в процессе трансляции. Закончив эту работу, ассемблер, подобно человеку, просматривает транслируемую программу команду за командой и заменяет с помощью таблиц соответствия символьные обозначения операций цифровыми кодами, имена — адресами соответствующих ячеек. Преобразованные команды записываются друг за другом в отведенное им место в памяти машины. После окончания процесса трансляции версию программы на машинном языке можно вывести на печать или перфорацию для дальнейшего использования, а можно сразу передать на нее управление и приступить к расчетам.

Разработка трансляторов с проблемно-ориентированных языков является несравненно более сложным делом. Такие языки по своей структуре далеки от машинного и трансляция программы не сводится к простой замене одних обозначений другими с помощью таблиц взаимно однозначного соответствия. Обычно над созданием транслятора с алгоритмического языка высокого уровня работает группа системных программистов в течение нескольких месяцев и даже лет. Чем богаче по своим возможностям язык, тем сложнее разработка транслятора для него. После создания транслятора схема работы с программами на соответствующем языке оказывается аналогичной описанной выше схеме работы с программами на автокоде.

Трансляция программ требует расхода машинного времени. Однако эти потери окупаются преимуществами, которые дает программирование на алгоритмических языках по сравнению с программированием на языке машины.

Транслятор берет на себя ряд элементов работы программиста, например распределение памяти. Он упрощает отладку программы, проводит подробный анализ ее текста и при наличии ошибок в правилах употребления языка указывает точное место каждой из них. Однако возможны ошибки, которые правилам употребления языка не противоречат. Предположим, например, что в некоторой формуле по невнимательности программиста вместо знака «минус»

поставлен знак «плюс». Такие ошибки транслятор обнаружить не может. Их должен находить сам программист с помощью пробных, тестовых расчетов.

§ 3. Стандартные подпрограммы. Библиотеки. Пакеты прикладных программ

В самом начале эры ЭВМ, еще при работе на вычислительной технике первого поколения, было отмечено, что при составлении программ для решения различных задач могут многократно повторяться фрагменты, требующие выполнения одних и тех же действий. Например, при расчете по формуле

$$\frac{e^{\sin(ax)} - e^{\sin(bx)}}{e^{\cos(ax)} + e^{\cos(bx)}}$$

нужно два раза вычислить синус и косинус и четыре раза экспоненту. В программировании разработан простой и удобный способ, который позволяет не выписывать по нескольку раз набор команд для выполнения повторяющихся операций. Этот способ заключается в оформлении соответствующих фрагментов программы в виде отдельных подпрограмм.

Подпрограммой называется последовательность команд, выполняющая какое-либо законченное действие и оформленная таким образом, чтобы ее можно было использовать в нескольких местах одной программы или в разных программах. Подпрограмма составляется один раз, а используется многократно. Когда возникает необходимость обратиться к подпрограмме, в соответствующем месте основной программы записывается несколько команд. Они формируют нужную информацию и передают управление подпрограмме. Выполнив необходимые вычисления, подпрограмма возвращает управление основной программе.

Многие подпрограммы создаются непосредственно пользователями ЭВМ в процессе разработки своих программ. Однако такие действия, как, например, перевод чисел из одной системы счисления в другую, вычисление элементарных функций, встречаются практически во всех программах. Составлять каждому пользователю для них свои подпрограммы неразумно. Чтобы избежать ненужного дублирования, стали разрабатывать для различных ЭВМ наборы подпрограмм «широкого спроса». Их создавали и тщательно отлаживали группы высококвалифицированных специа-

листов. Предоставленные в распоряжение всех пользователей, такие подпрограммы получили название *стандартных подпрограмм.* (Иногда их называют стандартными программами.) Создание стандартных подпрограмм и их широкое использование было исторически первым шагом на пути автоматизации программирования (алгоритмические языки появились позже). Этот шаг оказался очень важным. Стандартные подпрограммы не только упрощают и ускоряют программирование, они позволяют «обучать» машину, развивать ее возможности.

Стандартные подпрограммы расширяют набор операций, выполняемых ЭВМ. Предположим, что в некоторой машине вычисление $\sqrt{x}$ предусмотрено ее конструкцией и выполняется по одной из команд, а у другой машины такой команды нет, но для вычисления $\sqrt{x}$ создана стандартная подпрограмма. С точки зрения человека, столкнувшегося с необходимостью использовать операцию извлечения корня в своей программе, никакого принципиального различия между этими машинами нет. Выполнение операции *аппаратно* (с помощью одной из команд, предусмотренных конструкцией ЭВМ) или *программно* (с помощью стандартной программы) для пользователя выглядит практически одинаково. Это очень важно. Число операций, выполняемых аппаратно, сравнительно невелико и для уже построенных ЭВМ его изменить нельзя. При проектировании новых ЭВМ расширить систему команд можно, но это требует усложнения аппаратуры. Увеличение же числа операций, выполняемых программно, не связано ни с какими принципиальными ограничениями и осуществляется сравнительно просто: нужно разработать стандартные подпрограммы, соответствующие новым потребностям.

Идея развития возможностей ЭВМ программным путем оказалась очень плодотворной. Наряду со стандартными подпрограммами в помощь прикладным программистам начинают заготовлять программы для решения многих типичных задач. В качестве примера можно указать задачи линейной алгебры, численного интегрирования, решения дифференциальных уравнений, специальных прикладных задач. Совокупность таких программ, созданных для какой-либо ЭВМ, объединенных и организованных определенным образом, называют *библиотекой.*

Стандартные программы нужны практически всем пользователям. Те из них, которые пользуются особенно широ-

ким спросом, хранятся обычно в оперативной памяти машины. Библиотеки имеют более узкую направленность, рассчитаны на специальные группы пользователей и хранятся, как правило, во внешней памяти.

Число программ, входящих в состав достаточно развитой библиотеки, может измеряться сотнями. Чтобы упростить работу с ними, создают специальную обслуживающую программу, называемую *библиотекарем*. Библиотекарь «знает», какие программы входят в библиотеку и где они хранятся во внешней памяти. Когда программисту нужно использовать какую-нибудь библиотечную программу, он сообщает ее название библиотекарю. Тот находит заказанную программу во внешней памяти, переписывает ее оттуда в оперативную память и включает в соответствующее место основной программы. Тем самым с программиста снимаются многие хлопоты по использованию библиотеки. Сотрудники вычислительного центра, сопровождающие библиотеку, «информируют» библиотекаря о всех изменениях в ее составе: о включении новых программ и исключении программ, ставших ненужными. Благодаря этому по таким вопросам библиотекарь может давать пользователям справки. Большие развитые библиотеки для удобства работы часто делят на тематические разделы.

Программы первых библиотек, как и стандартные подпрограммы, были написаны на машинном языке. В дальнейшем широкое распространение получили библиотеки на алгоритмических языках, особенно на фортране.

Библиотеки сыграли важную роль в автоматизации программирования. Однако на определенном этапе развития вычислительной математики и вычислительной техники возникла необходимость в новой форме организации работы по использованию накопленного программного «богатства». Обычно библиотеки создавались постепенно, в течение достаточно длительного промежутка времени. Входящие в их состав программы разрабатывались разными авторами, нередко в разных организациях. Единого подхода к их созданию и накоплению не было. В результате программы, даже принадлежащие к одному тематическому разделу библиотеки, часто оказывались разрозненными, не рассчитанными на совместное использование. Поэтому они могли не охватывать всего круга типичных задач из соответствующей области. Недостаточно были развиты средства сборки основной программы из библиотечных программ, поэтому пользователь библиотеки был вынужден сам дописывать многие

недостающие фрагменты и тратить силы на стыковку программ. Любые изменения, даже если они были связаны с заменой только одной библиотечной программы, требовали осуществлять процесс сборки и стыковки заново. Пока программы были сравнительно небольшими, с этими недостатками мирились. Однако переход в конце 60-х — начале 70-х годов к большим задачам (размеры программ для их решения стали достигать сотен тысяч команд) потребовал усовершенствования формы работы. Так появились пакеты прикладных программ.

Пакет — это комплекс взаимосвязанных программ, предназначенный для решения любой задачи из какой-нибудь конкретной области. Пакеты имеют, как правило, ярко выраженный *проблемно-ориентированный характер*. Например, существуют пакеты для расчета крыла самолета, атомного реактора, а также пакеты «служебного» типа для решения определенного класса математических задач. В известном смысле пакет представляет собой дальнейшее развитие идеи тематического раздела библиотеки. Основное различие между ними состоит в том, что программы библиотеки обычно используются независимо, а программы пакета рассчитаны на совместное применение в различных комбинациях друг с другом.

Создание пакета требует комплексного подхода, совместного труда прикладных и системных программистов. Приступая к его разработке, создатели прежде всего проводят анализ всех задач из соответствующей области и выделяют класс подзадач, к которым они сводятся. Для каждой подзадачи разрабатывается алгоритм ее решения и пишется соответствующая программа, которую обычно называют *модулем*. Совокупность модулей составляет *функциональное наполнение пакета*.

Наряду с функциональным наполнением создается *системное наполнение*, которое состоит из служебных программ, предназначенных обеспечить пользователям пакета максимальные удобства. Эти программы управляют работой всего пакета, анализируют и выполняют задания пользователей, автоматизируют процесс сборки их программ из модулей, позволяют производить пополнение пакета, вносить изменения в его модули. Важно, чтобы язык общения пользователей с пакетом прикладных программ был прост, гибок, удобен.

В простейших пакетах, чтобы решить какую-нибудь конкретную задачу, пользователь пишет на алгоритмиче-

ском языке (например на фортране) программу, которая представляет собой последовательность обращений к нужным модулям пакета. При этом программист сам определяет, какие модули ему понадобятся и в каком порядке, сам организует передачу данных от одного модуля к другому. В целом это похоже на работу с библиотекой, однако пакет предоставляет более удобные средства для выполнения всех работ по сборке и стыковке модулей, а сам набор модулей более богат и взаимосвязан.

Наряду с этим существуют пакеты другого типа. При их создании заранее фиксируется набор допустимых задач из некоторой области. Для каждой задачи из этого набора составляется последовательность модулей, выполнение которых обеспечивает получение решения. При работе с таким пакетом пользователь может не знать ни его состава, ни принципов внутренней организации. Он должен только на языке общения с пакетом указать название своей задачи (принадлежащей к допустимым) и дать необходимую информацию: исходные данные, требуемую точность, форму выдачи результатов и т. д. Все остальное программа, управляющая работой пакета, сделает сама.

Мы описали два крайних случая. Достоинства каждого из них являются одновременно и его недостатками. Поэтому многие пакеты создаются как промежуточные варианты описанных выше. С одной стороны, для определенного набора типичных задач в них заранее фиксирована последовательность модулей, осуществляющая решение. С другой стороны, пользователь имеет возможность сам составлять нужные комбинации модулей, чтобы решать «несерийные» задачи. В этом случае он должен составить программу, вызывающую и стыкующую нужные модули. Такой промежуточный тип пакета является наиболее распространенным.

В настоящее время проводится разработка пакетов нового типа. Получив от пользователя описание постановки задачи, такой пакет должен провести ее анализ и осуществить сборку модулей, обеспечивающих решение. Важно подчеркнуть, что нужная последовательность модулей не задается заранее. Система управления пакетом находит ее сама по описанию задачи. Для этого используются методы «искусственного интеллекта». Разработка таких «умных» пакетов находится в начальной стадии. Пока с их помощью удается формировать без вмешательства человека нужную последовательность модулей только для сравнительно простых задач.

§ 4. Операционные системы

В предыдущих параграфах было рассказано о тех частях математического обеспечения ЭВМ, которые облегчают создание прикладных программ и использование в них разработанных ранее программ. Теперь мы познакомимся с *операционными системами*, управляющими процессом выполнения этого комплекса программ.

Прообразом современных операционных систем явились *мониторные системы*, которые предоставляли пользователям средства, упрощающие работу на ЭВМ в однопрограммном режиме. Выше уже отмечалось, что трансляторы и библиотеки хранятся во внешней памяти машин. На первых порах, чтобы воспользоваться этими средствами в своей работе, программист должен был самостоятельно описать все действия, связанные с их вызовом в оперативную память ЭВМ. С появлением мониторных систем для выполнения таких операций стало достаточно подложить к своей программе несколько управляющих перфокарт. В них на специальном языке, который называют *языком управления заданиями*, указывается, какой транслятор, какие библиотеки нужны для работы, и, в случае необходимости, сообщается некоторая дополнительная информация о задаче. На основании этих данных мониторная система осуществляет вызов нужного транслятора, выборку заказанных библиотечных программ и включение их в основную программу, исполнение программы, обработку ошибок, которые могут при этом выявиться, и выдачу соответствующей информации о них программисту.

Совокупность программы, исходных данных для нее и управляющих перфокарт называется *заданием*. Ниже в качестве примера приводится схема задания для мониторной системы Дубна, разработанная в Объединенном институте ядерных исследований для ЭВМ БЭСМ-6:

```
*NAME ИВАНОВ
*LIBRARY:2
*ALGOL
 программа на алголе-60
*EXECUTE
 исходные данные
*END FILE
```

Строки, начинающиеся со звездочек,— это управляющие карты. В первой из них приведена фамилия автора, которая

будет напечатана в начале выдачи по данной программе. Вторая карта указывает, что в задании предполагается использовать программы библиотеки № 2 (library — библиотека). Названия нужных библиотечных программ задаются не в управляющей карте, а непосредственно в тексте основной программы. Третья карта представляет собой заказ на вызов транслятора с языка алгол-60. После управляющих карт, которые мы только что обсудили, следует текст основной программы на алголе-60, а затем еще одна управляющая карта EXECUTE (выполнить). Она сообщает мониторной системе, что после трансляции программы нужно сразу приступить к ее выполнению, а не записывать, скажем, на хранение во внешнюю память машины. Исходные данные, необходимые для проведения расчетов, следуют за картой EXECUTE. Последняя управляющая карта фиксирует конец задания.

Отметим, что карты LIBRARY и ALGOL дают разную информацию о задании и являются поэтому независимыми. Их можно располагать в любом порядке. Общее число управляющих карт в задании не фиксировано. Например, если в основной программе не предусмотрено использование библиотеки, то карта LIBRARY не нужна, и наоборот, возможны задания, на формирование которых потребуется больше управляющих карт, чем в приведенном примере. Все управляющие карты, кроме индивидуальной карты-паспорта с фамилией автора задания, являются стандартными. В вычислительных центрах их готовят заранее, так что набрать из них нужный комплект не составляет никакого труда.

В предыдущей главе уже отмечалось, что однопрограммный режим использования высокопроизводительных ЭВМ неэффективен. Слишком много приходится простаивать центральному процессору, пока работают очень медленные по сравнению с ним устройства ввода и вывода. Поэтому для третьего поколения ЭВМ были разработаны операционные системы, которые обеспечивали *пакетную обработку заданий в мультипрограммном режиме.*

В однопрограммном режиме, пока выполняется одно задание, другие задания в машину не вводятся. Они ждут своей очереди. В мультипрограммном режиме работа организована иначе. Сначала формируется *пакет заданий*: собираются вместе несколько заданий и записываются во внешнюю память ЭВМ (обычно на магнитный диск). После этого управление передается операционной системе. Она

считывает первую программу в оперативную память и поручает центральному процессору начать ее выполнение. Центральный процессор работает с программой до тех пор, пока в ней не возникает необходимость обмена (ввода или вывода). В этот момент операционная система включает устройство ввода-вывода, которое может работать автономно, удаляет первую программу во внешнюю память, а оттуда считывает вторую программу. Теперь центральный процессор начинает работать со второй программой, а автономное устройство ввода-вывода выполняет обмен, предусмотренный первой программой. Когда во втором задании возникает необходимость обмена, операционная система поступает точно так же: включается соответствующее устройство ввода-вывода, а программа убирается во внешнюю память. На ее же место в оперативную память либо возвращается первая программа, если предусмотренный в ней обмен уже закончился, либо считывается третья программа.

Итак, при пакетной обработке заданий в мультипрограммном режиме машина выполняет сразу несколько программ. Центральный процессор работает с ними последовательно: пока по одной программе проводятся вычисления, другие программы либо обслуживаются внешними устройствами, либо ждут своей очереди. Если программы пакета не перегружены обменом и внешние устройства с ним справляются, то центральный процессор все время активно работает. В результате эффективность использования ЭВМ по сравнению с однопрограммным режимом повышается.

Однако в области развития и использования вычислительной техники ситуация меняется очень быстро. То, что еще вчера казалось верхом совершенства, сегодня обнаруживает недостатки. Так получилось с пакетной обработкой заданий. С появлением выносных терминалов начались настойчивые поиски путей улучшения контактов человека с машиной. Пакетный режим оказался для этой цели неудобным. Чтобы понять причину, представим, что несколько пользователей ввели с выносных терминалов свои программы. Машина сформировала из них пакет и приступила к его обработке. Мы уже знаем, что центральный процессор в каждый момент времени ведет расчеты только по одной программе, а остальные программы должны ждать, пока не возникнет необходимость обмена. Программы бывают разные. Может случиться, что в выполняемой программе

выдача результатов предусмотрена только после продолжительных расчетов. В обычном пакетном режиме, когда машина работает без прямых контактов с пользователями, это не страшно. В диалоговом режиме, когда пользователи сидят за терминалами и ждут своей очереди, длительное использование процессора одной задачей неприемлемо.

Выход из описанной ситуации довольно прост: каждой задаче нужно давать выполняться не дольше определенного времени (порядка долей секунды). Если за это время она не закончится или не наступит обмен, то операционная система прекращает с ней работу и отправляет всю информацию о ней во внешнюю память, вызвав оттуда следующую задачу. С ней также операционная система разрешает центральному процессору работать лишь определенное время, после чего вызывает третью задачу и т. д. Такая форма мультипрограммной работы называется *режимом разделения времени* центрального процессора между несколькими программами.

В настоящее время операционные системы, обеспечивающие разделение времени, являются наиболее широко распространенными. Они эффективно загружают центральный процессор и позволяют при этом сразу нескольким пользователям работать в диалоге с ЭВМ. Пользователи сидят за выносными терминалами и, общаясь с ЭВМ, выполняют свои программы. Благодаря частому переключению центрального процессора с одной программы на другую, у каждого из них создается иллюзия, что машина находится в его личном распоряжении. После очередной порции расчетов и выдач они имеют возможность подумать, проанализировать результаты, выданные на экран дисплея, внести изменения в программу, задать новый вариант данных и т. д. Может получиться так, что одновременно несколько пользователей задумаются над своими задачами. Чтобы машина не простаивала в подобных ситуациях, в нее вводят небольшой пакет программ, который не требует контакта с их авторами. Такой пакет называется фоновым. Он используется в качестве резерва в случае, если центральный процессор окажется не полностью загруженным заданиями, которые выполняются в режиме диалога.

Важное значение для применения ЭВМ имеет еще одна разновидность операционных систем — *системы реального времени*. Они используются в тех случаях, когда ЭВМ управляет каким-нибудь устройством или процессом (маневром

спутника, работой атомного реактора, производственно-технологическим процессом и т. д.). Работа в режиме реального времени означает, что время, отпущенное машине на переработку поступающей информации и формирование управляющих сигналов, строго ограничено. Если машина в него не уложилась, то ее работа в лучшем случае окажется ненужной. Что может произойти в худшем случае, зависит от характера управляемого объекта. Конкретная продолжительность допустимого временно́го интервала для разных объектов различна. Она может меняться в очень широких пределах от долей секунды до нескольких часов.

Иногда машина, работающая в режиме реального времени, оказывается загруженной объектом управления не слишком сильно, тогда она может параллельно выполнять другие задания. Однако при таком двойном использовании ЭВМ программа управления имеет высокий приоритет. Она постоянно хранится в оперативной памяти машины, независимо от того, работает в данный момент или нет. Как только управляемый объект начинает подавать сигналы — запросы, машина немедленно прерывает все остальные расчеты, убирает соответствующие программы во внешнюю память и полностью переключается на управление.

После этого краткого описания основных типов операционных систем познакомимся с их структурой и принципами работы. Операционная система — это нижний уровень математического обеспечения ЭВМ. Она представляет собой совокупность программ на машинном языке. В процессе разработки элементов операционной системы может использоваться автокод и даже алгоритмические языки высокого уровня. Однако после того, как ее разработка закончена, все программы должны быть оттранслированы, т. е. переведены на язык ЭВМ. Наиболее важная часть операционной системы постоянно находится в оперативной памяти машины, остальные программы хранятся во внешней памяти на специальном магнитном диске.

Основными частями операционной системы являются *планировщик заданий*, *супервизор* и *программа, управляющая вводом-выводом*.

Назначение планировщика состоит в том, чтобы проанализировать задания, введенные в машину, и поставить их в очередь на выполнение. Чтобы планировщик мог справиться со своей задачей, в каждом задании содержится информация

о требуемых для его выполнения вычислительных ресурсах (времени расчета, объеме оперативной памяти, используемых магнитных лентах и дисках и т. д.). Если какой-нибудь из параметров в задании не указан, то планировщик принимает для него некоторое стандартное, заранее оговоренное значение. Получив такую информацию о задании, планировщик прежде всего проверяет, можно ли его в принципе выполнить, т. е. имеются ли в настоящее время у машины необходимые ресурсы. Только в случае положительного ответа на этот вопрос задание ставится на очередь. В противном случае планировщик посылает соответствующее сообщение операторам ЭВМ, но задание пока на очередь не ставит. Пусть, например, запрошенная в задании лента в данный момент на магнитофонах не стоит. Планировщик сообщит о появлении такого задания и попросит операторов поставить нужную ленту.

Принципы образования очереди для отобранных, «выполнимых» заданий могут быть самые разные. Часто используется простое житейское правило: «раньше пришел, раньше обслуживаешься». Возможен такой принцип: «чем меньше вычислительных ресурсов требует задание, тем ближе к началу очереди оно помещается». Используются и более сложные критерии. После того как очередь образована, операционная система начинает работу с заданиями, стоящими в начале очереди: выделяет место в оперативной памяти, устанавливает связь с заказанной библиотекой, вызывает нужный транслятор, осуществляет вычисления по оттранслированной программе. По мере изменения ситуации (при окончании какого-нибудь задания, поступлении нового задания, установлении затребованной ленты и т. д.) снова включается в работу планировщик и, руководствуясь установленными принципами, перестраивает очередь.

