invention of probability theory, and its applied wing, statistics, made the same point about the *irregularities* of nature. Remarkably, there are numerical patterns in chance events. But these patterns show up only in statistical quantities such as long-term trends and averages. They make predictions, but these are about the likelihood of some event happening or not happening. They do not predict when it will happen. Despite that, probability is now one of the most widely used mathematical techniques, employed in science and medicine to ensure that any deductions made from observations are significant, rather than apparent patterns resulting from chance associations.

CHAPTER 19

Number Crunching

Calculating machines and computational mathematics

Mathematicians have always dreamed of building machines to reduce the drudgery of routine calculations. The less time you spend calculating, the more time you can spend thinking. From prehistoric times sticks and pebbles were used as an aid to counting, and piles of pebbles eventually led to the abacus, in which beads slide along rods to represent the digits of numbers. Especially as perfected by the Japanese, the abacus could carry out basic arithmetic rapidly and accurately in the hands of an expert. Around 1950 a Japanese abacus outperformed a mechanical hand-calculator.

A dream comes true?

By the 21st century, the advent of electronic computers and the widespread availability of integrated circuits (chips) had given machines a massive advantage. They were far faster than the human brain or a mechanical device – billions or trillions of arithmetical operations every second are now commonplace. The fastest as I write,

IBM's Blue Gene/L, can perform one quadrillion calculations (floating-point operations) per second. Today's computers also have vast memory, storing the equivalent of hundreds of books ready for almost instant recall. Colour graphics have reached a pinnacle of quality.

The rise of the computer

Earlier machines were more modest, but they still saved a lot of time and effort. The first development after the abacus was probably Napier's bones, or Napier's rods, a system of marked rods that Napier invented before he came up with logarithms. Essentially they were the universal components of traditional long multiplication. The rods could be used in place of pencil and paper, saving time writing numerals, but they mimicked hand calculations.

In 1642 Pascal invented the first genuinely mechanical calculator, the Arithmetic Machine, to help his father with his accounts. It could perform addition and subtraction, but not multiplication or division. It had eight rotating dials, so it effectively worked with eight-digit numbers. In the succeeding decade, Pascal built fifty similar machines, most of which are preserved in museums to this day.

In 1671 Leibniz designed a machine for multiplication, and built one in 1694, remarking that 'It is unworthy of excellent men to lose hours like slaves in the labour of calculation, which could be safely relegated to anyone else if machines were used.' He named his machine the Staffelwalze (step reckoner). His main idea was widely used by his successors.

One of the most ambitious proposals for a calculating machine was made by Charles Babbage. In 1812 he said he 'was sitting in the rooms of the Analytical Society, at Cambridge, my head leaning forward on the table in a kind of dreamy mood, with a table of logarithms lying open before me. Another member, coming into the room, and seeing me half asleep, called out, "Well, Babbage, what are you dreaming about?" to which I replied "I am thinking that all these tables" (pointing to the logarithms) "might be calculated by machinery".'

Babbage pursued this dream for the rest of his life, constructing a prototype called the difference engine. He sought government funding for more elaborate machines. His most ambitious project, the analytical engine, was in effect a programmable mechanical computer. None of these machines was built, although various components were made. A modern reconstruction of the difference engine is in the Science Museum in London – and it works. Augusta Ada King, Countess of Lovelace, contributed to Babbage's work by developing some of the first computer programs ever written.

Augusta Ada King

1815 – 52

Augusta Ada was the daughter of the poet Lord Byron and Anne Milbanke. Her parents separated a month after her birth and she never saw her father again. The child showed mathematical ability, and unusually Lady Byron thought that mathematics was good training for the mind and encouraged her daughter to study it. In 1833 Ada met Charles Babbage at a party, and soon after she was shown his prototype difference engine, finding it fascinating and rapidly understanding how it worked. She became Countess of Lovelace when her husband William was created an Earl in 1838.