Вторая часть операционной системы — супервизор *) управляет работой центрального процессора и других устройств ЭВМ, осуществляет надзор за выполняющимися программами. Супервизор «знает», какая программа в настоящий момент вычисляется (занимает процессор), какие программы ведут обмен, а какие обмен закончили и готовы к продолжению расчетов. Супервизор учитывает поступающие от программ запросы на обмен, контролирует оконча-

*) Супервизор (по-английски supervisor) — надсмотрщик, контролер, всевидящий.

ние работы устройств ввода-вывода, принимает решение, какой программе следует предоставить центральный процессор и на какое время.

В своей работе супервизор широко пользуется такой аппаратной особенностью современных ЭВМ, как прерывание. *Прерывание* — это сигнал, который посылается центральному процессору другими устройствами машины при наступлении события, требующего внимания супервизора. Например, такой сигнал посылают: арифметическое устройство, когда в выполняющейся программе обнаруживается ошибка; устройство вывода, закончив печатать заданную порцию информации; операторский терминал, от которого поступило какое-то сообщение. При появлении сигнала прерывания работа над выполняющейся программой приостанавливается и управление передается на фиксированную команду супервизора. Все это делается аппаратно, поскольку до данного момента супервизор не работал (процессор был занят выполнением очередной программы). Получив сигнал, супервизор включается в работу. Он определяет причину прерывания и решает, что делать дальше. Пусть, например, сигнал прерывания поступил от устройства вывода: оно известило об окончании работы. В этом случае супервизор отмечает у себя, что устройство свободно, а программа, закончив обмен, готова к продолжению расчетов, и возобновляет работу по прерванной программе. Если причина прерывания — окончание времени, выделенного программе, по которой проводятся вычисления, то супервизор переводит ее в состояние ожидания и переключает процессор на следующую программу, возобновляя ее вычисления с прерванного ранее места. Реакция супервизора на ошибку — это прекращение вычислений по данной программе, печать сообщения об ошибке, возобновление работы следующей программы. Так с помощью прерывания супервизор управляет работой ЭВМ в мультипрограммном режиме.

Важной частью операционной системы являются программы управления вводом-выводом. Они избавляют пользователей от сложной работы по составлению заданий для внешних устройств, требующей хорошего знания технических возможностей каждого из них. Операционная система предоставляет готовый набор программ для организации обмена. Имея его в своем распоряжении, пользователь лишь указывает, что нужно ввести или вывести, на какое устройство и в какой форме.

Среди других элементов операционных систем отметим библиотекаря и программы вызова трансляторов, программы связи с операторами ЭВМ, программы диагностики неисправностей ЭВМ. Следует также упомянуть программы сбора статистики, которые фиксируют, какие задания выполнялись, сколько машинного времени, бумаги и других ресурсов на каждое из них было израсходовано. Этот краткий обзор показывает, каким огромным и сложным хозяйством являются операционные системы современных ЭВМ и какую важную роль в их эксплуатации они играют.

ГЛАВА 5

СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Эта глава посвящена системам линейных алгебраических уравнений

$$\begin{aligned} &a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=f_1,\\ &a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n=f_2,\\ &\ldots\ldots\ldots\ldots\\ &a_{m1}x_1+a_{m2}x_2+\ldots+a_{mn}x_n=f_m. \end{aligned} \tag{1}$$

Такую систему удобно записать в компактной векторно-матричной форме

$$A\boldsymbol{x}=\boldsymbol{f}. \tag{1'}$$

Здесь A — матрица коэффициентов, содержащая m строк и n столбцов:

$$A=\begin{Vmatrix} a_{11} & a_{12} & \ldots & a_{1n}\\ a_{21} & a_{22} & \ldots & a_{2n}\\ \ldots & \ldots & \ldots & \ldots\\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{Vmatrix}, \tag{2}$$

$\boldsymbol{f}$ — заданный m-компонентный вектор, $\mathbf{x}$ — искомый n-компонентный вектор:

$$\boldsymbol{f}=\begin{bmatrix} f_1\\ f_2\\ \ldots\\ f_m \end{bmatrix}, \qquad \boldsymbol{x}=\begin{bmatrix} x_1\\ x_2\\ \ldots\\ x_n \end{bmatrix}. \tag{3}$$

Из линейной алгебры известно, что совокупность n-компонентных векторов вида (3) образует n-мерное *линейное пространство*: для них определены операции сложения и умножения на число, удовлетворяющие 8 аксиомам линейного пространства.

По аналогии с векторами в обычном трехмерном пространстве введем понятие ны или *евклидовой нормы*

n-компонентных векторов:

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{4}$$

Опираясь на евклидову норму (4), определим расстояние между векторами:

$$\rho(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{5}$$

Пространство n-компонентных векторов с расстоянием (5) называется *n-мерным евклидовым пространством* и обозначается символом $\mathbf{R}^n$. Понятия евклидовой нормы вектора (4) и расстояния между векторами (5) понадобятся нам при обсуждении методов решения системы (1).

§ 1. Формулы Крамера

Основное внимание в этой главе будет уделено системам, у которых число уравнений равно числу неизвестных ($m=n$):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= f_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= f_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= f_n, \end{aligned} \tag{6}$$

или в векторно-матричной форме:

$$Ax = f, \tag{6'}$$

где A — квадратная матрица, содержащая n строк и n столбцов, f и x — заданный и искомый n-компонентные векторы.

Предположим, что система (6) невырождена, т. е. ее определитель отличен от нуля: $\Delta = \det A \neq 0$. В этом случае она имеет единственное решение, которое может быть записано с помощью *формул Крамера*:

$$x_i = \Delta_i/\Delta, \qquad i = 1, 2, \ldots, n. \tag{7}$$

Здесь $\Delta_i = \det A_i$, матрица A_i получается из матрицы A заменой i-го столбца столбцом правых частей f:

$$A_i = \begin{Vmatrix} a_{11} & \ldots & a_{1i-1} & f_1 & a_{1i+1} & \ldots & a_{1n} \\ a_{21} & \ldots & a_{2i-1} & f_2 & a_{2i+1} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & \ldots & a_{ni-1} & f_n & a_{ni+1} & \ldots & a_{nn} \end{Vmatrix}.$$

Хотя формулы Крамера дают решение системы (6) в

явном виде, с вычислительной точки зрения они неэффективны. Расчет по ним требует слишком большого числа операций и оказывается очень чувствительным к ошибкам округления. Поэтому на практике для решения линейных алгебраических систем используются другие методы, которые по принципам организации вычислений можно разделить на два класса: прямые и итерационные.

Прямыми методами называются такие методы, которые позволяют за конечное число действий получить точное решение системы. Слова «точное решение» нужно понимать условно как характеристику алгоритма, а не реального вычислительного процесса. Алгоритмы, лежащие в основе прямых методов, дают точное решение, если все величины в системе заданы и все вычисления проводятся абсолютно точно, без ошибок округления. К прямым методам относится, например, метод последовательного исключения неизвестных Гаусса, с которым мы познакомимся в следующем параграфе.

Итерационные методы, как показывает название, основаны на построении итерационной последовательности, сходящейся к искомому решению. Вычисляя определенное число итераций и обрывая процесс, можно получить приближенное решение системы с любой наперед заданной точностью ε.

Выбор в каждом отдельном случае метода решения алгебраической системы (6) определяется многими факторами: особенностями матрицы A, порядком системы n, характеристиками ЭВМ — ее архитектурой и операционной системой, быстродействием, памятью, программным обеспечением. Важную роль играет также характер решаемой задачи. Например, метод, который проигрывает при решении одной системы, может иметь преимущества в случае, когда нужно решать несколько систем с одной матрицей A и разными правыми частями $\boldsymbol{f}$.

§ 2. Метод Гаусса

Метод последовательного исключения неизвестных Гаусса является одним из наиболее универсальных и эффективных методов решения линейных алгебраических систем. Как указывалось в предыдущем параграфе, он относится к числу прямых методов. Процесс решения по методу Гаусса состоит из двух этапов. На первом этапе (прямой ход) система (6) приводится к треугольному виду. На втором

этапе (обратный ход) идет последовательное определение неизвестных из этой треугольной системы.

Перейдем к подробному описанию метода Гаусса. Будем считать, что коэффициент a_{11}, который называют *ведущим элементом первого шага*, отличен от нуля. (В случае $a_{11}=0$ поменяем местами уравнения с номерами 1 и i, где $a_{i1}\neq 0$. Поскольку система предполагается невырожденной, такой номер i заведомо найдется.) Составим отношения

$$m_{21}=-\frac{a_{21}}{a_{11}}, \quad m_{31}=-\frac{a_{31}}{a_{11}}, \quad \ldots, \quad m_{n1}=-\frac{a_{n1}}{a_{11}}$$

и прибавим к i-му уравнению системы ($i=2, 3, \ldots, n$) 1-е уравнние,в умноженное на m_{i1}. Проделав это, мы исключим неизеестное x_1 из всех уравнений, начиная со 2-го. Преобразованная система примет вид

$$\begin{aligned} a_{11}x_1+a_{12}x_2+a_{13}x_3+\ldots+a_{1n}x_n&=f_1,\\ a_{22}^{(1)}x_2+a_{23}^{(1)}x_3+\ldots+a_{2n}^{(1)}x_n&=f_2^{(1)},\\ \ldots\ldots\ldots\ldots\ldots\ldots&\\ a_{n2}^{(1)}x_2+a_{n3}^{(1)}x_3+\ldots+a_{nn}^{(1)}x_n&=f_n^{(1)}. \end{aligned} \tag{8}$$

Здесь $a_{ij}^{(1)}$, $f_i^{(1)}$ ($i, j=2, 3, \ldots, n$) — новые значения коэффициентов и правых частей, которые получаются после выполнения первого шага прямого хода метода Гаусса. Основным результатом этого шага является система $n-1$-го порядка с неизвестными $x_2, x_3, \ldots, x_n$, образованная последними $n-1$ уравнениями системы (8). Дальнейшим преобразованиям будем подвергать именно эту систему.

Переходя к выполнению второго шага, предположим, что элемент $a_{22}^{(1)}$, который называют *ведущим элементом второго шага*, отличен от нуля. (В противном случае нужно провести соответствующую перестановку уравнений.) Составим отношения

$$m_{32}=-\frac{a_{32}^{(1)}}{a_{22}^{(1)}}, \quad m_{42}=-\frac{a_{42}^{(1)}}{a_{22}^{(1)}}, \quad \ldots, \quad m_{n2}=-\frac{a_{n2}^{(1)}}{a_{22}^{(1)}}$$

и прибавим к i-му уравнению системы (8) ($i=3, 4, \ldots, n$) 2-е уравнение, умноженное на m_{i2}. В результате получим систему

$$\begin{aligned} a_{11}x_1+a_{12}x_2+a_{13}x_3+\ldots+a_{1n}x_n&=f_1,\\ a_{22}^{(1)}x_2+a_{23}^{(1)}x_3+\ldots+a_{2n}^{(1)}x_n&=f_2^{(1)},\\ a_{33}^{(2)}x_3+\ldots+a_{3n}^{(2)}x_n&=f_3^{(2)},\\ \ldots\ldots\ldots\ldots\ldots\ldots&\\ a_{n3}^{(2)}x_3+\ldots+a_{nn}^{(2)}x_n&=f_n^{(2)}. \end{aligned} \tag{9}$$

После n—1-го шага мы придем к треугольной системе

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= f_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n &= f_2^{(1)}, \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n &= f_3^{(2)}, \\ \cdots\cdots\cdots\cdots & \\ a_{nn}^{(n-1)}x_n &= f_n^{(n-1)}. \end{aligned} \tag{10}$$

Матрицу коэффициентов этой системы обозначим через R:

$$R = \begin{Vmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ & a_{22}^{(1)} & a_{23}^{(1)} & \ldots & a_{2n}^{(1)} \\ & & a_{33}^{(2)} & \ldots & a_{3n}^{(2)} \\ & 0 & & \ldots & \\ & & & & a_{nn}^{(n-1)} \end{Vmatrix} \tag{11}$$

Приведение исходной системы (6) к треугольному виду (10) завершает первый этап построения решения методом Гаусса.

Второй этап — обратный ход — заключается в решении треугольной системы (10). Проводится оно следующим образом. Из последнего уравнения определяем x_n. По найденному значению x_n из n—1-го уравнения определяем x_{n-1}. Затем по значениям x_{n-1} и x_n из n—2-го уравнения определяем x_{n-2} и т. д. Последовательное вычисление неизвестных продолжается до тех пор, пока мы не определим из 1-го уравнения x_1. На этом процесс построения решения системы (6) с помощью решения эквивалентной ей системы (10) заканчивается.

Отметим, что прямой ход метода Гаусса требует $n^3/3 + O(n^2)$ операций сложения, столько же операций умножения и $n^2/2 + O(n)$ операций деления, обратный ход требует $n^2/2 + O(n)$ операций сложения, столько же операций умножения и n операций деления. Таким образом, приведение системы (6) к треугольному виду (10) на порядок n сложнее, чем построение решения этой системы.

В качестве примера, иллюстрирующего метод Гаусса, решим следующую систему трех уравнений с тремя неизвестными:

$$\begin{aligned} 1{,}2357x_1 + 2{,}1742x_2 - 5{,}4834x_3 &= -2{,}0735, \\ 6{,}0696x_1 - 6{,}2163x_2 - 4{,}6921x_3 &= -4{,}8388, \\ 3{,}4873x_1 + 6{,}1365x_2 - 4{,}7483x_3 &= 4{,}8755. \end{aligned} \tag{12}$$

Все результаты расчетов условимся представлять в виде чисел с плавающей запятой с пятью значащими цифрами.

В рассматриваемом случае

$$m_{21} = -4{,}9119, \quad m_{31} = -2{,}8221.$$

В результате после первого шага гауссова исключения получим систему

$$\begin{aligned} 1{,}2357x_1 + 2{,}1742x_2 - 5{,}4834x_3 &= -2{,}0735, \\ -16{,}895x_2 + 22{,}242x_3 &= 5{,}3462, \\ 0{,}0007x_2 + 10{,}727x_3 &= 10{,}727. \end{aligned} \tag{13}$$

Чтобы сделать второй шаг, вычислим

$$m_{32} = \frac{0{,}0007}{16{,}895} = -0{,}41432 \cdot 10^{-4}$$

и приведем систему к треугольному виду:

$$\begin{aligned} 1{,}2357x_1 + 2{,}1742x_2 - 5{,}4834x_3 &= -2{,}0735, \\ -16{,}895x_2 + 22{,}242x_3 &= 5{,}3462, \\ 10{,}728x_3 &= 10{,}727. \end{aligned} \tag{14}$$

Решение треугольной системы (14) дает следующие значения неизвестных:

$$x_1 = 0{,}99968, \quad x_2 = 0{,}99994, \quad x_3 = 0{,}99991. \tag{15}$$

Этот результат хорошо согласуется с точным решением системы (12):

$$x_1 = x_2 = x_3 = 1. \tag{16}$$

В заключение отметим, что сведение системы (6) к системе (10) на первом этапе метода Гаусса связано с преобразованием матрицы A к треугольному виду. Это может быть использовано для вычисления ее определителя. Прямой ход метода Гаусса основан на том, что многократно выполняется операция сложения одной из строк матрицы с другой строкой, взятой с некоторым множителем. Известно, что такая операция не меняет определителя. Иногда приходится также переставлять уравнения, чтобы перед началом очередного шага элемент $a_{ii}^{(i-1)}$ был отличен от нуля. Перестановка строк матрицы меняет знак ее определителя на противоположный. С учетом этих замечаний получим

$$\det A = (-1)^k \det R = (-1)^k a_{11} a_{22}^{(1)} \ldots a_{nn}^{(n-1)},$$

где k — число перестановок строк в процессе редукции матрицы A к треугольной матрице R (11).

В качестве примера подсчитаем определитель матрицы коэффициентов системы (12). При преобразовании этой системы к треугольному виду (14) мы не делали перестановок уравнений. Следовательно, в данном случае $\det A = \det R = -223{,}97$.

Число операций, которое требуется для подсчета определителя по методу Гаусса, не идет ни в какое сравнение с вычислением определителя в лоб как суммы $n!$ его членов. Например, при прямом вычислении сравнительно небольшого определителя 20-го порядка требуется выполнить $19 \cdot 20! \approx 4{,}5 \cdot 10^{19}$ умножений. На машине, которая делает миллион умножений в секунду, такой расчет продолжался бы $1{,}4 \cdot 10^{6}$ лет.

§ 3. Уменьшение ошибок округления

Обратимся еще раз к системе (12). Поменяем в ней местами 2-е и 3-е уравнения и снова решим ее методом Гаусса, используя, как и в § 2, запись результатов вычислений в виде чисел с плавающей запятой с 5 значащими цифрами. После первого шага мы получим систему (13), в которой 2-е и 3-е уравнения поменялись местами. Второй шаг будет выглядеть иначе. В рассматриваемом случае

$$m_{32} = \frac{16{,}895}{0{,}0007} = 24\,136.$$

В результате после исключения из 3-го уравнения неизвестной x_2 получим систему треугольного вида:

$$\begin{aligned} 1{,}2357x_1 + 2{,}1742x_2 - 5{,}4834x_3 &= -2{,}0735, \\ 0{,}0007x_2 + 10{,}727x_3 &= 10{,}727, \\ 258\,930x_3 &= 258\,910. \end{aligned} \tag{17}$$

Определяя из нее последовательно x_3, x_2, x_1, найдем решение:

$$x_1 = 2{,}9021; \quad x_2 = 1{,}4286; \quad x_3 = 0{,}9992. \tag{18}$$

Если бы мы могли осуществить идеальный вычислительный процесс без ошибок округления, то независимо от порядка уравнений в обоих случаях получили бы один и тот же ответ — точное решение (16). Реальный вычислительный процесс дает приближенное решение, которое, как показывает приведенный пример, может существенно зависеть от порядка уравнений. В одном случае получается вполне разумный ответ (15), в другом ответ (18) находится с очень

большой ошибкой: у переменной x_1 неверно определена первая значащая цифра.

Анализируя вычисления, легко найти причину такого различия. Неприятности во втором случае связаны с малым значением ведущего элемента второго шага $a_{22}^{(1)}=0{,}0007$ и, соответственно, с большим значением коэффициента $m_{32}=24\,136$. Умножение на этот коэффициент приводит к резкому увеличению значения элементов 3-го уравнения системы (17), что, в конечном счете, и порождает большую вычислительную ошибку.

Чтобы избежать нежелательного роста значения элементов матрицы во время прямого хода и тем самым предотвратить большую потерю точности, обычно метод Гаусса применяют в сочетании с какой-нибудь схемой выбора ведущего элемента на каждом шаге. Мы рассмотрим простейшую и наиболее употребительную схему выбора ведущего элемента по столбцу, которая реализуется во многих стандартных программах метода Гаусса. Она заключается в следующем. Перед началом i-го шага исключения сравнивают между собой элементы i-го столбца матрицы, построенной на предыдущем шаге: $a_{ii}^{(i-1)}, a_{i+1\,i}^{(i-1)}, \ldots, a_{ni}^{(i-1)}$. Пусть наибольший по модулю элемент стоит в k_i-й строке ($i\leqslant k_i\leqslant n$) тогда уравнения с номерами i и k_i меняют местами, переме; щая тем самым элемент $a_{k_i\,i}^{(i-1)}$ с максимальным модулем на ведущую позицию. После этого выполняют i-й шаг, исключая переменную x_i в уравнениях с номерами $i+1, i+2, \ldots, n$. При этом отношения m_{ji} ($i+1\leqslant j\leqslant n$), которые вычисляются на каждом шаге процесса исключения, оказываются ограниченными по модулю единицей. В результате наибольший по модулю элемент матрицы коэффициентов за один шаг может возрасти не более чем в 2 раза. Этого, как правило, достаточно, чтобы предотвратить заметный рост значений элементов матрицы коэффициентов на протяжении всего процесса исключения. Существуют более сложные и более надежные схемы выбора ведущих элементов, на описании которых мы останавливаться не будем.

Любая специальная схема выбора ведущих элементов усложняет программу и требует дополнительных вычислительных ресурсов. Однако в конечном счете это себя оправдывает благодаря повышению точности решения задачи.

§ 4. Итерационные методы

Как отмечалось в § 1, итерационные методы основаны на построении итерационной последовательности $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, ..., сходящейся к искомому решению системы $\boldsymbol{x}$. Каждый такой метод характеризуется своей итерационной формулой, позволяющей вычислять очередное приближение $\boldsymbol{y}_{k+1}$ по ранее найденным. В простейшем случае при вычислении $\boldsymbol{y}_{k+1}$ используется только одна предыдущая итерация $\boldsymbol{y}_k$. Такие методы называются *одношаговыми* или *двухслойными*. Итерационную формулу для одношаговых методов принято записывать в стандартной *канонической форме*:

$$B_{k+1}\frac{\boldsymbol{y}_{k+1}-\boldsymbol{y}_k}{\tau_{k+1}}+A\boldsymbol{y}_k=\boldsymbol{f}. \tag{19}$$

Здесь τ_{k+1} — *итерационные параметры* ($\tau_{k+1}>0$), B_{k+1} — вспомогательные невырожденные матрицы. Если τ и B не зависят от индекса $k+1$, т. е. формула (19) при любых k имеет один и тот же вид, то итерационный метод называется *стационарным*. Стационарные методы наиболее просты с точки зрения организации вычислительного процесса. Однако *нестационарные методы* имеют другие преимущества: они предоставляют дополнительные «степени свободы», связанные с выбором последовательностей τ_{k+1}, B_{k+1}. Это может быть использовано для ускорения сходимости итераций $\boldsymbol{y}_k$ к решению системы $\boldsymbol{x}$.

Определение очередного приближения $\boldsymbol{y}_{k+1}$ с помощью итерационной формулы (19) требует решения системы уравнений

$$B_{k+1}\boldsymbol{y}_{k+1}=\boldsymbol{F}_{k+1}, \tag{20}$$

где

$$\boldsymbol{F}_{k+1}=(B_{k+1}-\tau_{k+1}A)\boldsymbol{y}_k+\tau_{k+1}\boldsymbol{f}.$$

Такую процедуру приходится выполнять на каждом шаге. Поэтому применение метода может быть оправдано лишь при условии, что определение отдельных членов итерационной последовательности $\boldsymbol{y}_{k+1}$ требует существенно меньшего объема вычислений, чем прямое решение исходной системы (6). Это накладывает определенные ограничения на выбор матриц B_{k+1}.

Самая простая схема вычисления членов итерационной последовательности получается в случае, когда в качестве матрицы B_{k+1} берут единичную матрицу: $B_{k+1}=E$; тогда формула (19) позволяет выразить в явном ви е очередной

член последовательности y_{k+1} через предыдущий y_k:

$$y_{k+1} = y_k - \tau_{k+1} A y_k + \tau_{k+1} f. \tag{21}$$

Итерационные методы, основанные на такой рекуррентной формуле, называются *явными*.

Среди *неявных* методов ($B_{k+1} \neq E$) наибольшее распространение получили методы, в которых матрицы B_{k+1} выбираются треугольными. В этом случае нахождение очередной итерации y_{k+1} сводится к последовательному определению компонент y_{k+1} из треугольной системы (20) подобно тому, как это делается при обратном ходе в методе Гаусса.

Применение какого-нибудь итерационного метода предполагает сходимость последовательности y_k к решению системы x:

$$\lim_{k \to \infty} y_k = x. \tag{22}$$

Предельное равенство (22) означает, что

$$\rho(y_k, x) = \|y_k - x\| = \sqrt{(y_1^{(k)} - x_1)^2 + (y_2^{(k)} - x_2)^2 + \ldots + (y_n^{(k)} - x_n)^2} \to 0. \tag{23}$$

Из (23) видно, что необходимым и достаточным условием сходимости последовательности векторов y_k к вектору x является покомпонентная сходимость

$$\lim_{k \to \infty} y_i^{(k)} = x_i, \qquad i = 1, 2, \ldots, n. \tag{24}$$

Отличие итерации y_k от решения системы x называется *погрешностью*: $z_k = y_k - x$. Записывая y_k в виде $y_k = x + z_k$ и подставляя в (19), получим итерационную формулу для погрешности:

$$B_{k+1} \frac{z_{k+1} - z_k}{\tau_{k+1}} + A z_k = 0. \tag{25}$$

В отличие от (19) она не содержит правой части системы f, т. е. является однородной. Требование сходимости (22) означает, что погрешность z_k должна стремиться к нулю:

$$\lim_{k \to \infty} z_k = 0. \tag{26}$$

Достаточные условия сходимости каждого итерационного метода формулируются в виде условий, которым должны удовлетворять матрицы A, B_{k+1} и итерационные параметры τ_{k+1}. Некоторые из них, особенно относящиеся к оптимальному выбору итерационных параметров, трудно

проверить. В результате при проведении расчетов итерационные параметры часто приходится подбирать эмпирически.

После этой общей характеристики итерационных методов решения систем линейных алгебраических уравнений рассмотрим в качестве примера два конкретных метода.

1. Метод простой итерации. Метод простой итерации представляет собой явный стационарный метод: $B_{k+1}=E$, $\tau_{k+1}=\tau=\text{const}$. Для него рекуррентная формула (21) принимает вид

$$y_{k+1}=y_k-\tau Ay_k+\tau f. \tag{27}$$

Чтобы сформулировать достаточное условие сходимости метода простой итерации, нам понадобятся два свойства матриц.

Матрица A называется *симметричной*, если ее элементы удовлетворяют условию $a_{ij}=a_{ji}$ $(i, j=1, 2, \ldots, n)$.

Отсюда видно, что у симметричной матрицы любые два элемента, расположенные симметрично относительно главной диагонали, должны быть равны между собой. Рассмотрим в качестве примера матрицы

$$A_1=\begin{Vmatrix}-1 & 2\\ 2 & 5\end{Vmatrix},\quad A_2=\begin{Vmatrix}1 & 1\\ 3 & 5\end{Vmatrix}. \tag{28}$$

Первая из них является симметричной ($a_{12}=a_{21}=2$), вторая — нет ($a_{12}=1$, $a_{21}=3$).

Матрица A называется *положительно определенной*, если для любого ненулевого вектора x

$$\varphi(x)=\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}x_ix_j>0. \tag{29}$$

Рассмотрим еще раз в качестве примера матрицы (28). Матрица A_1 не является положительно определенной. Действительно, для этой матрицы

$$\varphi_1(x)=\varphi_1(x_1, x_2)=-x_1^2+4x_1x_2+5x_2^2.$$

Если мы возьмем вектор x с компонентами $x_1=1$, $x_2=0$, то получим $\varphi_1=-1<0$, что противоречит (29). Наоборот, матрица A_2 является положительно определенной. Для нее

$$\varphi_2(x)=\varphi_2(x_1, x_2)=x_1^2+4x_1x_2+5x_2^2=(x_1+2x_2)^2+x_2^2>0$$

при $\|x\|^2=x_1^2+x_2^2$.

Предположим, что дана система линейных алгебраических уравнений с симметричной и положительно определен-

ной матрицей A. Можно доказать, что в этом случае множество значений параметра τ, для которых метод простой итерации сходится при любом выборе начального приближения y_0, образует интервал $0<\tau<\tau_0$. Правая граница интервала определяется видом матрицы A: $\tau_0=\tau_0(A)$. Итерационный процесс сходится со скоростью геометрической прогрессии, причем знаменатель прогрессии q является функцией τ: $q=q(\tau)$. Существует оптимальное значение $\tau^* \in (0, \tau_0)$, при котором $q(\tau)$ достигает своего наименьшего значения, и, следовательно, скорость сходимости оказывается наибольшей. По мере удаления τ от τ^* и приближения к левой или правой границе интервала $(0, \tau_0)$ сходимость ухудшается: $q(\tau)$ возрастает и стремится снизу к единице.

2. Метод Зейделя. Будем считать, что все диагональные элементы матрицы A отличны от нуля:

$$a_{ii} \neq 0, \qquad i = 1, 2, \ldots, n, \tag{30}$$

и представим ее в виде суммы трех матриц:

$$A = A^- + D + A^+.$$

Здесь D — диагональная матрица, A^- и A^+ — нижняя и верхняя треугольные матрицы:

$$D = \begin{Vmatrix} a_{11} & & & 0 \\ & a_{22} & & \\ & & \ddots & \\ 0 & & & a_{nn} \end{Vmatrix},$$

$$A^- = \begin{Vmatrix} 0 & & & 0 \\ a_{21} & 0 & & \\ a_{31} & a_{32} & 0 & \\ \cdots & & & \\ a_{n1} & \ldots & a_{n,\,n-1} & 0 \end{Vmatrix},$$

$$A^+ = \begin{Vmatrix} 0 & a_{12} & a_{13} & \ldots & a_{1n} \\ & 0 & a_{23} & \ldots & a_{2n} \\ & & & \cdots & \\ & & 0 & & a_{n-1,\,n} \\ & 0 & & & 0 \end{Vmatrix}.$$

Метод Зейделя является неявным стационарным методом,

для которого каноническая форма итерационного соотношения записывается следующим образом:

$$(A^- + D)(y_{k+1} - y_k) + Ay_k = f, \tag{31}$$

т. е. в данном случае $\tau = 1$,

$$B = A^- + D = \begin{Vmatrix} a_{11} & & 0 \\ a_{21} & a_{22} & \\ \cdots & \cdots & \\ a_{n1} & a_{n2} \ \ldots & a_{nn} \end{Vmatrix}. \tag{32}$$

Рекуррентная формула (31), связывающая y_{k+1} и y_k, допускает другую форму записи:

$$(A^- + D)\, y_{k+1} + A^+ y_k = f. \tag{33}$$

Отсюда видно, что расчет очередной итерации y_{k+1} в методе Зейделя сводится к решению системы линейных алгебраических уравнений с невырожденной треугольной матрицей (32).

Можно доказать сходимость метода Зейделя для систем с симметричными положительно определенными матрицами и для систем, матрицы которых наделены свойством диагонального преобладания. Оба указанных условия являются достаточными.

Матрица A называется матрицей *с диагональным преобладанием*, если ее элементы удовлетворяют неравенствам

$$\sum_{j=1}^{i-1} |a_{ij}| + \sum_{j=i+1}^{n} |a_{ij}| < |a_{ii}|, \qquad i = 1, 2, \ldots, n, \tag{34}$$

т. е. в каждой строке матрицы модуль диагонального элемента превышает сумму модулей всех остальных элементов.

В отличие от метода простой итерации, свободных параметров, позволяющих регулировать скорость сходимости, в методе Зейделя нет. Такие параметры появляются в более сложных обобщениях метода Зейделя, на которых мы останавливаться не будем.

§ 5. Обусловленность матриц

Матрица системы A и ее правая часть f во многих случаях задаются приближенно. Причины погрешности могут быть самые разные — от ошибок округления при вводе чисел в машину до ошибок измерения, если система связана с обработкой экспериментальных данных. Ошибки вносит

также, как мы видели, вычислительный процесс. Естественно встает вопрос, как все это влияет на точность получаемого решения. Чтобы на него ответить, нам нужно познакомиться с особой характеристикой матриц, которую называют обусловленностью.

Матрица A ставит в соответствие каждому вектору $\boldsymbol{x}$ вектор $\boldsymbol{y}=A\boldsymbol{x}$. Будем считать, что вектор $\boldsymbol{x}$ принадлежит единичной сфере:

$$\|\boldsymbol{x}\|=\sqrt{\sum_{i=1}^{n} x_i^2}=1, \tag{35}$$

и подсчитаем длину вектора $\boldsymbol{y}=A\boldsymbol{x}$:

$$\|\boldsymbol{y}\|=\sqrt{\sum_{i=1}^{n} y_i^2}=\sqrt{\sum_{i=1}^{n}\left(\sum_{j=1}^{n} a_{ij}x_j\right)^2}=g(\boldsymbol{x}). \tag{36}$$

Функция $g(\boldsymbol{x})$ непрерывна на ограниченном замкнутом множестве (35); следовательно, она достигает на нем своего наибольшего и наименьшего значений. Введем в рассмотрение соответствующие величины, являющиеся важными характеристиками матрицы A:

$$M=\sup_{\|\boldsymbol{x}\|=1}\|A\boldsymbol{x}\|=\sup_{\|\boldsymbol{x}\|=1} g(\boldsymbol{x}), \tag{37}$$

$$m=\inf_{\|\boldsymbol{x}\|=1}\|A\boldsymbol{x}\|=\inf_{\|\boldsymbol{x}\|=1} g(\boldsymbol{x}). \tag{38}$$

Отметим, что $m\geqslant 0$, причем $m=0$ тогда и только тогда, когда матрица A вырождена и однородное уравнение $A\boldsymbol{x}=0$ имеет нетривиальные решения. Отношение $\mu=M/m\geqslant 1$ называется *обусловленностью матрицы* A.

Используя величины (37), (38), для любого единичного вектора $\boldsymbol{x}$ можно написать двойное неравенство

$$m\leqslant\|A\boldsymbol{x}\|\leqslant M, \quad \|\boldsymbol{x}\|=1. \tag{39}$$

Для произвольного вектора $\boldsymbol{x}$ в силу линейности отображения $\boldsymbol{y}=A\boldsymbol{x}$ неравенство (39) принимает вид

$$m\|\boldsymbol{x}\|\leqslant\|A\boldsymbol{x}\|\leqslant M\|\boldsymbol{x}\|. \tag{40}$$

Исследуем с помощью этого неравенства устойчивость решения системы (6) по отношению к малым изменениям входящих в нее величин. Рассмотрим простейший случай, когда матрица системы предполагается заданной точно, а правая часть — приближенно, так что вместо исходной системы (6) мы имеем систему с возмущенной правой

частью $\tilde{f}$. Обозначим ее решение через $\tilde{x}$:

$$A\tilde{x}=\tilde{f}. \tag{41}$$

Представим векторы $\tilde{f}$ и $\tilde{x}$ в виде

$$\tilde{f}=f+\Delta f, \qquad \tilde{x}=x+\Delta x$$

и вычтем (6) из (41). В результате получим систему уравнений, связывающую возмущения:

$$A\Delta x=\Delta f. \tag{42}$$

Применяя к системам (6) и (42) неравенства (40) и используя в случае (6) оценку сверху, а в случае (42) снизу, получим

$$\|f\|\leqslant M\|x\|, \quad \|\Delta f\|\geqslant m\|\Delta x\|.$$

Таким образом,

$$\frac{\|\Delta x\|}{\|x\|}\leqslant \mu\frac{\|\Delta f\|}{\|f\|}. \tag{43}$$

Величина $\|\Delta f\|/\|f\|$ характеризует относительное возмущение правой части, а величина $\|\Delta x\|/\|x\|$ — относительную ошибку в решении, вызванную этим возмущением. Обусловленность матрицы μ играет в (43) роль множителя, определяющего максимально возможное увеличение ошибки. Проведенные оценки показывают следующее. Если μ близко к единице, то система хорошо обусловлена: для такой системы относительная ошибка в определении решения сравнима с относительной погрешностью, с которой задается правая часть. По мере увеличения μ чувствительность решения к погрешности в правой части возрастает — система становится плохо обусловленной.

Рассмотрим в качестве примера, иллюстрирующего введенные понятия, следующую систему двух уравнений с двумя неизвестными:

$$\begin{aligned} x_1+0\cdot x_2&=1,\\ x_1+0{,}01x_2&=1. \end{aligned} \tag{44}$$

Это невырожденная система ($\det A=0{,}01\neq 0$), решение которой имеет вид

$$x_1=1, \qquad x_2=0. \tag{45}$$

Изменим немного в (44) правую часть, заменив во втором уравнении 1 на 1,01. В результате получим новую

систему

$$\begin{aligned} x_1 + 0 \cdot x_2 &= 1, \\ x_1 + 0{,}01 x_2 &= 1{,}01, \end{aligned} \tag{46}$$

имеющую решение

$$x_1 = 1, \quad x_2 = 1. \tag{47}$$

Мы видим, что небольшое возмущение правой части системы (44) привело к существенному изменению решения. Это говорит о плохой обусловленности матрицы

$$A = \begin{Vmatrix} 1 & 0 \\ 1 & 0{,}01 \end{Vmatrix}. \tag{48}$$

Действительно, подсчитав для нее параметр μ, мы убедимся в том, что он большой и разница в решениях систем (44) и (46) согласуется с оценкой (43).

Рассмотрим функцию (36), которая в данном случае имеет вид

$$\begin{aligned} g(x_1, x_2) &= \{x_1^2 + (x_1 + 0{,}01 x_2)^2\}^{1/2} = \\ &= \{2x_1^2 + 0{,}02 x_1 x_2 + 0{,}0001 x_2^2\}^{1/2}, \end{aligned} \tag{49}$$

и найдем ее наибольшее и наименьшее значения на единичной окружности

$$x_1^2 + x_2^2 = 1. \tag{50}$$

Чтобы решить такую задачу на условный экстремум, положим $x_1 = \cos\varphi$, $x_2 = \sin\varphi$. С переходом к переменной φ условие (50) выполняется автоматически, и задача сводится к задаче на безусловный экстремум для функции одной переменной

$$\begin{aligned} h(\varphi) &= g(\cos\varphi, \sin\varphi) = \\ &= \{2\cos^2\varphi + 0{,}02\cos\varphi\sin\varphi + 0{,}0001\sin^2\varphi\}^{1/2} = \\ &= \{1{,}00005 + a\cos(2(\varphi - \varphi_0))\}^{1/2}, \end{aligned} \tag{51}$$

где

$$a = \sqrt{1 + (0{,}00005)^2} = \sqrt{1 + 0{,}25 \cdot 10^{-8}} \approx 1,$$

$$\varphi_0 = \frac{1}{2}\arccos\frac{0{,}99995}{a}.$$

Отсюда видно, что наибольшее и наименьшее значения функция $h(\varphi)$ достигает соответственно при $\varphi = \varphi_0$ и $\varphi = \varphi_0 + \pi/2$:

$$M = h(\varphi_0) = \sqrt{1{,}00005 + a} \approx \sqrt{2},$$

$$m = h(\varphi_0 + \pi/2) = \sqrt{1{,}00005 - a} \approx 0{,}01/\sqrt{2}.$$

В результате получаем следующее значение обусловленности матрицы (48):

$$\mu = \frac{M}{m} = \frac{\sqrt{1{,}00005 + a}}{\sqrt{1{,}00005 - a}} \approx 200. \qquad (52)$$

Такое большое значение μ делает понятным существенное различие решений систем (44) и (46), несмотря на близость их правых частей. В данном случае

$$f = \{1, 1\}, \quad \|f\| = \sqrt{2}, \quad x = \{1, 0\}, \quad \|x\| = 1,$$
$$\Delta f = \{0, 0{,}01\}, \quad \|\Delta f\| = 0{,}01, \quad \Delta x = \{0, 1\}, \quad \|\Delta x\| = 1$$

и в соответствии с неравенством (43) получаем

$$\frac{\|\Delta x\|}{\|x\|} = 1 < \mu \frac{\|\Delta f\|}{\|f\|} \approx \sqrt{2}.$$

Разобранный пример построен специально, однако он наглядно показывает, какие трудности могут возникнуть при решении реальных систем с приближенно заданной правой частью, если они плохо обусловлены.

§ 6. Нормальные решения приближенных систем линейных алгебраических уравнений

Многие прикладные задачи приводят к системам линейных алгебраических уравнений, в которых приближенно задается не только правая часть, но и матрица. Если исходная система является невырожденной, то она остается невырожденной и при достаточно малых возмущениях. Относительная ошибка в решении, которую порождает возмущение, определяется в этом случае его относительной величиной и обусловленностью матрицы системы. Когда возмущение стремится к нулю, ошибка в решении также стремится к нулю, т. е. решение невырожденной системы устойчиво по отношению к малым возмущениям.

Картина существенно усложняется в случае, когда рассматриваемая система является либо переопределенной (число уравнений превышает число неизвестных, $m > n$), либо вырожденной ($\det A = 0$). Дело в том, что теорема Кронекера формулирует условия разрешимости таких систем в виде точных равенств, которые для приближенно заданных величин оказываются лишенными смысла.

Поясним характер возникающих трудностей на простом примере.

Рассмотрим систему

$$\begin{aligned} x_1 + 7x_2 &= 5, \\ \sqrt{2}\,x_1 + \sqrt{98}\,x_2 &= \sqrt{50}. \end{aligned} \tag{53}$$

Она является вырожденной (второе уравнение получается из первого умножением на $\sqrt{2}$) и имеет бесчисленное множество решений. В прикладных задачах, приводящих к линейным алгебраическим системам, такая неопределенность, как правило, бывает неприемлема. В этом случае в математическую постановку задачи, исходя из характера рассматриваемой проблемы, включают какое-нибудь дополнительное условие, обеспечивающее единственность решения. Можно, например, искать решение системы (53), ближайшее к некоторому заданному вектору $\boldsymbol{x}_0$ *(нормальное решение)*. Если принять, не ограничивая общности, $\boldsymbol{x}_0 = 0$, то задача сформулируется следующим образом: найти вектор $\bar{\boldsymbol{x}} = \{\bar{x}_1, \bar{x}_2\}$, удовлетворяющий системе (53) и имеющий наименьшую длину:

$$\|\bar{\boldsymbol{x}}\| \leqslant \|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2}. \tag{54}$$

Решение задачи (53), (54) легко найти из простых геометрических соображений. Оно лежит на пересечении прямых

$$\begin{aligned} x_1 + 7x_2 &= 5, \\ 7x_1 - x_2 &= 0. \end{aligned} \tag{55}$$

Первая прямая содержит все решения вырожденной системы (53), а вторая является перпендикуляром к первой, проходящим через начало координат. Согласно (55) нормальное решение системы (53) имеет вид

$$\bar{x}_1 = 0{,}1, \quad \bar{x}_2 = 0{,}7. \tag{56}$$

Предположим, что мы «не заметили» вырожденного характера системы (53) и решили ввести ее в ЭВМ. Для того чтобы записать иррациональные числа в машину, их нужно округлить. Округление «снимает» вырождение, в результате в памяти машины будет сформирована невырожденная система, имеющая единственное решение, которое можно подсчитать и вывести на печать.

Запишем, например, иррациональные числа с точностью до трех знаков. При таком округлении система (53) принимает вид

$$\begin{aligned} x_1 + 7x_2 &= 5, \\ 1{,}41x_1 + 9{,}90x_2 &= 7{,}07. \end{aligned} \tag{57}$$

Для нее

$$\Delta^{(3)}=0{,}03,\quad \Delta_{x_1}^{(3)}=0{,}01,\quad \Delta_{x_2}^{(3)}=0{,}02,$$
$$x_1^{(3)}=1/3,\quad x_2^{(3)}=2/3. \tag{58}$$

Здесь верхний индекс (3) указывает число знаков, которое было удержано при записи иррациональных чисел.

Попытаемся теперь повысить точность решения задачи, улучшая аппроксимацию иррациональных чисел. С этой целью построим последовательность аппроксимирующих систем

$$A_k\boldsymbol{x}=\boldsymbol{f}_k, \tag{59}$$

где k — число удерживаемых знаков при записи иррациональных чисел. Рассмотрим, как ведут себя решения этих систем по мере увеличения k. Выше уже было выписано решение $x^{(3)}$ (58). Приведем теперь для сравнения определители и решения систем при k=50, 200, 400, 600:

$$\Delta^{(50)}=-3\cdot 10^{-49},\quad x_1^{(50)}=9{,}666\ldots,\quad x_2^{(50)}=0{,}666\ldots,$$
$$\Delta^{(200)}=2\cdot 10^{-199},\quad x_1^{(200)}=-5{,}5,\quad x_2^{(200)}=1{,}5,$$
$$\Delta^{(400)}=1\cdot 10^{-399},\quad x_1^{(400)}=5,\quad x_2^{(400)}=0,$$
$$\Delta^{(600)}=-2\cdot 10^{-599},\quad x_1^{(600)}=8{,}5,\quad x_2^{(600)}=-0{,}5.$$

Мы видим, что определители, отличные от нуля за счет ошибок округления, имеют значения, которые составляют несколько единиц последнего удерживаемого разряда. Что касается решений, то они ведут себя крайне нерегулярно и с увеличением k ни к нормальному, ни к какому-нибудь другому фиксированному решению системы (53) не стремятся.

Округление иррациональных чисел и замену системы (53) системой (59) можно интерпретировать как внесение возмущения. При этом очень мало отличающиеся друг от друга возмущения приводят к существенно отличающимся решениям, т. е. простая замена решения системы (53) решением возмущенной системы делает задачу неустойчивой. Проведенный анализ показывает, что нуждается в уточнении сама постановка задачи о решении приближенно заданных систем линейных алгебраических уравнений. Эта проблема имеет фундаментальное значение для многих задач прикладной математики.