In her 1843 translation of Luigi Menabrea's *Notions sur la Machine Analytique de Charles Babbage* she added what are in effect sample programs of her own devising. She wrote that 'The distinctive characteristic of the Analytical Engine ... is the introduction into it of the principle which Jacquard devised for regulating, by means of punched cards, the most complicated patterns in the fabrication of brocaded stuffs ... We may say most aptly that the Analytical Engine weaves algebraical patterns just as the Jacquard loom weaves flowers and leaves.'

At the age of 36 she developed uterine cancer, and died after a lengthy period of severe pain, bled to death by her physicians.

The first mass-produced calculator, the Arithmometer, was manufactured by Thomas de Colmar in 1820. It employed a stepped drum mechanism and was still in production in 1920. The next major step was the pin wheel mechanism of the Swedish inventor Willgodt T. Odhner. His calculator set the pattern for dozens, if not hundreds of similar machines, made by a variety of manufacturers. The motive power was supplied by the operator, who turned a handle to revolve a series of discs on which the digits 0–9 were displayed. With practice, complicated calculations could be carried out at high speed. The scientific and engineering calculations for the Manhattan project of the Second World War, to make the first atomic bomb, were performed using such machines by a squad of 'calculators' – mainly young women. The advent of cheap powerful electronic computers in the 1980s made mechanical calculators obsolete, but their use in business and scientific computing was widespread until that time.

Calculating machines contribute more than just simple arithmetic, because many scientific calculations can be implemented numerically as lengthy series of arithmetical operations. One of the earliest numerical methods, which solves equations to arbitrarily high precision, was invented by Newton, and is appropriately known as *Newton's method*. It solves an equation $f(x) = 0$ by calculating a series of successive approximations to a solution, each improving on the previous one but based on it. From some initial guess, x_1, improved approximations $x_2, x_3, \ldots, x_n, x_{n+1}$, are derived using the formula

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

where f' is the derivative of f. The method is based on the geometry of the curve $y = f(x)$ near the solution. The point x_{n+1} is where the tangent to the curve at x_n crosses the x-axis. As the diagram shows, this is closer to x than the original point.

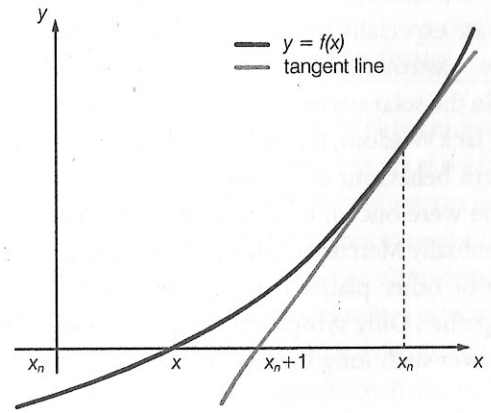
A second important application of numerical methods is to differential equations. Suppose we wish to solve the differential equation

$$\frac{dx}{dt} = f(x)$$

given that $x = x_0$ at time $t = 0$. The simplest method, due to Euler, is to approximate $\frac{dx}{dt}$ by $\frac{x(t+\varepsilon) - x(t)}{\varepsilon}$, where ε is very small. Then an approximation to the differential equation takes the form

$$x(t + \varepsilon) = x(t) + \varepsilon f(x(t))$$

Starting with $x(0) = x_0$, we successively deduce the values of $f(\varepsilon)$, $f(2\varepsilon)$, $f(3\varepsilon)$ and, in general, $f(n\varepsilon)$ for any integer $n > 0$. A typical value for ε might be 10^{-6}, say. A million iterations of the formula tells us $x(1)$, another million leads to $x(2)$ and so on. With today's computers a million calculations are trivial, and this becomes an entirely practical method.



Newton's method for solving an equation numerically

However, the Euler method is too simple-minded to be fully satisfactory, and numerous improvements have been devised. The best known are a whole class of Runge–Kutta methods, named after the German mathematicians Carl Runge and Martin Kutta, who invented their first such method in 1901. One of these, the so-called *fourth order* Runge–Kutta method, is very widely used in engineering, science and theoretical mathematics.