В самом начале этой главы были введены понятия нормы векторов и расстояния между векторами. Прямоугольную матрицу, имеющую m строк и n столбцов, можно

рассматривать как вектор размерности $m \times n$ и в соответствии с формулами (4), (5) ввести понятие нормы и расстояния для матриц:

$$\|A\| = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}, \tag{60}$$

$$\rho(A_1, A_2) = \|A_1 - A_2\|. \tag{61}$$

Подсчитаем в качестве примера расстояние между матрицами систем (53) и (57). Согласно (60), (61) получаем

$$\rho = \sqrt{(\sqrt{2} - 1{,}41)^2 + (\sqrt{98} - 9{,}90)^2} \approx 0{,}04.$$

Опираясь на понятия нормы и расстояния, уточним понятие приближенно заданной системы. Будем характеризовать такую систему какой-нибудь индивидуальной системой $(\tilde{A}, \tilde{f})$ и точностями β и δ, с которыми задаются матрица и правая часть. При этом любая система (A, f), удовлетворяющая условию

$$\|\tilde{A} - A\| \leqslant \beta, \quad \|\tilde{f} - f\| \leqslant \delta, \tag{62}$$

эквивалентна системе $(\tilde{A}, \tilde{f})$ по точности: с точки зрения доступной информации о системе они неразличимы. Поэтому, говоря о приближенно заданной системе, мы должны рассматривать не индивидуальную систему $(\tilde{A}, \tilde{f})$, а всю совокупность систем $\Sigma_{\beta, \delta}$, эквивалентных ей по точности.

Решение x любой системы $(A, f) \in \Sigma_{\beta, \delta}$ будем называть *допустимым решением* рассматриваемой задачи. Совокупность всех допустимых решений обозначим через $X_{\beta, \delta}$. В случае, когда множество $X_{\beta, \delta}$ — непустое, рассматриваемую приближенную систему назовем *совместной*. По смыслу задачи все допустимые решения совместной системы эквивалентны по точности. Если в $X_{\beta, \delta}$ входят вырожденные совместные системы, то множество допустимых решений $X_{\beta, \delta}$ оказывается неограниченным и содержит элементы, как угодно сильно различающиеся по норме. Это делает необходимым включить в постановку задачи дополнительное правило отбора допустимых решений, без которого ответ оказывается слишком неопределенным.

Во многих случаях в качестве дополнительного условия естественно потребовать, чтобы расстояние до некоторого заданного вектора x_0 было минимальным. Решение, удовлетворяющее такому условию, называют *нормальным решением* приближенной системы и обозначают через $\bar{x}$. Можно

считать, не ограничивая общности, $x_0=0$; тогда получим

$$\|\bar{x}\| \leqslant \|x\|, \qquad x \in X_{\beta,\delta}. \tag{63}$$

По определению, нормальное решение само входит в множество допустимых решений $X_{\beta,\delta}$. Это означает, что должны существовать такие матрица $\bar{A}$ и правая часть $\bar{f}$, что

$$\bar{A}\bar{x}=\bar{f}, \tag{64}$$

$$\|\tilde{A}-\bar{A}\| \leqslant \beta, \ \|\tilde{f}-\bar{f}\| \leqslant \delta. \tag{65}$$

Можно доказать, что у любой совместной приближенной системы существует единственное нормальное решение $\bar{x}$ и что задача (63) — (65), из которой оно определяется, является устойчивой. Разработаны алгоритмы численного решения этой задачи, однако на их описании мы останавливаться не будем. Приведем лишь в качестве примера решение системы (53). Будем рассматривать ее как приближенную систему, в которой матрица и правая часть задаются с точностями β и δ, тогда нормальные решения задачи при $\beta=\delta=10^{-1}$ и $\beta=\delta=10^{-4}$ имеют вид

$$\bar{x}_1=0{,}0968, \quad \bar{x}_2=0{,}6944, \quad \beta=\delta=10^{-1};$$

$$\bar{x}_1=0{,}09998, \quad \bar{x}_2=0{,}69998, \quad \beta=\delta=10^{-4}.$$

Мы видим, как по мере увеличения точности, с которой предполагается заданной система, эти решения приближаются к решению (56).

ГЛАВА 6

ЗАДАЧИ ОПТИМИЗАЦИИ

Эта глава посвящена математическим вопросам, связанным с проблемой оптимизации. Явно или неявно мы встречаемся с оптимизацией в любой сфере человеческой деятельности от сугубо личного до самого высокого общегосударственного уровня. Экономическое планирование, управление, распределение ограниченных ресурсов, анализ производственных процессов, проектирование сложных объектов всегда должно быть направлено на поиск наилучшего варианта с точки зрения намеченной цели. Это — важнейшее условие научно-технического прогресса.

При небывалом разнообразии задач оптимизации только математика может дать общие методы их решения. Однако для того, чтобы воспользоваться математическим аппаратом, необходимо сначала сформулировать интересующую нас проблему как математическую задачу, придав количественные оценки возможным вариантам, количественный смысл словам «лучше», «хуже».

Многие задачи оптимизации сводятся к отысканию наименьшего (или наибольшего) значения некоторой функции, которую принято называть *целевой функцией* или *критерием качества*. Постановка задачи и методы исследования существенно зависят от свойств целевой функции и той информации о ней, которая может считаться доступной в процессе решения, а также которая известна априори (до опыта, заранее; здесь — до начала решения задачи).

Наиболее просты, с математической точки зрения, случаи, когда целевая функция задается явной формулой и является при этом дифференцируемой функцией. В этом случае для исследования свойств функции, определения направлений ее возрастания и убывания, поиска точек локального экстремума может быть использована производная.

В последние десятилетия в условиях научно-технического прогресса круг задач оптимизации, поставленных практикой, резко расширился. Во многих из них целевая функция не задается формулой, ее значения могут получаться в результате сложных расчетов, браться из эксперимента и т. д. Такие задачи являются более сложными, потому что для них нельзя провести исследование целевой функции с помощью производной. Пришлось уточнять их математическую постановку и разрабатывать специальные методы решения, рассчитанные на широкое применение ЭВМ. Следует также иметь в виду, что сложность задачи существенно зависит от ее размерности, т. е. от числа аргументов целевой функции.

Первые три параграфа данной главы посвящены одномерным задачам оптимизации, в двух последних рассматриваются многомерные задачи. Выделение и подробный разбор одномерных задач имеет определенный смысл. Эти задачи наиболее просты, на них легче понять постановку вопроса, методы решения и возникающие трудности. В ряде случаев, хотя и очень редко, одномерные задачи имеют самостоятельный практический интерес. Однако самое главное заключается в том, что алгоритмы решения многомерных задач оптимизации часто сводятся к последовательному многократному решению одномерных задач и не могут быть поняты без умения решать такие задачи.

§ 1. Задача о наилучшей консервной банке

Перед вами поставили задачу: указать наилучший вариант консервной банки фиксированного объема V, имеющей обычную форму прямого кругового цилиндра. Получив такое задание, вы неизбежно должны спросить: «По какому признаку следует сравнивать банки между собой, какая банка считается наилучшей?» Иными словами, вы попросите указать цель оптимизации.

Рассмотрим два варианта этой задачи.

1. Наилучшая банка должна иметь наименьшую поверхность S. (На ее изготовление пойдет наименьшее количество жести.)

2. Наилучшая банка должна иметь наименьшую длину швов l. (Швы нужно сваривать, и мы хотим сделать эту работу минимальной.)

Для решения этой задачи запишем формулы для объема

банки, площади ее поверхности и длины швов:

$$V=\pi r^2/h, \quad S=2\pi r^2+2\pi rh, \quad l=4\pi r+h. \qquad (1)$$

Объем банки задан, это устанавливает связь между радиусом r и высотой h. Выразим высоту через радиус: $h=V/(\pi r^2)$ и подставим полученное выражение в формулы для поверхности и длины швов. В результате получим

$$S(r)=2\pi r^2+\frac{2V}{r}, \qquad 0<r<\infty, \qquad (2)$$

$$l(r)=4\pi r+\frac{V}{\pi r^2}, \qquad 0<r<\infty. \qquad (3)$$

Таким образом, с математической точки зрения, задача о наилучшей консервной банке сводится к определению такого значения r, при котором достигает своего наименьшего значения в одном случае функция $S(r)$, в другом — функция $l(r)$.

Рассмотрим первый вариант задачи. Вычислим производную функции $S(r)$:

$$S'(r)=4\pi r-\frac{2V}{r^2}=\frac{2}{r^2}(2\pi r^3-V) \qquad (4)$$

и исследуем ее знак. При $0<r<r_1=\sqrt[3]{V/(2\pi)}$ производная отрицательна и функция $S(r)$ убывает, при $r_1<r<\infty$ производная положительна и функция $S(r)$ возрастает. Следовательно, своего наименьшего значения эта функция достигает в точке $r=r_1$, в которой ее производная обращается в нуль. График функции $S(r)$, иллюстрирующий проведенный анализ, показан на рис. 14.

Итак, радиус и высота банки, наилучшие с точки зрения условия минимальности $S(r)$, определяются формулами

$$r_1=\sqrt[3]{\frac{V}{2\pi}}, \quad h_1=2r_1, \qquad (5)$$

при этом

$$S(r_1)=3\sqrt[3]{2\pi V^2}\leqslant S(r). \qquad (6)$$

Рассмотрим теперь задачу во второй постановке. Продифференцируем функцию $l(r)$:

$$l'(r)=4\pi-\frac{2V}{\pi r^3}=\frac{2}{\pi r^3}(2\pi^2r^3-V). \qquad (7)$$

Как и в предыдущем случае, при $0<r<r_2=\sqrt[3]{V/(2\pi^2)}$ производная отрицательна и функция $l(r)$ убывает, при

$r_2 < r < \infty$ производная положительна и функция $l(r)$ возрастает. Следовательно, своего наименьшего значения эта функция достигает в точке $r = r_2$, в которой ее производная обращается в нуль. График функции показан на рис. 15.

Рис. 14. График функции $S(r)$.

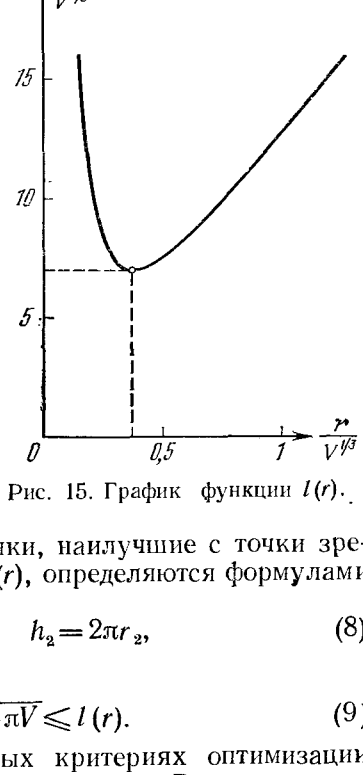

Рис. 15. График функции $l(r)$.

Итак, радиус и высота банки, наилучшие с точки зрения условия минимальности $l(r)$, определяются формулами

$$r_2 = \sqrt[3]{\frac{V}{2\pi^2}}, \quad h_2 = 2\pi r_2, \tag{8}$$

при этом

$$l(r_2) = 3\sqrt[3]{4\pi V} \leqslant l(r). \tag{9}$$

Мы видим, что при разных критериях оптимизации получаются существенно разные ответы. В первом случае (5) высота «наилучшей» банки равна ее диаметру, во втором (8) она в π раз больше диаметра.

§ 2. Одномерные задачи оптимизации

После обсуждения простого, но характерного примера рассмотрим общие вопросы постановки и методов решения одномерных задач оптимизации. С математической точки зрения такую задачу можно сформулировать следующим образом.

Найти наименьшее (или наибольшее) значение целевой функции $f(x)$, заданной на множестве X. Определить значение переменной $x \in X$, при котором она принимает свое экстремальное значение.

В математическом анализе при изучении свойств функций, непрерывных на отрезке, доказывается следующая теорема.

Теорема Вейерштрасса. *Всякая функция $f(x)$, непрерывная на отрезке $[a, b]$, принимает на этом отрезке свое наименьшее и наибольшее значения, т. е. на отрезке $[a, b]$ существуют такие точки x_1, x_2, что для любого $x \in [a, b]$ выполняются неравенства*

$$f(x_1) \leqslant f(x) \leqslant f(x_2). \tag{10}$$

Не исключается, в частности, возможность того, что наименьшее или наибольшее значение достигается сразу в нескольких точках. Вы легко можете убедиться в этом, рассмотрев в качестве примера функцию $y=\sin x$ на отрезке $[0, 4\pi]$. Она достигает своего наименьшего значения, равного — 1, сразу в двух точках: $x=3\pi/2$, $x=7\pi/2$. Наибольшее значение, равное 1, достигается тоже в двух точках: $x=\pi/2$, $x=5\pi/2$.

Теорема Вейерштрасса играет в данном случае роль теоремы существования: согласно этой теореме задача оптимизации, в которой целевая функция $f(x)$ задана и непрерывна на отрезке, всегда имеет решение.

Теперь нам предстоит обсудить методы решения задач оптимизации. В этом параграфе мы рассмотрим наиболее простой класс задач, аналогичных задаче о наилучшей консервной банке. При их исследовании мы будем предполагать, что целевая функция $f(x)$ дифференцируема на отрезке $[a, b]$ и имеется возможность найти явное выражение для ее производной $f'(x)$.

Точки, в которых производная обращается в нуль, называются *критическими* или *стационарными точками* функции $f(x)$. Если интерпретировать производную как скорость изменения функции, то в критических точках эта скорость равна нулю, изменение функции на мгновение «останавливается».

Функция $f(x)$ может достигать своего наименьшего (наибольшего) значения либо в одной из двух граничных точек отрезка $[a, b]$, либо в какой-нибудь его внутренней точке. В последнем случае такая точка обязательно должна быть критической, это необходимое условие экстремума. Учиты-

вая изложенное, можем сформулировать следующее п р а в и л о решения задачи оптимизации для рассматриваемого класса функций.

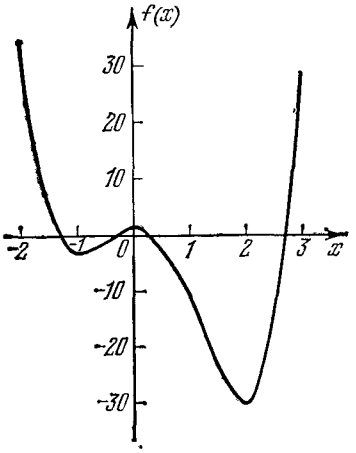

Рис. 16. График функции $f(x)=3x^4-4x^3-12x^2+2$.

Для того чтобы определить наименьшее и наибольшее значения дифференцируемой функции $f(x)$ на отрезке $[a, b]$, нужно найти все ее критические точки на данном отрезке, присоединить к ним граничные точки a и b и во всех этих точках сравнить значения функции. Наименьшее и наибольшее из них дадут наименьшее и наибольшее значения функции для всего отрезка.

Поскольку граничные точки a и b искать не нужно, то с технической точки зрения все сводится к определению критических точек, которые являются корнями уравнения

$$f'(x)=0. \tag{11}$$

Для иллюстрации изложенного правила решения задачи оптимизации рассмотрим на отрезке $[-2, 3]$ функцию

$$f(x)=3x^4-4x^3-12x^2+2. \tag{12}$$

Вычислим ее производную:

$$f'(x)=12x^3-12x^2-24x.$$

Таким образом, уравнение (11) для определения критических точек в данном случае принимает вид

$$x^3-x^2-2x=0. \tag{13}$$

Все корни этого уравнения: $x_1=-1$, $x_2=0$, $x_3=2$ принадлежат исходному отрезку. Добавляя к ним граничные точки: $a=-2$, $b=3$, вычислим соответствующие значения функции (12):

$$f(-2)=34,\quad f(-1)=-3,\quad f(0)=2,$$
$$f(2)=-30,\quad f(3)=29.$$

Из сравнения этих чисел следует, что наименьшее значение функции $f(x)$ достигается в одной из критических точек $x=2$, а наибольшее — в граничной точке $x=-2$, причем

$$f_{\min} = f(2) = -30,$$
$$f_{\max} = f(-2) = 34.$$

График функции (12), иллюстрирующий проведенное исследование, показан на рис. 16.

В простейших случаях, как в задаче о консервной банке или в рассмотренном примере (12), нули производной удается найти аналитически. На это в первую очередь и рассчитан данный метод, хотя не исключается возможность численного решения уравнения (11). Однако при этом важно найти все критические точки, иначе мы рискуем допустить ошибку, пропустив истинное наименьшее или наибольшее значение функции.

§ 3. Численное решение одномерных задач оптимизации

Рассмотрим следующий пример. Химический завод производит некоторое вещество. Выход интересующего нас продукта определяется температурой: $y=f(T)$. Температуру можно варьировать в определенных пределах: $T_1 \leqslant T \leqslant T_2$. Вид функции f заранее не известен, он зависит от используемого сырья. Получив очередную партию сырья, нужно найти температуру T, при которой наиболее выгодно вести производство, т. е. функция $f(T)$ достигает своего наибольшего значения.

С математической точки зрения мы имеем типичную одномерную задачу оптимизации, сформулированную в начале предыдущего параграфа. В то же время между этой задачей и задачей о консервной банке имеется существенное различие. В данном случае нет никакой формулы для целевой функции $f(T)$. Чтобы определить ее значение при некоторой температуре T, нужно провести опыт либо в лаборатории (если это возможно), либо прямо в производственных условиях. Совершенно ясно, что возможно лишь конечное число измерений и тем самым функция $f(T)$ будет известна нам в конечном числе точек. Значений ее производной мы вообще определить не можем. Строго говоря, мы даже не знаем, существует ли у нее производная, хотя опыт таких исследований и здравый смысл говорят, что функция $f(T)$, по-видимому, дифференцируема.

Нам остается добавить, что каждое измерение требует времени, а задерживать производство нельзя. Поэтому необходимо получить ответ на поставленный вопрос после небольшого числа измерений, т. е. по значениям функции $y=f(T)$ в нескольких точках.

Возможны также задачи оптимизации, в которых целевая функция $y=f(x)$ находится в результате численного решения некоторой математической задачи. Данный случай по своему характеру весьма близок к предыдущему: мы не имеем явной формулы для целевой функции, но можем определить ее значение для любого аргумента $x \in [a, b]$. Ясно, что при этом в ходе решения задачи нам окажется непосредственно доступной информация о целевой функции в конечном числе точек.

Итак, обсудим математические вопросы, связанные со следующей постановкой одномерной задачи оптимизации: определяя значения непрерывной функции $f(x)$ в некотором конечном числе точек отрезка $[a, b]$, нужно приближенно найти ее наименьшее (или наибольшее) значение на данном отрезке.

Возможны разные подходы к решению этой задачи. Рассмотрим сначала метод, идея которого наиболее проста и естественна.

1. Метод равномерного распределения точек по отрезку. Возьмем некоторое целое число n, вычислим шаг $h=(b-a)/n$ и определим значения функции $f(x)$ в точках $x_k=a+kh$ ($k=0, 1, \ldots, n$): $y_k=f(x_k)$. После этого найдем среди полученных чисел наименьшее:

$$m_n = \min(y_0, y_1, \ldots, y_n),$$
$$m_n \geqslant m = \min f(x), \quad x \in [a, b].$$

Число m_n можно приближенно принять за наименьшее значение функции $f(x)$ на отрезке $[a, b]$. Благодаря непрерывности функции $f(x)$ имеем

$$\lim_{n\to\infty} m_n = m = \min f(x),$$

т. е. с увеличением числа точек n ошибка, которую мы допускаем, принимая m_n за m, стремится к нулю.

Какое же n нужно взять, чтобы погрешность в определении наименьшего значения функции $\delta_n=m_n-m$ не превышала заданной точности ε, т. е. чтобы $\delta_n \leqslant \varepsilon$? Это — обычный вопрос, который всегда встает при приближенном решении математических задач. Вспомним, что для метода

вилки, благодаря двухсторонней оценке корня, мы легко получили условие достижения точности ε в виде неравенства (2.24).

Для данной задачи ситуация оказывается более сложной. Если нам известно только то, что функция $f(x)$ непрерывна на отрезке $[a, b]$, то ответить на поставленный

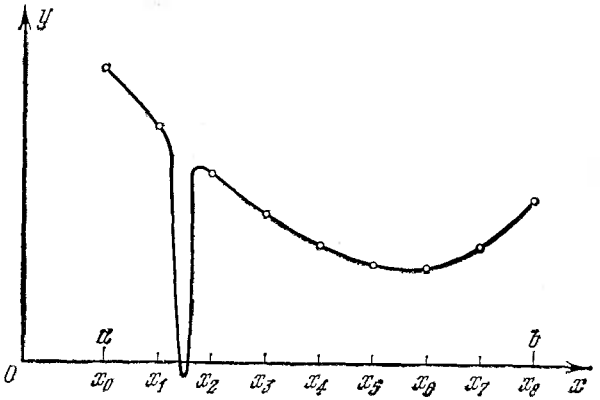

Рис. 17. Пример, иллюстрирующий трудности, которые могут возникнуть при приближенном определении наименьшего значения функции по ее значениям в нескольких точках.

вопрос нельзя. Эта трудность не связана с предложенным способом выбора точек x_k, она носит принципиальный характер. Какое бы n мы ни взяли и как бы ни выбирали точек на отрезке $[a, b]$, всегда можно указать такую непрерывную функцию, что для нее m_n будет отличаться от m больше чем на ε.

Справедливость этого утверждения иллюстрирует рис. 17. На нем приведен график некоторой непрерывной функции. Допустим, что, желая найти ее наименьшее значение, мы взяли $n=8$. Определяя значения функции $y_k=f(x_k)$ в точках x_k $(k=0, 1, \ldots, 8)$ (на рис. значения $y_k=f(x_k)$ отмечены кружочками), получим

$$m_8=\min(y_0, y_1, \ldots, y_8)=y_6=f(x_6).$$

Величину $m_8=y_6$ согласно описанному методу следует приближенно принять за наименьшее значение функции m.

Представьте, что у нас нет перед глазами рис. 17 (как и предполагает постановка задачи), а известны только 9 чисел y_k. По ним невозможно установить, что функция

$f(x)$ имеет между точками x_1 и x_2 узкий «язык», который опускается гораздо ниже $y_6 = m_8$. Из-за небольшого числа точек мы его пропустим. Если взять большее n, то данный «язык» обнаружится, но может оказаться незамеченным какой-нибудь другой еще более узкий «язык». При отсутствии дополнительной информации о свойствах функции $f(x)$, о том, насколько «резкими» могут быть ее изменения, сомнения останутся, какое бы большое число точек мы ни взяли.

Дать строго обоснованную оценку числа точек n, необходимого для решения задачи с точностью ε ($\delta_n \leqslant \varepsilon$), можно только, сужая класс рассматриваемых функций. Предположим, например, что функция $y = f(x)$ не просто непрерывна, а удовлетворяет условию Липшица с известной постоянной α, тогда легко получить следующее неравенство для нужного числа точек n в данном методе:

$$n \geqslant \frac{\alpha}{\varepsilon}(b-a). \tag{14}$$

Однако и этот результат малоэффективен при отсутствии явной формулы для функции $f(x)$. Он может быть использован, если имеется априорная информация о величине постоянной Липшица α. В противном случае предположение о справедливости для целевой функции условия Липшица и оценка константы α, входящей в (14), носит характер гипотезы, которую, как правило, невозможно проверить. (Подумайте сами, как это сделать в задаче о химическом производстве, если каждое значение целевой функции $f(T)$ получается в результате экспериментальных измерений, а условие Липшица, с точки зрения математического определения, нужно проверять для любой пары аргументов из рассматриваемой области.) Поэтому при решении вопроса о числе точек и точности важно максимально полно использовать всю дополнительную информацию о свойствах целевой функции, о степени ее гладкости, вытекающую из характера и особенностей задачи. Не последнюю роль играет и такой фактор, как опыт, интуиция исследователя.

2. Метод распределения точек по отрезку, учитывающий результаты вычисления целевой функции. Для метода, который был описан, характерно равномерное распределение «пробных» точек x_k по отрезку $[a, b]$. Их положение строго фиксировано заранее и никак не зависит от информации о функции $f(x)$, которую мы получаем по мере вычисления чисел $y_k = (x$. Такой подход дает воз-

можность внимательно просмотреть весь отрезок. Он помогает наиболее надежно избежать пропуска какого-нибудь узкого «языка» у целевой функции. В этом бесспорное достоинство метода.

Однако недаром говорят, что наши недостатки — это продолжение наших достоинств. Распределяя точки x_k равномерно, мы уделяем одинаковое внимание всем участкам отрезка: тем, где целевая функция велика, и тем, в направлении которых она убывает. Это существенно удлиняет расчеты и затягивает исследование.

Организацию вычислений по такой схеме можно сравнить с поведением в лесу неопытного грибника. В поисках грибов он ходит по всему лесу, не чувствуя разницы между грибными и негрибными местами. В результате ему приходится тратить много сил и времени понапрасну на осмотр негрибных мест. Совершенно иначе ведет себя опытный грибник. Он подолгу задерживается на грибных местах и осматривает их особенно внимательно, а через негрибные места проходит быстро, не тратя на них лишнего времени.

Чтобы организовать поиск наименьшего значения функции по методу «опытного грибника», нужно отказаться от предписанного заранее выбора точек x_k, а выбирать очередную точку с учетом информации, которую мы уже получили о функции $f(x)$ в результате ее вычисления в предыдущих точках. При этом основное внимание следует уделять тем участкам отрезка $[a, b]$, где вычисления дают малые значения функции, просматривая другие участки более бегло. Реализовать эту идею можно, например, следующим образом.

Вычислим значения функции $f(x)$ в двух граничных точках $x_0=a$ и $x_1=b$: $y_0=f(x_0)$, $y_1=f(x_1)$. После этого следующую точку x_2 выберем ближе к тому концу отрезка, на котором функция принимает меньшее значение. Ее положение определим соотношением между числами y_0 и y_1: чем больше разница между ними, тем сильнее будет сдвиг точки x_2 в соответствующую сторону. Точку x_3 выберем с учетом взаимного расположения точек x_0, x_1, x_2 и соотношения между числами y_0, y_1, y_2 и т. д. Мы не будем останавливаться на описании возможных алгоритмов выбора очередной точки x_k по информации, полученной в результате вычисления целевой функции на предыдущих шагах,— это специальный вопрос. Отметим лишь, что исследования в данной области продолжаются, алгоритмы совершенствуются, и пока рано говорить об окончательном решении задачи.

3. Специальные методы. Воспользуемся еще раз примером о сборе грибов, чтобы поставить новые вопросы о методах решения задачи оптимизации. Грибник может попасть в данный лес впервые и ничего не знать о нем заранее, кроме одного: грибы есть. Возможен и другой случай. Человек приходит в лес, который он уже немного знает. Его поведение в первом и во втором случае будет различным. Причем, если он сумеет правильно воспользоваться известными ему особенностями леса, то наполнит корзину грибами гораздо быстрее.

До сих пор, обсуждая задачи оптимизации, мы говорили об универсальных методах их решения. Однако во многих случаях из характера задачи вытекает какая-то дополнительная информация о свойствах целевой функции. Это может быть использовано для разработки специальных алгоритмов. Такой подход позволяет существенно сократить объем вычислений и получить ответ наиболее эффективным способом.

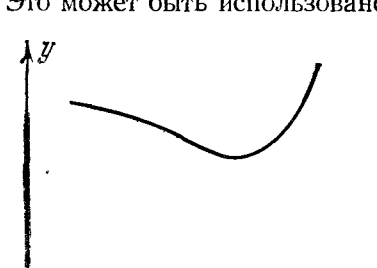

Рис. 18. Пример функции, имеющей один минимум.

В качестве примера рассмотрим случай, когда нам известно заранее, что целевая функция $y=f(x)$ имеет на отрезке $[a, b]$ только один минимум. График такой функции показан на рис. 18.

Для решения задачи в этом случае можно воспользоваться следующим методом. Возьмем некоторый шаг h и будем последовательно вычислять значения функции $f(x)$ в точках $x_0=a$, $x_1=a+h$, $x_2=a+2h$, . . ., сравнивая получаемые числа y_0, y_1, y_2, ... Сначала они будут убывать: $y_0 > y_1 > y_2 > \ldots$, однако в дальнейшем найдется такая точка $x_k=a+kh$, что для значения в ней функции $y_k=f(x_k)$ справедливы неравенства $y_{k-1} > y_k$, $y_{k+1} \geqslant y_k$. Это означает, что наименьшее значение функции достигается на отрезке $[x_{k-1}, x_{k+1}]$ и его приближенно можно принять равным $y_k=f(x_k)$. Если требуемая точность в решении задачи еще не обеспечена, то нужно уменьшить шаг h и повторить описанную процедуру для отрезка $[x_{k-1}, x_{k+1}]$.

Задача об оптимальной температуре для химического процесса часто относится к задачам подобного типа. График функции $f(T)$ для многих химических реакций имеет

вид кривой на рис. 18, но только перевернутой: при увеличении температуры T функция сначала возрастает, а потом, пройдя через максимум, начинает убывать. Нам нужно найти этот максимум, для чего можно воспользоваться описанной выше процедурой. Она потребует небольшого числа измерений функции $f(T)$. То, что мы ищем максимум, а не минимум, не имеет принципиального значения с точки зрения метода, просто все неравенства изменят свои знаки на противоположные.

§ 4. Многомерные задачи оптимизации

До сих пор мы обсуждали одномерные задачи оптимизации, в которых целевая функция зависела только от одного аргумента. Однако подавляющее число реальных задач оптимизации, представляющих практический интерес, являются многомерными: в них целевая функция зависит от нескольких аргументов, причем иногда их число может быть весьма большим.

Вспомним, например, задачу о химическом производстве. Мы отметили, что в ней целевая функция зависит от температуры, и при определенном ее выборе производительность (выход интересующего нас продукта) оказывается максимальной. Однако, наряду с температурой, производительность зависит также от давления, соотношения между концентрациями вводимого сырья, катализаторов и ряда других факторов. Таким образом, задача выбора наилучших условий химического производства — это типичная многомерная задача оптимизации.

Математическая постановка таких задач аналогична их постановке в одномерном случае: ищется наименьшее (наибольшее) значение целевой функции, заданной на некотором множестве E возможных значений ее аргументов. В случае, когда целевая функция непрерывна, а множество E является замкнутой ограниченной областью, остается справедливой теорема Вейерштрасса. Тем самым выделяется класс задач оптимизации, для которых гарантировано существование решения. В дальнейшем мы всегда будем предполагать, не оговаривая этого особо, что все рассматриваемые задачи принадлежат этому классу.

Как и в одномерном случае, характер задачи и соответственно возможные методы решения существенно зависят от той информации о целевой функции, которая нам доступна в процессе ее исследования. В одних случаях целе-

вая функция задается аналитической формулой, являясь при этом дифференцируемой функцией. Тогда можно вычислить ее частные производные, получить явное выражение для градиента, определяющего в каждой точке направления

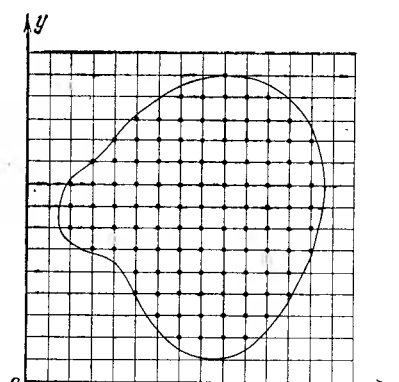

Рис. 19. Построение сетки с шагом h и выбор «пробных» точек в узлах сетки для приближенного определения наименьшего значения функции двух переменных.

возрастания и убывания функций, и использовать эту информацию для решения задачи. В других случаях никакой формулы для целевой функции нет, а имеется лишь возможность определить ее значение в любой точке рассматриваемой области (с помощью расчетов, в результате эксперимента и т. д.). В таких задачах в процессе решения мы фактически можем найти значения целевой функции лишь в конечном числе точек, и по этой информации требуется приближенно установить ее наименьшее значение для всей области.

Многомерные задачи, естественно, являются более сложными и трудоемкими, чем одномерные, причем обычно трудности при их решении возрастают при увеличении размерности. Для того чтобы вы лучше почувствовали это, возьмем самый простой по своей идее приближенный метод поиска наименьшего значения функции, который уже обсуждался для одномерных задач в § 3. Покроем рассматриваемую область сеткой с шагом h (рис. 19) и определим значения функции в ее узлах. Сравнивая полученные числа между собой, найдем среди них наименьшее и примем его приближенно за наименьшее значение функции для всей области.

Как мы уже говорили выше, данный метод используется для решения одномерных задач. Иногда он применяется

также для решения двумерных, реже трехмерных задач. Однако для задач большей размерности он практически непригоден из-за слишком большого времени, необходимого для проведения расчетов.

Действительно, предположим, что целевая функция зависит от пяти переменных, а область определения является пятимерным кубом, каждую сторону которого при построении сетки мы делим на 40 частей. Тогда общее число узлов сетки будет равно $41^5 \approx 10^8$. Пусть вычисление значения функции в одной точке требует 1000 арифметических операций (это немного для функции пяти переменных). В таком случае общее число операций составит 10^{11}. Если в нашем распоряжении имеется ЭВМ с быстродействием 1 млн. операций в секунду, то для рещения задачи с помощью данного метода потребуется 10^5 секунд, что превышает сутки непрерывной работы. Добавление еще одной независимой переменной увеличит это время в 40 раз.

Проведенная оценка показывает, что для больших задач оптимизации метод сплошного перебора непригоден. Иногда сплошной перебор заменяют случайным поиском. В этом случае точки сетки просматриваются не подряд, а в случайном порядке. В результате поиск наименьшего значения целевой функции существенно ускоряется, но теряет свою надежность.

Перейдем к обсуждению методов, позволяющих вести поиск наименьшего значения функции целенаправленно.

1. Метод покоординатного спуска. Пусть нужно найти наименьшее значение целевой функции $u=f(M)=f(x_1, x_2, \ldots, x_n)$. Здесь через M обозначена точка n-мерного пространства с координатами $x_1, x_2, \ldots, x_n$: $M=(x_1, x_2, \ldots, x_n)$. Выберем какую-нибудь начальную точку $M_0=(x_{10}, x_{20}, \ldots, x_{n0})$ и рассмотрим функцию f при фиксированных значениях всех переменных, кроме первой: $f(x_1, x_{20}, x_{30}, \ldots, x_{n0})$. Тогда она превратится в функцию одной переменной x_1. Изменяя эту переменную, будем двигаться от начальной точки $x_1=x_{10}$ в сторону убывания функции, пока не дойдем до ее минимума при $x_1=x_{11}$, после которого она начинает возрастать. Точку с координатами $(x_{11}, x_{20}, x_{30}, \ldots, x_{n0})$ обозначим через M_1, при этом $f(M_0) \geqslant (M_1)$.

Фиксируем теперь переменные: $x_1=x_{11}$, $x_3=x_{30}$, $\ldots, x_n=x_{n0}$ и рассмотрим функцию f как функцию одной переменной x_2: $f(x_{11}, x_2, x_{30}, \ldots, x_{n0})$. Изменяя x_2, будем опять двигаться от начального значения $x_2=x_{20}$ в сторону убывания функции, пока не дойдем до минимума при

$x_2=x_{21}$. Точку с координатами $\{x_{11}, x_{21}, x_{30}, \ldots, x_{n0}\}$ обозначим через M_2, при этом $f(M_1)\geqslant f(M_2)$.

Проведем такую же минимизацию целевой функции по переменным $x_3, x_4, \ldots, x_n$. Дойдя до переменной x_n, снова вернемся к x_1 и продолжим процесс. Эта процедура вполне оправдывает название метода. С ее помощью мы построим последовательность точек M_0, M_1, M_2, ..., которой соответствует монотонная последовательность значений функции $f(M_0)\geqslant f(M_1)\geqslant f(M_2)\geqslant\ldots$ Обрывая ее на некотором шаге k, можно приближенно принять значение функции $f(M_k)$ за ее наименьшее значение в рассматриваемой области.

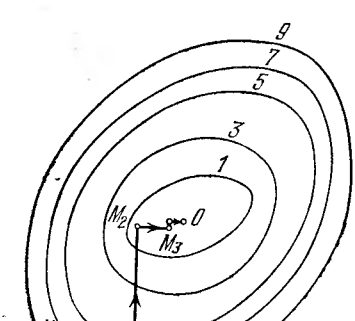

Рис. 20. Поиск наименьшего значения функции методом покоординатного спуска.

Отметим, что данный метод сводит задачу поиска наименьшего значения функции нескольких переменных к многократному решению одномерных задач оптимизации. Если целевая функция $f(x_1, x_2, \ldots, x_n)$ задана явной формулой и является дифференцируемой, то мы можем вычислить ее частные производные и использовать их для определения направления убывания функции по каждой переменной и поиска соответствующих одномерных минимумов. В противном случае, когда явной формулы для целевой функции нет, одномерные задачи следует решать с помощью методов, описанных в § 3.

На рис. 20 изображены линии уровня некоторой функции двух переменных $u=f(x, y)$. Вдоль этих линий функция сохраняет постоянные значения, равные 1, 3, 5, 7, 9. Показана траектория поиска ее наименьшего значения, которое достигается в точке O, с помощью метода покоординатного спуска. При этом нужно ясно понимать, что рисунок служит только для иллюстрации метода. Когда мы приступаем к решению реальной задачи оптимизации, такого рисунка, содержащего в себе готовый ответ, у нас, конечно, нет.

2. Метод градиентного спуска. Рассмотрим функцию f, считая для определенности, что она зависит от трех переменных x, y, z. Вычислим ее частные производные $\partial/\partial x$,

$\partial f/\partial y$, $\partial f/\partial z$ и образуем с их помощью вектор, который называют *градиентом* функции:

$$\operatorname{grad} f(x, y, z) = \frac{\partial f}{\partial x}(x, y, z)\,\boldsymbol{i} + \frac{\partial f}{\partial y}(x, y, z)\,\boldsymbol{j} + \frac{\partial f}{\partial z}(x, y, z)\,\boldsymbol{k}.$$

Здесь $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$ — единичные векторы, параллельные координатным осям. Частные производные характеризуют изменение функции f по каждой независимой переменной в отдельности. Образованный с их помощью вектор градиента дает общее представление о поведении функции в окрестности точки (x, y, z). Направление этого вектора является направлением наиболее быстрого возрастания функции в данной точке. Противоположное ему направление, которое часто называют *антиградиентным*, представляет собой направление наиболее быстрого убывания функции. Модуль градиента

$$|\operatorname{grad} f(x, y, z)| = \sqrt{\left(\frac{\partial f}{\partial x}(x, y, z)\right)^2 + \left(\frac{\partial f}{\partial y}(x, y, z)\right)^2 + \left(\frac{\partial f}{\partial z}(x, y, z)\right)^2}$$

определяет скорость возрастания и убывания функции в направлении градиента и антиградиента. Для всех остальных направлений скорость изменения функции в точке (x, y, z) меньше модуля градиента. При переходе от одной точки к другой как направление градиента, так и его модуль, вообще говоря, меняются. Понятие градиента естественным образом переносится на функции любого числа переменных.

Перейдем к описанию метода градиентного спуска. Основная его идея состоит в том, чтобы двигаться к минимуму в направлении наиболее быстрого убывания функции, которое определяется антиградиентом. Эта идея реализуется следующим образом.

Выберем каким-либо способом начальную точку, вычислим в ней градиент рассматриваемой функции и сделаем небольшой шаг в обратном антиградиентном направлении. В результате мы придем в точку, в которой значение функции будет меньше первоначального. В новой точке повторим процедуру: снова вычислим градиент функции и сделаем шаг в обратном направлении. Продолжая этот процесс, мы будем двигаться в сторону убывания функции. Специальный выбор направления движения на каждом шаге позволяет надеяться на то, что в данном случае приближение к наименьшему значению функции будет более быстрым, чем в методе покоординатного спуска.

Метод градиентного спуска требует вычисления градиента целевой функции на каждом шаге. Если она задана аналитически, то это, как правило, не проблема: для частных производных, определяющих градиент, можно получить явные формулы. В противном случае частные производные в нужных точках приходится вычислять приближенно, заменяя их соответствующими разностными отношениями:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_1, \ldots, x_i + \Delta x_i, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{\Delta x_i}.$$

Отметим, что при таких расчетах Δx_i нельзя брать слишком малым, а значения функции нужно вычислять с достаточно высокой степенью точности, иначе при вычислении разности

$$\Delta f = f(x_1, \ldots, x_i + \Delta x_i, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)$$

будет допущена большая ошибка.

На рис. 21 изображены линии уровня той же функции двух переменных $u = f(x, y)$, что и на рис. 20, и приведена

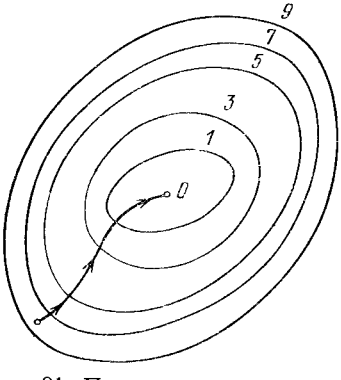

Рис. 21. Поиск наименьшего значения функции методом градиентного спуска.

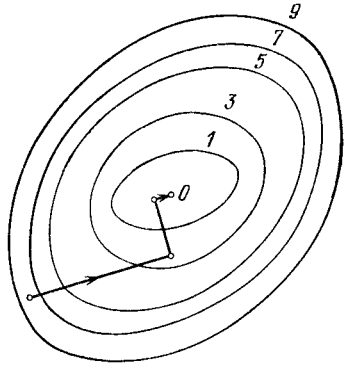

Рис. 22. Поиск наименьшего значения функции методом наискорейшего спуска.

траектория поиска ее минимума с помощью метода градиентного спуска. Сравнение рис. 20 и 21 показывает, насколько более эффективным является метод градиентного спуска.

3. Метод наискорейшего спуска. Вычисление градиента на каждом шаге, позволяющее все время двигаться в направлении наиболее быстрого убывания целевой функции,

может в то же время замедлить вычислительный процесс. Дело в том, что подсчет градиента — обычно гораздо более сложная операция, чем подсчет самой функции. Поэтому часто пользуются модификацией градиентного метода, получившей название метода наискорейшего спуска.

Согласно этому методу после вычисления в начальной точке градиента функции делают в направлении антиградиента не маленький шаг, а движутся до тех пор, пока функция убывает. Достигнув точки минимума на выбранном направлении, снова вычисляют градиент функции и повторяют описанную процедуру. При этом градиент вычисляется гораздо реже, только при смене направлений движения.

На рис. 22 показана траектория поиска наименьшего значения целевой функции по методу наискорейшего спуска. Функция выбрана та же, что и на рис. 20, 21. Хотя траектория ведет к цели не так быстро, как на рис. 21, экономия машинного времени за счет более редкого вычисления градиента может быть весьма существенной.

4. Проблема «оврагов». Мы рассказали о трех вариантах методов спуска и показали на рис. 20—22 как хорошо они работают. В результате у вас могло сложиться впечатление, что проблема решена. На самом деле это не так. Все было хорошо потому, что был выбран «удобный» пример. Но посмотрите на рис. 23. На нем также показаны линии уровня некоторой функции, однако их конфигурация отличается от рис. 20—22. Линии уровня сильно вытянуты в одном направлении и сплющены в другом. Они напоминают рельеф местности с оврагом. Этот случай крайне неудобен для описанных выше методов.

Действительно, попытаемся найти наименьшее значение такой функции с помощью градиентного спуска. Двигаясь все время в направлении антиградиента, мы быстро спустимся на дно «оврага» и, поскольку движение идет хотя и маленькими, но конечными шагами, проскочим его. Оказавшись на противоположной стороне «оврага» и вычислив там градиент функции, мы будем вынуждены развернуться почти на 180° и сделать один или несколько шагов в обратном направлении. При этом мы снова проскочим дно «оврага» и вернемся на его первоначальную сторону. Продолжая этот процесс, мы вместо того, чтобы двигаться по дну «оврага» в сторону его понижения, будем совершать зигзагообразные скачки поперек «оврага», почти не приближаясь к цели. Таким образом, в случае «оврага» (этот нематематический тер-

мин прочно закрепился в литературе) описанные выше методы спуска оказываются неэффективными.

Для борьбы с «оврагами» был предложен ряд специальных приемов. Один из них заключается в следующем. Из двух близких точек совершают градиентный спуск на дно «оврага». Потом соединяют найденные точки прямой и делают вдоль нее большой (овражный) шаг. Из найденной точки снова спускаются на дно «оврага» и делают второй

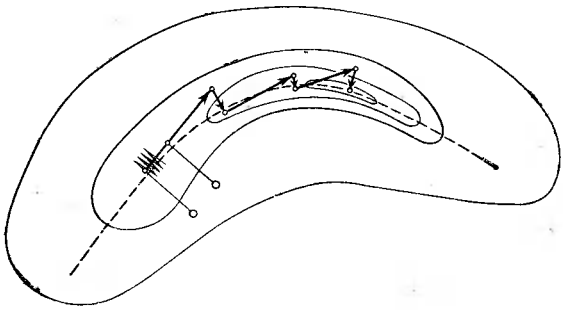

Рис. 23. Поиск наименьшего значения функции в случае «оврага».