The needs of modern nonlinear dynamics have spawned several sophisticated methods that avoid accumulating errors over long periods of time by preserving certain structure associated with the exact solution. For example, in a mechanical system without friction, the total energy is conserved. It is possible to set up the numerical method so that at each step energy is conserved *exactly*. This procedure avoids the possibility that the computed solution will slowly drift away from the exact one, like a pendulum that is slowly coming to rest as it loses energy.

More sophisticated still are symplectic integrators, which solve mechanical systems of differential equations by explicitly and exactly preserving the symplectic structure of Hamilton's equations, which is a curious but hugely important kind of geometry tailored to the two types of variable, position and momentum. Symplectic integrators are especially important in celestial mechanics, where – for example – astronomers may wish to follow the movements of the planets in the solar system over billions of years. Using symplectic integrators, Jack Wisdom, Jacques Laskar and others have shown that the long-term behaviour of the solar system is chaotic, that Uranus and Neptune were once much closer to the Sun than they are now, and that eventually Mercury's orbit will move towards that of Venus, so that one or other planet may well be thrown out of the solar system altogether. Only symplectic integrators offer any confidence that results over such long time periods are accurate.

Computers need mathematics

As well as using computers to help mathematics, we can use mathematics to help computers. In fact, mathematical principles were important in all early computer designs, either as proof of concept or as key aspects of the design.

All digital computers in use today work with binary notation, in which numbers are represented as strings of just two digits: 0 and 1. The main advantage of binary is that it corresponds to switching: 0 is off and 1 is on. Or 0 is no voltage and 1 is 5 volts, or whatever standard is employed in circuit design. The symbols 0 and 1 can also be interpreted within mathematical logic, as *truth values*: 0 means false and 1 means true. So computers can perform logical calculations as well as arithmetical ones. Indeed, the logical operations are more basic, and the arithmetical operations can be viewed as sequences of logical operations. Boole's algebraic approach to the mathematics of 0 and 1, in *The Laws of Thought*, provides an effective formalism for the logic of computer calculations. Internet search engines carry out Boolean searches, that is, they search for items defined by some combination of logical criteria, such as 'containing the word "cat" but not containing "dog"'.

What numerical analysis did for them

Newton not only had to sort out patterns in nature: he had to develop effective methods of calculation. He made extensive use of power series to represent functions, because he could differentiate or integrate such series term by term. He also used them to calculate values of functions, an early numerical method still in use today. One page of his manuscripts, dating to 1665, shows a numerical calculation of the area under a hyperbola, which we now recognize as the logarithmic function. He added together the terms of an infinite series, working to an astonishing 55 decimal places.

Algorithms

Mathematics has aided computer science, but in return computer science has motivated some fascinating new mathematics. The notion of an *algorithm* – a systematic procedure for solving a problem – is one. (The name comes from the Arabian algebraist al-Khwarizmi.) An especially interesting question is: how does the running time of an algorithm depend on the size of the input data?

For example, Euclid's algorithm for finding the highest common factor of two whole numbers m and n, with $m \leq n$, is as follows:

- Divide n by m to get remainder r.
- If $r = 0$ then the highest common factor is m: STOP.
- If $r > 0$ then replace n by m and m by r and go back to the start.

It can be shown that if n has d decimal digits (a measure of the size of the input data to the algorithm) then the algorithm stops after at most $5d$ steps. That means, for instance, that if we are given two 1000-digit numbers, we can compute their highest common factor in at most 5000 steps – which takes a split second on a modern computer.

The Euclidean algorithm has linear running time: the length of the computation is proportional to the size (in digits) of the input data. More generally, an algorithm has polynomial running time, or is of class P, if its running time is proportional to some fixed power (such as the square or cube) of the size of the input data. In contrast, all known algorithms for finding the prime factors of a number have exponential running time – some fixed constant raised to the power of the size of the input data. This is what makes the RSA cryptosystem (conjecturally) secure.

Roughly speaking, algorithms with polynomial running time lead to practical computations on today's computers, whereas algorithms with exponential running time do not – so the corresponding computations cannot be performed in practice,

even for quite small sizes of initial data. This distinction is a rule of thumb: a polynomial algorithm might involve such a large power that it is impractical, and some algorithms with running time worse than polynomial still turn out to be useful.