овражный шаг. В результате, двигаясь достаточно быстро вдоль «оврага», приближаемся к искомому наименьшему значению целевой функции (см. рис. 23). Такой метод достаточно эффективен для функций двух переменных, однако при большем числе переменных могут возникнуть трудности.

Все описанные выше методы приспособлены к случаю, когда наименьшее значение функции достигается внутри рассматриваемой области, и становятся малоэффективными, если наименьшее значение достигается на границе или вблизи нее. Для решения таких задач приходится разрабатывать специальные методы. Мы не будем на них останавливаться. Вам должно быть и без того ясно, что большое число специальных методов — это признак слабости, а не силы. Ведь, приступая к решению практической задачи, мы, как правило, не знаем всех ее особенностей и не можем сразу выбрать наиболее эффективный метод.

5. Проблема многоэкстремальности. Посмотрите на рис. 20—23 и сравните их с рис. 24. Первые четыре рисунка относятся к функциям, имеющим только один минимум. Поэтому, откуда бы ни начинался поиск, мы придем в конце

концов к нужной точке. На рис. 24 приведены линии уровня функции с двумя локальными минимумами в точках O_1 и O_2. Такие функции принято называть *многоэкстремальными*. Сравнивая между собой значения функции в точках O_1 и O_2: $f_1=3$, $f_2=1$, находим, что наименьшее значение функция достигает в точке O_2.

Рис. 24. Пример функции с двумя локальными минимумами в точках O_1 и O_2.

Представьте себе теперь, что, не имея перед глазами рис. 24 и не зная о многоэкстремальности функции, мы начали поиск наименьшего значения с помощью метода градиентного спуска из точки A_1. Поиск приведет нас в точку O_1, которую ошибочно можно принять за искомый ответ. С другой стороны, если мы начнем поиск с точки A_2, то окажемся на правильном пути и быстро придем в точку O_2.

Как бороться с многоэкстремальностью? Универсального ответа на этот вопрос нет. Самый простой прием состоит в том, что проводят поиск несколько раз, начиная его с разных точек. Если при этом получаются разные ответы, то сравнивают в них значения целевой функции и выбирают наименьшее. Расчеты останавливают после того, как несколько новых поисков не меняют полученного ранее результата. Выбор начальных точек поиска, обоснованность прекращения расчетов в значительной степени зависят от опыта и интуиции специалистов, решающих задачу.

Нарисованная картина может показаться слишком мрачной. На самом деле во многих случаях имеется различная дополнительная информация о характере задачи, которая

существенно помогает при выборе метода, начальной точки поиска и т. д. Кроме того, пока мы не делали никаких предположений о специальных свойствах целевой функции и о характере рассматриваемой области. Это затрудняет анализ. Конкретизация задачи, выделение определенных классов функций и областей позволяет провести более глубокое исследование и разработать специальные методы, которые решают задачу исчерпывающим образом. Важным классом таких «специализированных» задач оптимизации являются *задачи линейного программирования*, о которых будет рассказано в следующем параграфе. Существуют и другие типы задач оптимизации, конкретные специфические особенности которых позволяют провести их детальный анализ и разработать эффективные методы решения. Однако на них мы останавливаться не будем.

§ 5. Линейное программирование

В этом параграфе мы познакомимся с линейным программированием. Так называют задачи оптимизации, в которых целевая функция является линейной функцией своих аргументов, а условия, определяющие их допустимые значения, имеют вид линейных уравнений и неравенств. Линейное программирование начало развиваться в первую очередь в связи с задачами экономики, с поиском способов оптимального распределения и использования ресурсов. Оно послужило основой широкого использования математических методов в экономике. Следует подчеркнуть, что в рамках реальных экономических задач число независимых переменных обычно бывает очень большим (тысячи, десятки тысяч аргументов). Поэтому практическая реализация алгоритмов решения таких задач принципиально невозможна без использования современной вычислительной техники.

Рассмотрим в качестве примера две типичные задачи линейного программирования: транспортную задачу и задачу об использовании ресурсов.

Транспортная задача. В городе имеются два склада муки и два хлебозавода. Ежедневно с первого склада вывозится 50 т муки, со второго — 70 т. Эта мука доставляется на хлебозаводы, причем первый завод получает 40 т, второй — 80 т. Допустим, что перевозка одной тонны муки с первого склада на первый завод стоит 1 р. 20 к., с первого склада на второй завод — 1 р. 60 к., со второго склада на первый завод — 80 к. и со второго склада на второй завод — 1 р.

Как нужно спланировать перевозки, чтобы их стоимость была минимальной?

Для того чтобы ответить на этот вопрос, дадим математическую постановку задачи. Обозначим через x_1 и x_2 количество муки, которую следует перевезти с первого склада соответственно на первый и второй заводы, а через x_3 и x_4 — количество муки, которую нужно перевезти со второго склада на первый и второй заводы. Эти условия приводят к системе уравнений

$$\begin{aligned} x_1 + x_2 &= 50, \\ x_3 + x_4 &= 70, \\ x_1 + x_3 &= 40, \\ x_2 + x_4 &= 80, \end{aligned} \tag{15}$$

$$x_i \geqslant 0, \qquad i = 1, 2, 3, 4. \tag{16}$$

Первые два уравнения системы определяют, сколько муки нужно вывести с каждого склада, два других уравнения показывают, сколько муки нужно привезти на каждый завод. Неравенства (16) означают, что в обратном направлении с заводов на склады муку не возят. Общая стоимость всех перевозок определяется формулой

$$f = 1{,}2x_1 + 1{,}6x_2 + 0{,}8x_3 + x_4. \tag{17}$$

С математической точки зрения задача заключается в том, чтобы найти числа x_i ($i = 1, 2, 3, 4$), удовлетворяющие условиям (15), (16) и минимизирующие стоимость перевозок (17).

Рассмотрим систему (15). Это система четырех уравнений с четырьмя неизвестными. Однако независимыми в ней являются только первые три уравнения, 4-е — их следствие (если сложить 1-е и 2-е уравнения и вычесть 3-е, получится 4-е). Таким образом, фактически нужно рассмотреть следующую систему, эквивалентную (15):

$$\begin{aligned} x_1 + x_2 &= 50, \\ x_3 + x_4 &= 70, \\ x_1 + x_3 &= 40. \end{aligned} \tag{18}$$

В ней число уравнений на единицу меньше числа неизвестных, так что мы можем выбрать какую-нибудь неизвестную, например x_1, и выразить через нее с помощью уравнений

(18) три остальных. Соответствующие формулы имеют вид

$$\begin{aligned} x_2 &= 50 - x_1, \\ x_3 &= 40 - x_1, \\ x_4 &= 30 + x_1. \end{aligned} \tag{19}$$

Учитывая (16), получаем систему

$$\begin{aligned} x_1 &\geqslant 0, \\ 50 - x_1 &\geqslant 0, \\ 40 - x_1 &\geqslant 0, \\ 30 + x_1 &\geqslant 0, \end{aligned} \tag{20}$$

из которой

$$0 \leqslant x_1 \leqslant 40. \tag{21}$$

Таким образом, задавая любое x_1, удовлетворяющее (21), и вычисляя x_2, x_3, x_4 по формулам (19), мы получим один из возможных планов перевозки. При реализации этого плана с каждого склада будет вывезено и на каждый завод доставлено нужное количество муки.

Вычислим стоимость перевозок. Для этого подставим (19) в формулу (17). В результате получим

$$f = 142 - 0{,}2x_1. \tag{22}$$

Эта формула определяет величину f как функцию одной переменной x_1, которую можно выбирать произвольно в пределах условий (21). Стоимость окажется минимальной, если мы придадим величине x_1 наибольшее возможное значение: $x_1 = 40$. Значения остальных величин x_i находятся по формулам (19).

Итак, оптимальный по стоимости план перевозок имеет вид

$$x_1 = 40, \quad x_2 = 10, \quad x_3 = 0, \quad x_4 = 70. \tag{23}$$

Стоимость перевозок в этом случае составляет 134 р. При любом другом допустимом плане перевозок она окажется выше: $f > f_{\min} = 134$

Задача об использовании ресурсов. Мебельная фабрика выпускает стулья двух типов. На изготовление одного стула первого типа, стоимостью 8 р., расходуется 2 м досок стандартного сечения, 0,5 м^2 обивочной ткани и 2 чел./ч рабочего времени. Для стульев второго типа аналогичные данные составляют: 12 р., 4 м, 0,25 м^2 и 2,5 чел./ч.

Допустим, что в распоряжении фабрики имеется 440 м досок, 65 м² обивочной ткани, 320 чел./ч рабочего времени. Какое количество стульев каждого типа надо изготовить, чтобы в рамках этих ресурсов стоимость произведенной продукции была максимальной?

Для ответа на этот вопрос постараемся опять сформулировать задачу как математическую. Обозначим через x_1 и x_2 запланированное к производству число стульев соответственно первого и второго типов. Ограниченный запас сырья и трудовых ресурсов означает, что x_1 и x_2 должны удовлетворять неравенствам

$$\begin{aligned} 2x_1 + 4x_2 &\leqslant 440, \\ \frac{1}{2}x_1 + \frac{1}{4}x_2 &\leqslant 65, \\ 2x_1 + \frac{5}{2}x_2 &\leqslant 320. \end{aligned} \tag{24}$$

Кроме того, по смыслу задачи они должны быть неотрицательными:

$$x_1 \geqslant 0, \quad x_2 \geqslant 0. \tag{25}$$

Стоимость запланированной к производству продукции определяется формулой

$$f(x_1, x_2) = 8x_1 + 12x_2. \tag{26}$$

Итак, с математической точки зрения задача составления оптимального по стоимости выпущенной продукции плана фабрики сводится к определению пары целых чисел x_1, x_2, удовлетворяющих линейным неравенствам (24), (25) и дающих наибольшее значение линейной функции (26). Мы опять получили типичную задачу линейного программирования. По своей постановке она несколько отличается от транспортной задачи, однако это различие не существенно.

Для анализа сформулированной задачи рассмотрим плоскость и введем на ней декартову систему координат x_1, x_2. Найдем на этой плоскости множество точек, координаты которых удовлетворяют (24), (25). Неравенства (25) означают, что это множество лежит в первой четверти. Выясним теперь смысл ограничений, которые задаются неравенствами (24).

Проведем на плоскости прямую, определяемую уравнением

$$2x_1 + 4x_2 = 440 \tag{27}$$

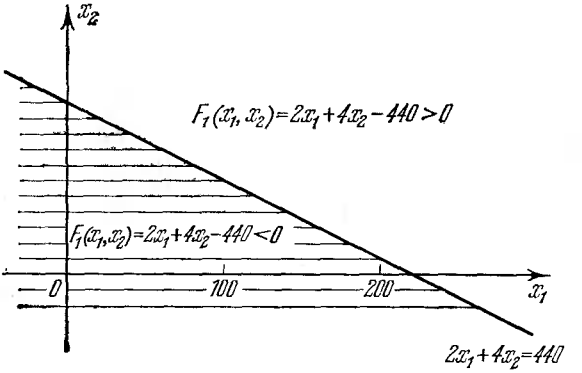

Рис. 25. Решение неравенства $2x_1+4x_2 \leqslant 440$.

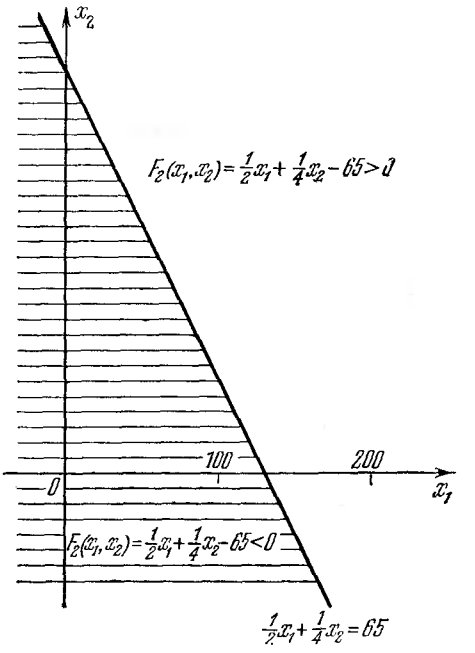

Рис. 26. Решение неравенства $\frac{1}{2}x_1+\frac{1}{4}x_2 \leqslant 65$.

(рис. 25). Она делит плоскость на две полуплоскости. На одной из них, расположенной ниже прямой (27), функция $F_1(x_1, x_2)=2x_1+4x_2-440$ принимает отрицательные значения, на другой, расположенной выше прямой (27),— положительные. Таким образом, первое из неравенств (24) выполняется на множестве точек, которое включает в себя прямую (27) и полуплоскость, расположенную ниже этой прямой. На рис. 25 соответствующая часть плоскости заштрихована.

Совершенно аналогично можно найти множества точек, удовлетворяющих второму и третьему неравенствам из системы (24). Они показаны на рис. 26, 27 штриховкой.

Возьмем пересечение трех найденных множеств и выделим его часть, расположенную в первой четверти. В результате получим множество точек, удовлетворяющих всей совокупности ограничений (24), (25). Данное множество имеет вид пятиугольника, показанного на рис. 28. Его вершинами являются точки пересечения прямых, на которых неравенства (24), (25) переходят в точные равенства. Координаты вершин пятиугольника указаны на рис. 28.

Любой точке P с целочисленными координатами (x_1, x_2), принадлежащей данному пятиугольнику, соответствует план выпуска стульев, который может быть выполнен при имеющихся запасах сырья и трудовых ресурсов (реализуемый план). Наоборот, если точка P не принадлежит пятиугольнику, то соответствующий план не может быть выполнен (нереализуемый план).

Рассмотрим на плоскости x_1, x_2 линии уровня целевой функции (26):

$$8x_1+12x_2=C. \qquad (28$$

Это уравнение описывает семейство прямых, параллельных прямой

$$8x_1+12x_2=0. \qquad (29$$

При параллельном переносе этой прямой вправо параметр C возрастает, влево — убывает.

Свойства функции (26) тесно связаны с прямыми (28). Вдоль каждой из них она сохраняет постоянное значение, равное C, а при переходе с одной прямой на другую ее значение меняется.

Будем рассматривать только первую четверть. Предположим, что мы перешли из точки P_1, расположенной на одной прямой, в точк P расположенную на друго

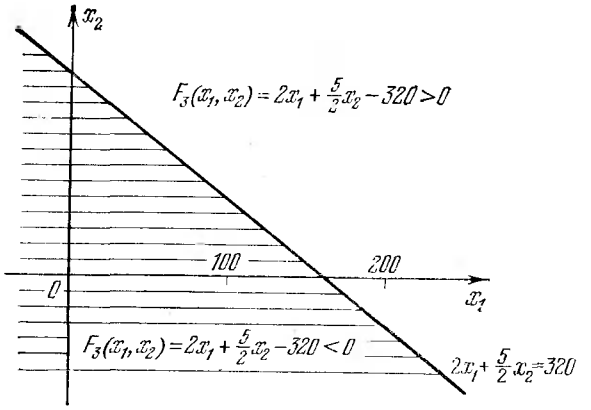

Рис. 27. Решение неравенства $2x_1 + \frac{5}{2}x_2 \leqslant 320$.

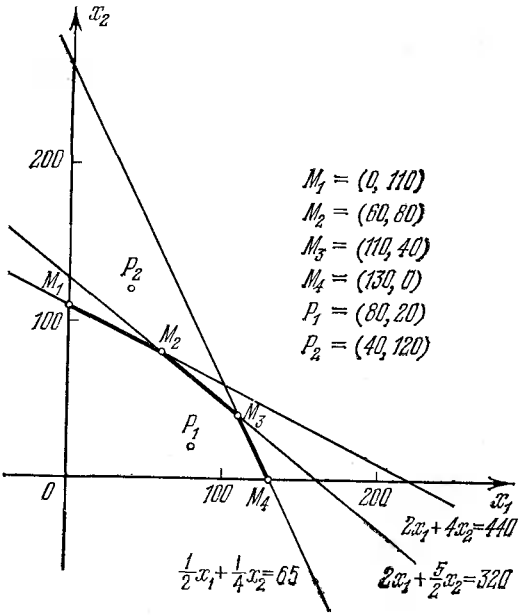

Рис. 28. Пятиугольник $OM_1M_2M_3M_4$, точки которого удовлетворяют системе неравенств (24), (25).

прямой (рис. 29). Если вторая прямая расположена дальше от начала координат, чем первая, то функция f при этом переходе возрастет. Отсюда следует важный вывод: оптимальный план должен располагаться на прямой семейства (28), наиболее удаленной от начала координат.

Этот вывод позволяет закончить решение задачи. Посмотрите на рис. 30. На нем воспроизведен пятиугольник реализуемых планов и нарисована прямая семейства (28), проходящая через точку M_2 с координатами (60, 80). Она является предельной прямой семейства, имеющей общую точку с пятиугольником. Если мы попытаемся с помощью параллельного переноса отодвинуть ее дальше от начала координат, то получим прямую, не имеющую общих точек с пятиугольником, т. е. соответствующие планы нереализуемы.

Итак, оптимальный план найден,— он предписывает производство 60 стульев первого типа и 80 стульев второго типа. Стоимость этой продукции 1440 р. На выполнение плана нужно затратить: 440 м досок, 50 м² обивочной ткани, 320 чел./ч рабочего времени.

Вы видите, что оптимальный план требует полного использования запаса досок и трудовых ресурсов, в то время как обивочная ткань будет израсходована не полностью — останется 15 м².

Этот результат ясен из рис. 30. Точка M_2, определяющая оптимальный план, является вершиной пятиугольника. Она лежит на пересечении прямых

$$2x_1 + 4x_2 = 440,$$
$$2x_1 + \frac{5}{2}x_2 = 320.$$

Уравнения этих прямых получаются из первого и третьего условий системы (24) при замене их на строгие равенства. Это означает полный расход досок и трудовых ресурсов. Однако точка M_2 не принадлежит прямой

$$\frac{1}{2}x_1 + \frac{1}{4}x_2 = 65,$$

так что второе условие (24), связанное с ограниченным запасом обивочной ткани, имеет в ней форму неравенства

$$50 < 65.$$

Проведенный анализ показывает, что дальнейшее увеличение стоимости продукции регламентируется запасом досок и трудовыми ресурсами.

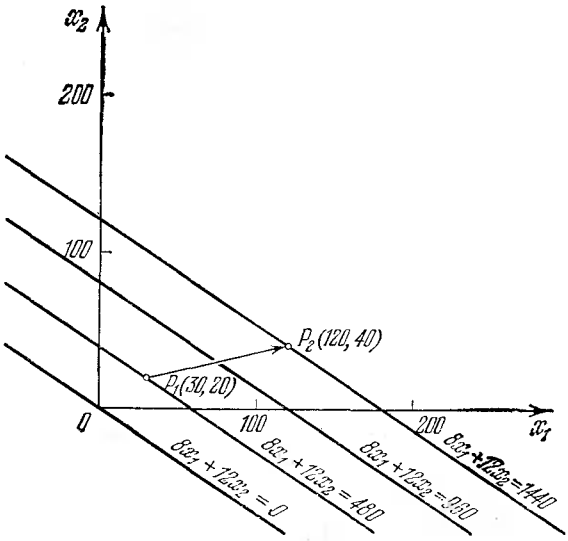

Рис. 29. Линии уровня функции $f(x_1, x_2)=8x_1+12x_2$. Возрастание функции при переходе из точки P_1 в точку P_2.

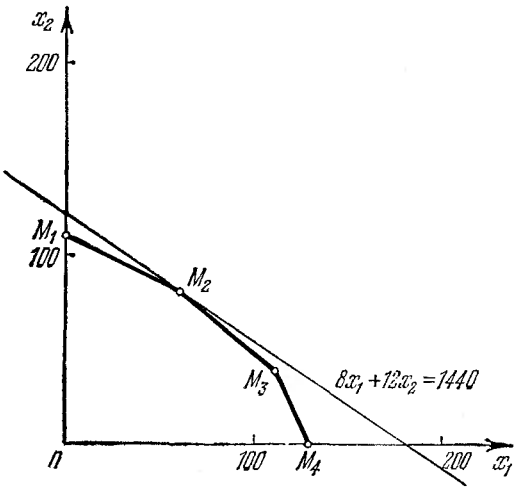

Рис. 30. Определение оптимального плана производства стульев.

Мы рассмотрели две задачи линейного программирования. По своей постановке они носят достаточно общий характер, несмотря на «учебный» вариант конкретных условий. Небольшое число переменных (4 маршрута, 2 вида продукции) позволило просто и наглядно получить их решение.

В настоящее время разработаны и реализованы в виде стандартных программ общие алгоритмы решения задач линейного программирования. Они позволяют получать ответ наиболее эффективным путем с минимальными затратами машинного времени. Применение ЭВМ к решению задач линейного программирования, начавшееся в 50-е годы, послужило основой широкого применения математических методов в экономике.

ГЛАВА 7

ОПРЕДЕЛЕННЫЙ ИНТЕГРАЛ. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

§ 1. Как подсчитать путь при неравномерном движении или работу переменной силы

Пусть материальная точка движется со скоростью $v(t)$. Требуется определить путь, который она пройдет за время $T_1 \leqslant t \leqslant T_2$. Если точка движется равномерно, т. е. $v(t) = v = \text{const}$, то ответ дается простой формулой:

$$S = v(T_2 - T_1).$$

Подсчитаем пройденный путь в случае неравномерного движения. Для этого фиксируем между начальным и конечным моментами времени ряд промежуточных моментов: $T_1 < t_1 < t_2 < \ldots < t_{n-1} < T_2$. Положим также, для единообразия обозначений, $t_0 = T_1$ и $t_n = T_2$. В результате исходный временной отрезок $[T_1, T_2]$ окажется разбитым на более мелкие отрезки $[t_{k-1}, t_k]$ $(k = 1, 2, \ldots, n)$.

Рассмотрим один из этих отрезков. Выберем на нем какой-нибудь момент времени $\tau_k \in [t_{k-1}, t_k]$ и найдем соответствующее ему значение скорости $v_k = v(\tau_k)$. Если данный временной отрезок является достаточно коротким, то скорость $v(t)$ изменяется в течение него мало. Пренебрегая этими изменениями, будем приближенно считать движение на этом отрезке равномерным со скоростью $v_k = v(\tau_k)$ $(t_{k-1} \leqslant t \leqslant t_k)$. В результате за данный отрезок времени будет пройден путь

$$\Delta s_k \approx v(\tau_k)\,\Delta t_k, \qquad \Delta t_k = t_k - t_{k-1}.$$

Чтобы подсчитать весь путь, нужно проделать данную процедуру для каждого временного отрезка и результаты сложить:

$$s \approx v(\tau_1)\,\Delta t_1 + v(\tau_2)\,\Delta t_2 + \ldots + v(\tau_n)\,\Delta t_n. \qquad (1)$$

Полученное приближенное выражение зависит от того, как мы разбили временной отрезок $[T_1, T_2]$ на части и как для каждой из них выбрали момент времени τ_k, по которому определяется скорость равномерного движения $v_k = v(\tau_k)$, заменяющего заданное неравномерное движение при $t \in [t_{k-1}, t_k]$. Разным разбиениям и разному выбору моментов τ_k соответствуют разные значения суммы (1).

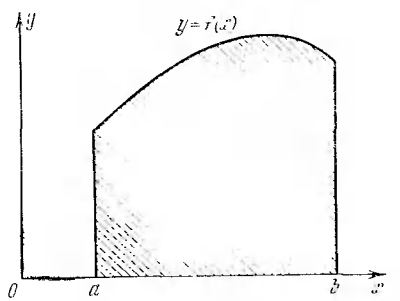

Рис. 31. Криволинейная трапеция, ограниченная графиком функции $f(x)$, осью x и двумя вертикальными прямыми $x = a$ и $x = b$.

Будем делать разбиение временного отрезка $[T_1, T_2]$ все более мелким. При этом отклонение движения от равномерного для каждого частичного временного отрезка $[t_{k-1}, t_k]$ должно уменьшаться. Поэтому естественно ожидать, что в пределе при неограниченном измельчении разбиения отрезка $[T_1, T_2]$ приближенное равенство (1) перейдет в точное. Мы получим формулу пройденного пути. Предел выражения, стоящего в правой части равенства (если он существует), называется *определенным интегралом* от функции $v(t)$ по отрезку $[T_1, T_2]$ и обозначается

$$s = \int_{T_1}^{T_2} v(t)\,dt. \qquad (2)$$

Рассмотрим еще одну задачу. Пусть материальная точка движется вдоль оси x под действием некоторой силы, величина которой зависит от положения точки: $F = F(x)$. Требуется подсчитать работу, совершенную при перемещении точки из положения $x = a$ в положение $x = b$. Эта задача с математической точки зрения совершенно аналогична предыдущей, и для ее решения можно использовать тот же подход.

Выберем на отрезке $[a, b]$ ряд промежуточных точек $a < x_1 < x_2 < \ldots < x_{n-1} < b$ и положим $x_0 = a$, $x_n = b$. В резуль-

тате исходный отрезок $[a, b]$ окажется разбитым на n отрезков $[x_{k-1}, x_k]$ $(k=1, 2, \ldots, n)$. Возьмем на каждом из них по произвольной точке $\xi_k \in [x_{k-1}, x_k]$ и подсчитаем величину силы в этой точке: $F_k = F(\xi_k)$. Если отрезок $[x_{k-1}, x_k]$

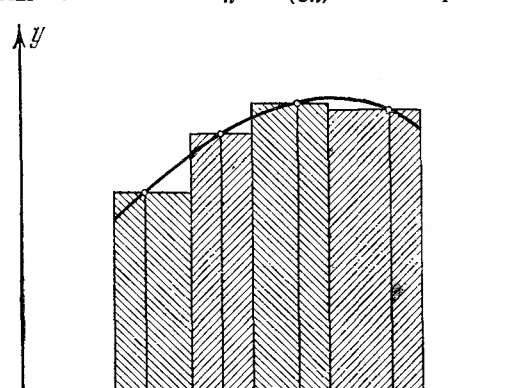

Рис. 32. Построение ступенчатого многоугольника, аппроксимирующего криволинейную трапецию.

Рис. 33. Улучшение аппроксимации криволинейной трапеции ступенчатым многоугольником при измельчении разбиения.

достаточно мал, то сила изменяется на нем мало и мы приближенно можем считать ее постоянной: $F(x) \approx F(\xi_k)$, $x \in [x_{k-1}, x_k]$. При этом для работы по перемещению матери-

альной точки из положения x_{k-1} в положение x_k получим

$$\Delta A_k \approx F(\xi_k)\,\Delta x_k, \qquad \Delta x_k = x_k - x_{k-1}.$$

Складывая эти величины, подсчитаем полную работу:

$$A \approx F(\xi_1)\,\Delta x_1 + F(\xi_2)\,\Delta x_2 + \ldots + F(\xi_n)\,\Delta x_n. \tag{3}$$

Эта формула аналогична формуле (1). Чем мельче разбиение, тем она точнее. В пределе при неограниченном измельчении разбиения она дает точное выражение для работы в виде интеграла от функции $F(x)$:

$$A = \int_a^b F(x)\,dx. \tag{4}$$

Совершенно аналогично можно подсчитать заряд Q, который переносится через поперечное сечение проводника током $I(t)$ за время $T_1 \leqslant t \leqslant T_2$:

$$Q = \int_{T_1}^{T_2} I(t)\,dt. \tag{5}$$

Рассмотрим последний пример. На рис. 31 показана фигура, ограниченная графиком некоторой функции $y = f(x)$, осью x и двумя вертикальными прямыми $x=a$ и $x=b$. Такую фигуру принято называть *криволинейной трапецией*. Требуется подсчитать ее площадь.

Для решения этой задачи, как и предыдущих, разобьем отрезок $[a, b]$ точками x_k ($k=0, 1, \ldots, n$, $x_0=a$, $x_n=b$) на частичные отрезки $[x_{k-1}, x_k]$. На каждом из них выберем произвольную точку $\xi_k \in [x_{k-1}, x_k]$ ($k=1, 2, \ldots, n$), вычислим в ней значение функции $f(x)$ и построим прямоугольник высотой $f(\xi_k)$ (рис. 32). В результате мы получим ступенчатый многоугольник, который можно рассматривать как некоторое приближение к более сложной фигуре — заданной криволинейной трапеции.

Легко подсчитать площадь этого многоугольника $\tilde{S}$ как сумму площадей отдельных прямоугольников:

$$\tilde{S} = f(\xi_1)\,\Delta x_1 + f(\xi_2)\,\Delta x_2 + \ldots + f(\xi_n)\,\Delta x_n. \tag{6}$$

При измельчении разбиения точность аппроксимации криволинейной трапеции ступенчатым многоугольником повышается (ср. рис. 32 и 33). Поэтому за площадь криволинейной трапеции естественно принять предел площадей ступенчатых многоугольников при неограниченном измель-

чении разбиения, т. е. соответствующий определенный интеграл:

$$S = \int_a^b f(x)\,dx. \tag{7}$$

Можно было бы предложить множество других задач, решение которых также записывается через определенный интеграл, но в этом нет необходимости: характер таких задач достаточно ясен из разобранных примеров.

§ 2. Понятие определенного интеграла

После того как мы рассмотрели на интуитивном уровне несколько задач, приводящих к идее определенного интеграла, обсудим это важнейшее понятие метематического анализа на уровне строгих математических определений.

Пусть на отрезке $[a, b]$ задана некоторая функция $f(x)$. Разобьем этот отрезок точками x_k: $a=x_0<x_1<\ldots<x_{n-1}<x_n=b$ на частичные отрезки $[x_{k-1}, x_k]$ $(k=1, 2, \ldots, n)$. Такую операцию условимся называть *разбиением* и обозначать символом $T(x_k)$. Вычислим длины полученных отрезков $\Delta x_k=x_k-x_{k-1}$ и положим $\Delta=\max\{\Delta x_k\}$. Величина Δ является длиной наибольшего отрезка при разбиении $T(x_k)$.

Выберем на каждом отрезке $[x_{k-1}, x_k]$ какую-нибудь точку $\xi_k \in [x_{k-1}, x_k]$ и вычислим в ней значение функции $f(x)$. Из найденных величин образуем сумму, которую называют *интегральной суммой*:

$$I(x_k, \xi_k) = f(\xi_1)\Delta x_1 + f(\xi_2)\Delta x_2 + \ldots + f(\xi_n)\Delta x_n. \tag{8}$$

Значение интегральной суммы определяется выбором точек разбиения x_k и точек ξ_k. Выражения (1), (3), (6) представляют собой интегральные суммы для соответствующих функций.

Нас будет интересовать поведение интегральных сумм при неограниченном измельчении разбиения, т. е. при $\Delta \to 0$. Сформулируем два определения.

Число $\mathcal{J}$ называется *пределом интегральных сумм* $I(x_k, \xi_k)$ при $\Delta \to 0$, если для всякой точности $\varepsilon>0$ можно указать такое δ, что для любого разбиения $T(x_k)$, удовлетворяющего условию $\Delta<\delta$, при произвольном выборе точек $\xi_k \in [x_{k-1}, x_k]$ выполняется неравенство

$$|\mathcal{J} - I(x_k, \xi_k)| < \varepsilon. \tag{9}$$

Если для функции $f(x)$, заданной на отрезке $[a, b]$, существует предел $\mathcal{J}$ интегральных сумм при $\Delta \to 0$, то она называется *интегрируемой* на отрезке $[a, b]$, а число $\mathcal{J}$ называется *определенным интегралом* от этой функции по отрезку $[a, b]$ и обозначается символом

$$\mathcal{J} = \int_a^b f(x)\,dx.$$

Рассмотрим в качестве примера функцию, равную постоянной на отрезке $[a, b]$: $f(x) = C$. Проведем какое-нибудь разбиение отрезка $T(x_k)$, выберем точки ξ_k и составим интегральную сумму

$$I(x_k, \xi_k) = C(\Delta x_1 + \Delta x_2 + \ldots + \Delta x_n) = C(b - a).$$

Мы видим, что она не зависит ни от разбиения, ни от выбора точек ξ_k. Следовательно, существует предел интегральных сумм при $\Delta \to 0$, равный той же величине:

$$\lim_{\Delta \to 0} I(x_k, \xi_k) = \int_a^b C\,dx = C(b - a).$$

Мы доказали, что функция, равная постоянной, интегрируема, и вычислили соответствующий интеграл.

Вопрос об условиях интегрируемости в своей общей постановке подробно рассматривается в курсе математического анализа, и мы на нем останавливаться не будем. Отметим лишь, что свойством интегрируемости обладает достаточно широкий класс функций, к которому, в частности, относятся функции, непрерывные на отрезке $[a, b]$; функции, имеющие на отрезке $[a, b]$ конечное число точек разрыва и ограниченные; функции, монотонные на отрезке $a, b]$.

§ 3. Формула Ньютона — Лейбница

Чтобы воспользоваться формулами (2), (4), (5), (7) для подсчета соответствующих величин, нужно уметь вычислять определенные интегралы. Один из способов их вычисления основан на *формуле Ньютона — Лейбница*:

$$\int_a^b f(x)\,dx = F(b) - F(a), \qquad (10)$$

где $F(x)$ — какая-нибудь первообразная подынтегральной функции $f(x)$ на отрезке $[a, b]$. Напомним, что функция $F(x)$ называется *первообразной* функции $f(x)$, если она дифференцируема и ее производная $F'(x)$ равна $f(x)$.

Формула Ньютона — Лейбница играет важную роль в математическом анализе, устанавливая связь задачи определенного интегрирования с задачей отыскания первообразной (неопределенного интегрирования). Она позволяет вычислять интегралы от элементарных функций, первообразные которых тоже являются элементарными функциями. Если задача о первообразной подынтегральной функции решена в явном виде, то вычисление определенного интеграла сводится к подсчету разности ее значений в граничных точках отрезка $[a, b]$. Однако существует много функций, первообразные которых не выражаются через элементарные функции. Для них описанный способ вычисления определенных интегралов неприменим.

Рассмотрим в качестве примера два интеграла:

$$\int_1^2 \frac{1}{x}\,dx \quad \text{и} \quad \int_1^2 \frac{e^{-x}}{x}\,dx.$$

В первом из них подынтегральная функция $f(x)=1/x$, ее первообразная $F(x)=\ln x$. Используя формулу Ньютона — Лейбница, получаем

$$\int_1^2 \frac{1}{x}\,dx = \ln 2 - \ln 1 = \ln 2. \tag{11}$$

Для вычисления второго интеграла данный метод неприменим, потому что первообразная функции $f(x)=e^{-x}/x$ не является элементарной функцией.

Отметим также, что формула Ньютона — Лейбница не позволяет вычислять интегралы от функций, которые задаются графиком или таблицей.

Все это позволяет сделать вывод, что формула Ньютона — Лейбница не дает общего, универсального метода нахождения определенного интеграла от произвольной функции $f(x)$ по ее значениям на отрезке $[a, b]$, она не является алгоритмом решения рассматриваемой задачи. Поэтому мы не будем останавливаться ни на обосновании формулы Ньютона — Лейбница, ни на обсуждении методов отыскания первообразных. Все эти вопросы подробно изучаются в курсе математического анализа. Наша цель — познакомить

с некоторыми универсальными вычислительными алгоритмами решения задачи определенного интегрирования. Такие алгоритмы позволяют подсчитывать интегралы непосредственно по значениям подынтегральной функции $f(x)$ и не зависят от способа ее задания. Соответствующие формулы обычно называют *формулами численного интегрирования* или *квадратурными формулами* (буквально — формулами вычисления площадей).

§ 4. Алгоритмы численного интегрирования

Наиболее просто к идее численного интегрирования можно подойти, принимая во внимание определение интеграла как предела сумм. Если взять какое-нибудь достаточно мелкое разбиение отрезка $[a, b]$ и построить для него интегральную сумму (8), то ее значение можно приближенно принять за значение соответствующего интеграла.

Пусть, например, отрезок $[a, b]$ разбит на n равных частей длины $h=(b-a)/n$ и в качестве точек ξ_k выбраны средние точки соответствующих отрезков: $\xi_k=a+h\,(k-1/2)$ $(k=1, 2, \ldots, n)$. В этом случае выражение для интегральной суммы примет вид

$$I_n=(f(\xi_1)+f(\xi_2)+\ldots+f(\xi_n))\frac{b-a}{n}.$$

Если функция $f(x)$ интегрируема на отрезке $[a, b]$, то

$$\lim_{n\to\infty} I_n=\mathcal{I}=\int_a^b f(x)\,dx. \tag{12}$$

Согласно (12) выражение для интеграла $\mathcal{I}$ можно записать в виде $\mathcal{I}=I_n+\alpha_n$, причем $\lim\limits_{n\to\infty}\alpha_n=0$. Пренебрегая величиной α_n, получим приближенную формулу для вычисления интеграла $\mathcal{I}$, которую обычно называют *формулой прямоугольников*:

$$\mathcal{I}\approx I_n=(f(\xi_1)+f(\xi_2)+\ldots+f(\xi_n))\frac{b-a}{n}. \tag{13}$$

Причина такого названия связана с геометрической интерпретацией этой формулы. Интеграл $\mathcal{I}$ — это площадь криволинейной трапеции, а интегральная сумма, которой мы приближенно аппроксимируем интеграл,— площадь фигуры, составленной из прямоугольников (рис. 34). Все прямоугольники имеют одинаковое основание $h=(b-a)/n$,

а их высоты определяются значениями функции $f(x)$ в средних точках ξ_k отрезков разбиения. Разность между площадями этих двух фигур равна α_n, с возрастанием n она стремится к нулю.

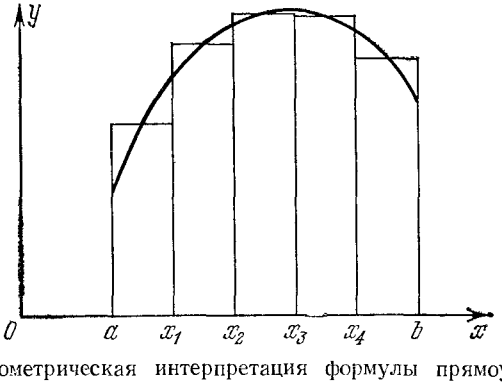

Рис. 34. Геометрическая интерпретация формулы прямоугольников.

Предположим, что функция $f(x)$ имеет на отрезке $[a, b]$ непрерывную вторую производную. В этом случае справедливо следующее утверждение: на отрезке $[a, b]$ существует такая точка x_n^*, что погрешность формулы (13) можно записать в виде

$$\alpha_n = \frac{(b-a)^3}{24n^2} f''(x_n^*). \tag{14}$$

Важная особенность данной формулы состоит в том, что нам гарантировано существование соответствующей точки x_n^*, но ничего не известно о ее положении. Поэтому формула (14) не позволяет вычислить α_n, но дает возможность ее оценить:

$$|\alpha_n| \leqslant \frac{(b-a)^3}{24n^2} \max |f''(x)|. \tag{15}$$

Это неравенство показывает, что с возрастанием n погрешность в формуле (13) убывает не медленнее, чем $1/n^2$.

Вывод формулы прямоугольников основан на замене интеграла интегральной суммой (13). Наряду с этим для построения формул численного интегрирования можно использовать другой подход: построить вспомогательную функцию, близкую к подынтегральной функции $f(x)$, и приближенно заменить интеграл от функции $f(x)$ интегралом от вспомогательной функции.

Рассмотрим простейшую реализацию этой идеи. Пусть отрезок $[a, b]$ разбит на n равных частей длины $h=(b-a)/n$ точками $x_k=a+kh$ ($k=0, 1, \ldots, n$, $x_0=a$, $x_n=b$). Построим функцию $g_n(x)$, определенную следующим образом. На каждом из отрезков $[x_{k-1}, x_k]$ она является линейной, а в граничных точках x_{k-1}, x_k принимает те же значения, что и функция $f(x)$: $f(x_{k-1})$ и $f(x_k)$. Аналитически эта функция задается с помощью формул

$$g_n(x)=f(x_{k-1})+\frac{f(x_k)-f(x_{k-1})}{h}(x-x_{k-1}), \qquad (16)$$

$$x\in[x_{k-1}, x_k], \qquad k=1, 2, \ldots, n.$$

Ее график представляет собой ломаную линию, начальная, конечная и угловые точки которой принадлежат также графику функции $f(x)$ (рис. 35). С увеличением n число общих точек растет и ломаная $y=g_{n_k}(x)$ приближается к линии $y=f(x)$.

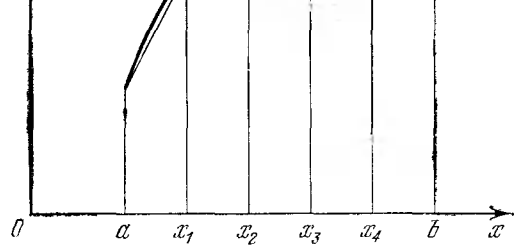

Рис. 35. Геометрическая интерпретация формулы трапеций.

Вычислим с помощью формулы Ньютона — Лейбница интеграл от функции $g_n(x)$ по отрезку $[x_{k-1}, x_k]$:

$$\int_{x_{k-1}}^{x_k} g_n(x)\,dx=$$

$$=f(x_{k-1})\int_{x_{k-1}}^{x_k}dx+\frac{f(x_k)-f(x_{k-1})}{h}\int_{x_{k-1}}^{x_k}(x-x_{k-1})\,dx=$$

$$=f(x_{k-1})\,h+\frac{1}{2}(f(x_k)-f(x_{k-1}))\,h=\frac{1}{2}(f(x_{k-1})+f(x_k))\,h. \qquad (17)$$

Полученный ответ имеет простой геометрический смысл. Интеграл равен площади фигуры, ограниченной графиком функции $g_n(x)$ (16), осью x и вертикальными линиями $x=x_{k-1}$, $x=x_k$. В данном случае эта фигура является трапецией (см. рис. 35), и ее площадь равна произведению полусуммы оснований на высоту. Однако мы получили этот результат прямым интегрированием, а не ссылкой на формулы геометрии.

Подсчитаем теперь интеграл от функции $g_n(x)$ по всему отрезку $[a, b]$:

$$T_n = \int_a^b g_n(x)\,dx = \int_a^{x_1} g_n(x)\,dx + \int_{x_1}^{x_2} g_n(x)\,dx + \ldots + \int_{x_{n-1}}^{b} g_n(x)\,dx.$$

Подставляя вместо интегралов по отдельным отрезкам $[x_{k-1}, x_k]$ их значения (17), получим

$$T_n = \left(\frac{1}{2}f(a) + f(x_1) + f(x_2) + \ldots + f(x_{n-1}) + \frac{1}{2}f(b)\right)\frac{b-a}{n}.$$

Полученное выражение можно представить в виде

$$T_n = \left(f(x_1) + f(x_2) + \ldots + f(x_{n-1}) + f(b)\right)\frac{b-a}{n} + (f(a) - f(b))\frac{b-a}{2n}.$$

Первое слагаемое в этой формуле представляет собой интегральную сумму для функции $f(x)$ в случае, когда в качестве точек ξ_k на отрезках $[x_{k-1}, x_k]$ выбираются правые концы этих отрезков: $\xi_k = x_k$ ($k=1, 2, \ldots, n$). При $n \to \infty$ интегральная сумма стремится к интегралу $\mathcal{I} = \int_a^b f(x)\,dx$.

Второе слагаемое при $n \to \infty$ стремится к нулю. Таким образом,

$$\lim_{n\to\infty} T_n = \lim_{n\to\infty} \int_a^b g_n(x)\,dx = \mathcal{I} = \int_a^b f(x)\,dx. \qquad (18)$$

Согласно этому выражение для интеграла $\mathcal{I}$ можно записать в виде $\mathcal{I} = T_n + \beta_n$, причем $\lim_{n\to\infty} \beta_n = 0$. Прене-

брегая величиной β_n, получим приближенную формулу для вычисления интеграла $\mathcal{J}$, которую обычно называют *формулой трапеций*:

$$\mathcal{J} \approx T_n = \left(\frac{1}{2} f(a) + f(x_1) + \ldots + f(x_{n-1}) + \frac{1}{2} f(b)\right) \frac{b-a}{n}. \quad (19)$$

Причина такого названия, как и в предыдущем случае, связана с геометрической интерпретацией формулы. Она приближенно представляет площадь криволинейной трапеции, соответствующей интегралу $\mathcal{J}$, в виде суммы площадей обычных трапеций, которые образуются графиком функции $g_n(x)$ (см. рис. 35). Разность между площадями этих двух фигур равна β_n, с возрастанием n она стремится к нулю.