The main theoretical difficulty now arises. Given a specific algorithm, it is (fairly) easy to work out how its running time depends on the size of the input data and to determine whether it is class P or not. However, it is extraordinarily difficult to decide whether a more efficient algorithm might exist to solve the same problem more rapidly. So, although we know that many problems can be solved by an algorithm in class P, we have no idea whether any sensible problem is not-P.

Sensible here has a technical meaning. Some problems must be not-P, simply because *outputting the answer* requires non-P running time. For example list all possible ways to arrange n symbols in order. To rule out such obviously non-P problems, another concept is needed: the class NP of *non-deterministic* polynomial algorithms. An algorithm is NP if any guess at an answer can be checked in a time proportional to some fixed power of the size of input data. For example, given a guess at a prime factor of a large number, it can quickly be checked by a single division sum.

A problem in class P is automatically NP. Many important problems, for which P algorithms are not known, are also known to be NP. And now we come to the deepest and most difficult unsolved problem in this area, the solution of which will win a million-dollar prize from the Clay Mathematics Institute. Are P and NP the same? The most plausible answer is no, because $P = NP$ means that a lot of apparently very hard computations are actually easy – there exists some short cut which we've not yet thought of.

The $P = NP$? problem is made more difficult by a fascinating phenomenon, called NP-completeness. Many NP problems are such that if they are actually class P, then *every* NP problem is class P as

well. Such a problem is said to be *NP-complete*. If any particular NP-complete problem can be proved to be P, then P = NP. On the other hand, if any particular NP-complete problem can be proved to be not-P, then P is not the same as NP. One NP-complete problem that attracted attention recently is associated with the computer game Minesweeper. A more mathematical one is the Boolean Satisfiability Problem: given a statement in mathematical logic, can it ever be true for some assignment of truth values (true or false) for its variables?

Numerical analysis

Mathematics involves much more than calculations, but calculations are an unavoidable accompaniment to more conceptual investigations. From the earliest times, mathematicians have sought mechanical aids to free them from the drudgery of calculation, and to improve the likelihood of accurate results. The mathematicians of the past would have envied our access to electronic computers, and marvelled at their speed and accuracy.

Calculating machines have done more for mathematics than just act as servants. Their design and function have posed new theoretical questions for mathematicians to answer. These questions range from justifying approximate numerical methods for solving equations, to deep issues in the foundations of computation itself.

At the start of the 21st century, mathematicians now have access to powerful software, making it possible not just to perform numerical calculations on computers, but to perform algebraic and analytic ones. These tools have opened up new areas, helped to solve long-standing problems, and freed up time for conceptual thinking. Mathematics has become much richer as a result, and it has also become applicable to many more practical problems. Euler had the conceptual tools to study fluid flow round complicated shapes, and even though aircraft had not been invented, there are plenty of interesting questions about ships in water. But he did not have any practical methods to implement those techniques.

What numerical analysis does for us

Numerical analysis plays a central role in the design of modern aircraft. Not so long ago, engineers worked out how air would flow past the wings and fuselage of an aircraft using wind-tunnels. They would place a model of the aircraft in the tunnel, force air past it with a system of fans and observe the flow patterns. Equations such as those of Navier and Stokes provided various theoretical insights, but it was impossible to solve them for real aircraft because of their complicated shape.

Today's computers are so powerful, and numerical methods for solving PDEs on computers have become so effective, that in many cases the wind-tunnel approach has been discarded in favour of a numerical wind-tunnel – that is, a computer model of the aircraft. The Navier–Stokes equations are so accurate that they can safely be used in this way. The advantage of the computer approach is that any desired feature of the airflow can be analysed and visualized.

A new development, not touched on above, is the use of computers as an aid to proof. Several important theorems, proved in recent years, rely on massive but routine calculations, carried out by computer. It has been argued that computer-assisted proofs change the fundamental nature of proof, by removing the requirement that the proof can be verified by a human mind. This assertion is controversial, but even if it is true, the result of the change is to make mathematics into an even more powerful aid to human thought.