Предположим, как и при анализе формулы прямоугольников (13), что функция $f(x)$ имеет на отрезке $[a, b]$ непрерывную вторую производную. В этом случае справедливо следующее утверждение: на отрезке $[a, b]$ существует такая точка x_n^{**}, что погрешность формулы (19) можно записать в виде

$$\beta_n = -\frac{(b-a)^3}{12n^2} f''(x_n^{**}). \quad (20)$$

Эта формула совершенно аналогична (14). Она не позволяет вычислить величину β_n, поскольку неизвестно положение точки x_n^{**}, но дает возможность ее оценить:

$$|\beta_n| \leqslant \frac{(b-a)^3}{12n^2} \max |f''(x)|. \quad (21)$$

Данная оценка отличается от оценки погрешности формулы прямоугольников (15) только числовым множителем. Таким образом, формулы прямоугольников и трапеций характеризуются примерно одинаковой точностью.

Идею, которая была использована при построении формулы трапеций, можно развивать дальше и использовать для получения более точных формул численного интегрирования. Например, если воспользоваться для аппроксимации подынтегральной функции на отдельных отрезках $[x_{k-1}, x_k]$ не линейной функцией $g_n(x)$, а полиномом второй степени, то мы придем к *формуле Симпсона*:

$$\mathcal{J} = \int_a^b f(x)\,dx \approx S_n =$$
$$= (f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) +$$
$$+ 4f(x_{n-1}) + f(b)) \frac{b-a}{3n}. \quad (22)$$

В этой формуле, как и в формуле трапеций, предполагается, что отрезок $[a, b]$ разбит на n равных частей длины $h=(b-a)/n$ точками $x_k=a+kh$ $(k=0, 1, \ldots, n)$, причем число n должно быть обязательно четным. Значение функции $f(x)$ в нечетных точках разбиения $x_1, x_3, \ldots, x_{n-1}$ входит с коэффициентом 4, в четных $x_2, x_4, \ldots, x_{n-2}$ — с коэффициентом 2 и в двух граничных $x_0=a$, $x_n=b$ — с коэффициентом 1.

Вывод формулы Симпсона не содержит новых идей по сравнению с выводом формулы трапеций, но является более громоздким. Мы на нем останавливаться не будем.

Обозначим через γ_n погрешность формулы Симпсона. Предположим, что функция $f(x)$ имеет на отрезке $[a, b]$ непрерывную четвертую производную; тогда на данном отрезке найдется такая точка $\tilde{x}_n$, что величину γ_n можно записать в виде

$$\gamma_n = -\frac{(b-a)^5}{180n^4} f^{(4)}(\tilde{x}_n). \tag{23}$$

Отсюда получаем оценку

$$|\gamma_n| \leqslant \frac{(b-a)^5}{180n^4} \max |f^{(4)}(x)|. \tag{24}$$

Отметим, что с возрастанием n погрешность убывает как $1/n^4$, т. е. быстрее, чем в формулах прямоугольников и трапеций.

В заключение для иллюстрации изложенного материала снова обратимся к интегралам, рассмотренным в § 3. Первый из них: $\int_1^2 \frac{1}{x}\,dx$, был вычислен по формуле Ньютона — Лейбница (10). Подсчитаем его теперь по формулам прямоугольников (13), трапеций (19) и Симпсона (22), выбирая $n=10$.

В табл. 1 приведены значения подынтегральной функции $f(x)=1/x$ в точках $\xi_k=1+h$ $(k-1/2)$, $h=(b-a)/n=0{,}1$ $(k=1, 2, \ldots, 10)$, используемые в формуле прямоугольников. Складывая значения функции, приведенные в третьем столбце таблицы, и умножая сумму на $h=0{,}1$, получим приближенное значение интеграла по формуле прямоугольников:

$$\mathcal{J} \approx I_{10} = 0{,}692\,835\,360.$$

В табл. 2 приведены значения подынтегральной функции $f(x)=1/x$ в точках $x_k=1+kh$ $(k=0, 1, \ldots, n)$, используемые в формулах трапеций и Симпсона. Сложим значения функ-

Таблица 1

k	ξ_k	$f(\xi_k)$
1	1,05	0,952380952
2	1,15	0,869565217
3	1,25	0,800000000
4	1,35	0,740740740
5	1,45	0,689655172
6	1,55	0,645161290
7	1,65	0,606060606
8	1,75	0,571428571
9	1,85	0,540540540
10	1,95	0,512820512

Таблица 2

k	x_k	$f(x_k)$
0	1,0	1,000000000
1	1,1	0,909090909
2	1,2	0,833333333
3	1,3	0,769230769
4	1,4	0,714285714
5	1,5	0,666666666
6	1,6	0,625000000
7	1,7	0,588235294
8	1,8	0,555555555
9	1,9	0,526315789
10	2,0	0,500000000

ции в третьем столбце, умножив предварительно нулевую и десятую строчки на 1/2, и умножим полученную сумму на $h=0{,}1$. В результате получим значение интеграла по формуле трапеций:

$$\mathcal{I} \approx T_{10} = 0{,}693\,771\,403.$$

Наконец, получим значение интеграла по формуле Симпсона:

$$\mathcal{I} \approx S_{10} = 0{,}693\,150\,230.$$

В данном случае мы знаем точное значение интеграла:

$$\mathcal{I} = \ln 2 = 0{,}693\,147\,180.$$

Подсчитаем погрешности, которые дают все три формулы:

$$\alpha_{10} = \mathcal{I} - I_{10} = 0{,}000\,311\,820,$$
$$\beta_{10} = \mathcal{I} - T_{10} = -0{,}000\,624\,223,$$
$$\gamma_{10} = \mathcal{I} - S_{10} = -0{,}000\,003\,050.$$

Как и следовало ожидать, погрешность формулы Симпсона является наименьшей.

Вычислим производные подынтегральной функции $f(x)=1/x$:

$$f'(x) = -1/x^2, \quad f''(x) = 2/x^3, \quad f'''(x) = -6/x^4, \quad f^{(4)}(x) = 24/x^5.$$

На отрезке $[1, 2]$ $f''(x)>0$, $f^{(4)}(x)>0$, причем их наибольшие значения равны соответственно 2 и 24. Найденные погрешности имеют правильные знаки согласно формулам (14),

(20), (23), а их величины удовлетворяют неравенствам (15), (21), (24):

$$|\alpha_{10}| \leqslant \frac{2}{24 \cdot 100} = 0{,}000\,833\,333,$$

$$|\beta_{10}| \leqslant \frac{12}{12 \cdot 100} = 0{,}001\,666\,666,$$

$$|\gamma_{10}| \leqslant \frac{24}{180 \cdot 10\,000} = 0{,}000\,013\,333.$$

Рассмотрим теперь второй интеграл: $\mathcal{I} = \int_1^2 \frac{e^{-x}}{x}\,dx$. Его нельзя вычислить по формуле Ньютона — Лейбница, но легко подсчитать с помощью формул численного интегрирования.

Таблица 3

k	x_k	$f(x_k)$	$c_k f(x_k)$
0	1,0	0,367879441	0,367879441
1	1,1	0,302610076	1,210440304
2	1,2	0,250995176	0,501990352
3	1,3	0,209639841	0,838559364
4	1,4	0,176140688	0,352281376
5	1,5	0,148753440	0,595013760
6	1,6	0,126185324	0,252370648
7	1,7	0,107460896	0,429843584
8	1,8	0,091832716	0,183665432
9	1,9	0,078720326	0,314881304
10	2,0	0,067667642	0,067667642

Воспользуемся для расчета наиболее точной из трех формул — формулой Симпсона, выбирая, как и в предыдущем случае, $n=10$. В табл. 3 в третьем столбце приведены значения подынтегральной функции $f(x)=e^{-x}/x$ в точках $x_k=1+kh$, $h=0{,}1$ ($k=0, 1, \ldots, 10$), а в четвертом они умножены на коэффициент 1, 2 или 4 в зависимости от номера k в соответствии с формулой Симпсона. Складывая значения функции, приведенные в четвертом столбце таблицы, и умножая сумму на $h/3=1/30$, получим

$$\mathcal{I} \approx S_{10} = 0{,}170\,486\,440.$$

Вычислим четвертую производную функции $f(x)=e^{-x}/x$:

$$f^{(4)}(x)=e^{-x}\left(\frac{1}{x}+\frac{4}{x^2}+\frac{12}{x^3}+\frac{24}{x^4}+\frac{24}{x^5}\right).$$

На отрезке $[1, 2]$ $f^{(4)}(x)>0$, $\max|f^{(4)}(x)|=65e^{-1}<24$. Таким образом, S_{10} дает значение интеграла $\mathcal{J}$ с избытком ($\gamma_{10}<0$), а для величины ошибки справедлива оценка

$$|\gamma_{10}|<\frac{24}{180\cdot 10\,000}=0{,}000\,013\,333.$$

Попробуйте сами вычислить значения рассматриваемого интеграла по формуле трапеций, используя данные третьего столбца табл. 3. (Не забудьте умножить строки $k=0$ и $k=10$ на 1/2.) Сравните полученные результаты. Найдите вторую производную функции $f(x)=e^{-x}/x$, определите по ней знак и оцените величину погрешности формулы трапеций.

В заключение отметим следующее. Оценка погрешности формул численного интегрирования с помощью неравенств типа (15), (21), (24) часто оказывается малоэффективной из-за трудностей, связанных с оценкой производных подынтегральной функции $f(x)$. Поэтому на практике для вычисления погрешности обычно пользуются следующим приемом. Выбирают число n, кратное 4, находят значения интеграла $\mathcal{J}$ по формуле Симпсона с числом точек n и $n/2$: S_n и $S_{n/2}$ ($n/2$ — целое четное число) и приближенно определяют погрешность численного интегрирования с помощью соотношения

$$\gamma_n=\mathcal{J}-S_n\approx\frac{1}{15}(S_n-S_{n/2}).$$

§ 5. Построение первообразной с помощью численного интегрирования

Формула Ньютона — Лейбница позволяет выразить значение определенного интеграла от функции $f(x)$ через ее первообразную $F(x)$. В математическом анализе устанавливается и прямо противоположная возможность: первообразная функции $f(x)$, непрерывной на отрезке $[a, b]$, может быть записана в виде определенного интеграла с переменным верхним пределом:

$$F(x)=\int_{x_0}^{x}f(t)\,dt. \tag{25}$$

Здесь x_0, x — две точки отрезка $[a, b]$, причем нижний предел интегрирования x_0 предполагается фиксированным, а верхний x — переменным. В случае непрерывной функции $f(x)$ функция $F(x)$, определенная с помощью (25), является дифференцируемой и ее производная равна $f(x)$:

$$F'(x) = \frac{d}{dx}\left(\int_{x_0}^{x} f(t)\,dt\right) = f(x). \tag{26}$$

Формула (25) в сочетании с какой-нибудь формулой численного интегрирования, например Симпсона, представляет собой универсальный алгоритм построения первообразной. Приведем два примера, иллюстрирующих этот алгоритм.

Функция $f(x) = \sin x/x$ непрерывна и, следовательно, имеет первообразные. Они не могут быть выражены через элементарные функции, но представление в виде интеграла с переменным верхним пределом для них справедливо. Одну из первообразных мы получим, выбирая нижний предел интегрирования $x_0 = 0$. Ее называют *интегральным синусом* и обозначают

$$\operatorname{Si} x = \int_0^x \frac{\sin t}{t}\,dt.$$

Интегральный синус определен на всей числовой прямой, является нечетной функцией x, имеет конечные предельные значения на бесконечности:

$$\lim_{x \to \pm\infty} \operatorname{Si} x = \pm\pi/2.$$

Согласно (26)

$$(\operatorname{Si} x)' = \sin x/x.$$

По знаку производной легко определить области возрастания и убывания функции, разделенные точками экстремума $x_k = k\pi$ ($k = \pm 1, \pm 2, \ldots$). Методы численного интегрирования позволяют вычислить значение $\operatorname{Si} x$ при любом x. График интегрального синуса при $x \geqslant 0$ приведен на рис. 36.

В качестве второго примера рассмотрим *функцию ошибок* $\operatorname{erf} x$, играющую важную роль в теории вероятностей *). Подобно интегральному синусу, она вводится в виде инте-

*) Обозначение образовано с помощью первых букв английского названия функции ошибок — error function.

грала с переменным верхним пределом от функции e^{-x^2}, которая не имеет первообразных в классе элементарных функций:

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Функция ошибок определена на всей числовой прямой является нечетной функцией x, имеет конечные предельные значения на бесконечности:

$$\lim_{x \to \pm\infty} \operatorname{erf} x = \pm 1.$$

Согласно (26)

$$(\operatorname{erf} x)' = \frac{2}{\sqrt{\pi}} e^{-x^2}.$$

Производная всюду положительна, следовательно, функция ошибок монотонно возрастает. Ее график приведен на рис. 37.

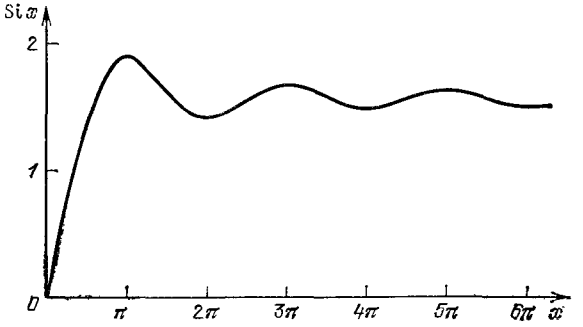

Рис. 36. График интегрального синуса.

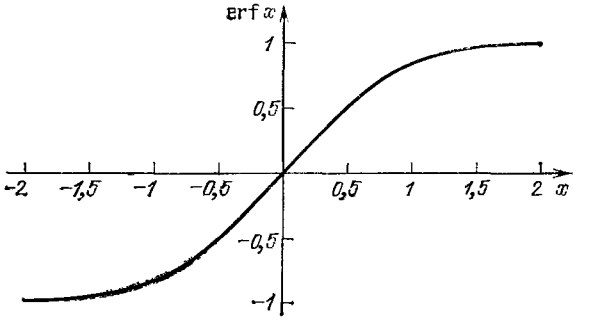

Рис. 37. График функции ошибок.

Существует ряд других специальных функций, которые вводятся как интегралы с переменным верхним пределом. Не будем останавливаться на их описании. Отметим лишь, что разобранные примеры показывают, насколько условно деление функции на элементарные и не элементарные. По существу, чтобы работать с какой-то функцией, нужно знать ее свойства и иметь алгоритм вычисления при любом значении аргумента. С этой точки зрения применение интегрального синуса или функции ошибок ничем не отличается от применения более привычных элементарных функций.

ЗАКЛЮЧЕНИЕ

Создатели первых ЭВМ ставили перед собой вполне определенную цель: с помощью полной автоматизации упростить и ускорить процесс вычислений. Эта задача была решена на основе принципа программного управления. Однако включение в систему команд наряду с арифметическими операциями операций передачи управления, хранение программы в виде чисел в оперативной памяти машины и связанная с этим возможность изменять программу в ходе вычислений позволили ЭВМ перешагнуть первоначальные рамки быстродействующего вычислительного автомата и превратили их в мощный, гибкий инструмент, широко применяемый в самых различных областях человеческой деятельности.

Мы уже говорили в главе 1 о математических моделях как о методе математического описания сложных процессов, систем, явлений, конструкций. ЭВМ, благодаря своему огромному быстродействию и логическим возможностям, позволяют провести всесторонний анализ этих моделей и получить детальную количественную информацию о свойствах изучаемого объекта. Данный метод исследования часто называют *вычислительным экспериментом*. Он стал в настоящее время одним из наиболее эффективных и универсальных способов познания законов реального мира и их использования в практической деятельности людей.

Вычислительный эксперимент имеет целый ряд преимуществ перед экспериментом реальным. Он значительно дешевле и доступнее. Во многих случаях вычислительный эксперимент позволяет глубже понять результаты реального эксперимента, сопоставить их с теорией. Часто вычислительный эксперимент проводится для планирования будущих экспериментов и прогнозирования их результатов, для проектирования экспериментальных установок следующего поколения и определения оптимальных режимов их работы. Вычислительный эксперимент совершенно незаменим при изучении таких сложных объектов, как космос,

человеческое общество, где постановка большого числа натуральных экспериментов либо затруднена, либо вообще невозможна.

Подчеркивая достоинства вычислительного эксперимента, нужно в то же время отметить его ограниченность. Мы знаем, что математическая модель всегда является упрощенным описанием реального объекта, что полученные с ее помощью результаты носят для изучаемого объекта приближенный характер. Установить пределы применимости математической модели, степень ее соответствия объекту можно только с помощью настоящего эксперимента (критерий практики). Вот почему, какими бы совершенными ЭВМ мы ни обладали, вычислительный эксперимент никогда не вытеснит обычный эксперимент. Будущее — за их разумным, гармоничным сочетанием.

ЭВМ — не только техническая база вычислительного эксперимента, одновременно они являются важным элементом реальных экспериментов. Высокий уровень автоматизации многих естественнонаучных экспериментов и способов регистрации их результатов позволяет получить в короткий срок весьма большой объем информации: десятки и сотни тысяч снимков, осциллограмм, показаний детекторов и т. д. Для интерпретации этой информации требуется сложная математическая обработка. Во многих случаях такую обработку необходимо проводить практически одновременно с экспериментом. Сделать это можно только с помощью ЭВМ.

Включение ЭВМ в общий экспериментальный комплекс потребовало создания эффективных численных методов решения математических задач, возникающих при обработке и интерпретации результатов эксперимента, и разработки реализующих эти методы программ. Пришлось также решать сложные технические проблемы сопряжения измерительной аппаратуры с устройствами ввода ЭВМ. Для полной автоматизации обработки важно, чтобы преобразование сигналов любой природы в систему чисел, понятную ЭВМ, осуществлялось автоматически, без участия человека.

Применение ЭВМ позволяет не только обрабатывать эксперимент, но с помощью системы обратных связей управлять им: поддерживать нужные значения параметров, определяющих его условия, менять параметры по заданному закону, вести поиск оптимальных режимов протекания процессов.

Очень быстро были оценены логические возможности ЭВМ, на которые стали обращать особое внимание при проектировании новых машин. ЭВМ стали применяться для логического анализа сложных объектов и ситуаций, для решения задач связанных с получением выводов из некоторой системы исходных предпосылок. Это стимулировало развитие математической логики. Характерным примером решения сложной логической задачи на ЭВМ является создание трансляторов для перевода программ с алгоритмического на машинный язык. Логические возможности ЭВМ позволили создавать программы для игры в шахматы, перевода с одного языка на другой, стихосложения и т. д. Возникли дискуссии, может ли машина «мыслить», проводилось обсуждение вопроса, что такое интеллект, творческая деятельность. Споры часто носили схоластический характер, потому что спорящие стороны вкладывали в одни и те же слова разный смысл. Однако они стимулировали изучение ЭВМ как мощного средства повышения интеллектуальных возможностей человека. То, что человек, вооруженный ЭВМ, интеллектуально сильнее человека без вычислительной машины, то, что человек с помощью ЭВМ сумел решить существенно более широкий круг задач, чем до их появления, сомнений не вызывало и не оспаривалось.

ЭВМ стимулировали развитие кибернетики — науки об общих законах управления. Машины начали широко применяться для управления сложными системами самой различной природы: производственными процессами, полетом космических ракет, естественнонаучными экспериментами и т. д.

На ЭВМ возложили диспетчерские функции в задачах, связанных с переработкой большого объема информации. Вычислительные машины широко используются для оптимальной организации перевозок (см. гл. 6), в информационно-поисковых системах по продаже авиационных и железнодорожных билетов, в системе библиотечного поиска.

Широкое применение получили ЭВМ в задачах проектирования и конструирования объектов самой различной природы: инженерно-строительных сооружений, самолетов и космических аппаратов, нефтяных промыслов, радиоэлектронной аппаратуры. Они не только ускоряют разработку проекта, повышают его качество и тем самым дают большой экономический эффект, но и решают задачи проектирования, анализ которых без ЭВМ оказался бы невозможен.

В качестве характерного примера можно привести проектирование самих ЭВМ, которое в настоящее время нельзя осуществить без применения ЭВМ. Рост логической сложности, внедрение новой элементной базы привели к тому, что даже целая группа высококвалифицированных специалистов не может удержать в памяти огромное количество связей внутри проектируемой машины. Поэтому работу по проведению связей поручают специальным программам трассировки, которые способны запомнить все существующие связи и по определенным правилам проводить новые. Когда проект готов, он проверяется с помощью моделирования на ЭВМ. Если проверка устанавливает, что проект работоспособен, то машина готовит необходимую техническую документацию для производства отдельных плат, узлов и связей между ними. Эта производственная документация представляет собой перфорированную ленту, которая вводится в программно-управляющее устройство, обеспечивающее технологический цикл производства новой вычислительной машины. В противном случае ЭВМ-«родитель» сообщает проектировщику об имеющемся дефекте. Проектировщик должен понять, в чем состоит ошибка задания, и исправить ее. Так диалог между проектировщиком и ЭВМ создает объединение удивительных творческих способностей человека с педантичностью, быстродействием и прекрасной памятью машины. Этим примером мы и закончим обсуждение темы, поскольку дальнейшее перечисление без анализа деталей и подробностей не сможет добавить существенно новой информации.

Широта и разнообразие областей применения ЭВМ, существенный прогресс, которого удается добиться всюду, где они начинают использоваться, существование проблем и проектов, которые вообще не могли бы развиваться без ЭВМ,— все это делает вычислительные машины одним из определяющих факторов научно-технического прогресса нашего времени.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Андрей Николаевич Тихонов
Дмитрий Павлович Костомаров

ВВОДНЫЕ ЛЕКЦИИ ПО ПРИКЛАДНОЙ МАТЕМАТИКЕ

Редактор *И. В. Викторенкова*
Техн. редактор *С. Я. Шкляр*
Корректоры *О. А. Бутусова, И. Я. Кришталь*

ИБ № 12306

Сдано в набор 26.03.84. Подписано к печати 91. 07. 84. Т-14880. Формат 84×108$^{1}/_{32}$. Бумага тип. № 2. Литературная гарнитура. Высокая печать. Усл. печ. л. 10,08. Усл. кр.-отт. 10,08. Уч.-изд. л. 10,13. Тираж 55 000 экз. Заказ № 2855/293. Цена 65 коп.

Издательство «Наука»
Главная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

Набрано в ордена Октябрьской Революции и ордена Трудового Красного Знамени Первой Образцовой типографии имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 113054 Москва М-54, Валовая, 28
Отпечатано в Подольском филиале ПО «Периодика» ул. Кирова, д. 25.