GRESHAM COLLEGE

8 OCTOBER 2019

THE MATHS OF FUTURE COMPUTING

PROFESSOR CHRIS BUDD OBE

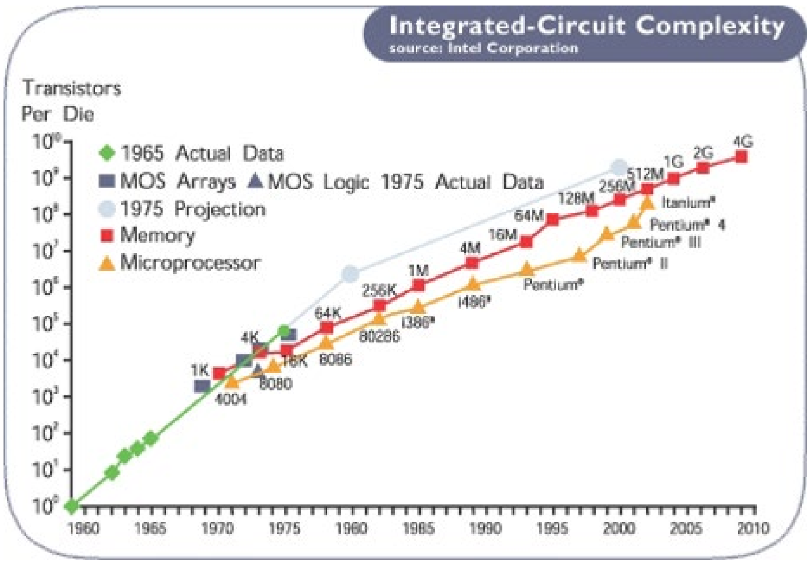
1. Introduction

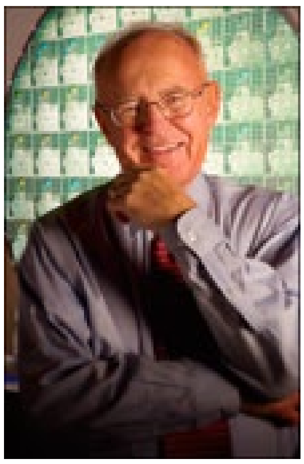
You may think it odd that the first lecture in this series of Gresham lectures by the Professor of Geometry, are on the subject of computing. Surely, you say, such a lecture should be given by the Professor of Information Technology.

It is certainly true that the main impact that computers have on our daily lives comes from the Internet and its various manifestations, all of which are essentially very high-speed conveyors of information. Furthermore, domestic devices that we use every day, such as smart phones, cash machines, TV sets, IPads and, possibly even washing machines, contain more computing power than the main university computer when I was at Cambridge (and certainly more power than that which took men to the Moon in 1969).

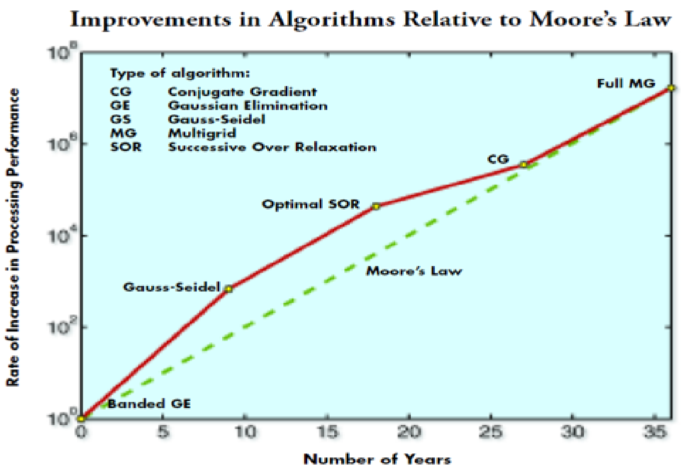
However, mathematics lies at the heart of all modern computing. The first (mechanical) computers designed in the 19th Century were explicitly designed, by mathematicians such as Babbage, Lovelace and Kelvin, to solve mathematical problems. Similarly, the modern electronic computer was invented by Turing and von Neumann and was firmly based on the principles of mathematical logic. Such computers were either used to break codes by using mathematical methods, or were used to solve the mathematical equations in scientific problems such as the atomic bomb or the moon landings. Super computers are now used to solve complex problems, such as high-resolution weather forecasting and climate predictions, using mathematical principles. Advanced mathematical algorithms also lie at the heart of Google, Amazon, Facebook, medical imaging, computer banking, and the operation of the mobile phone.

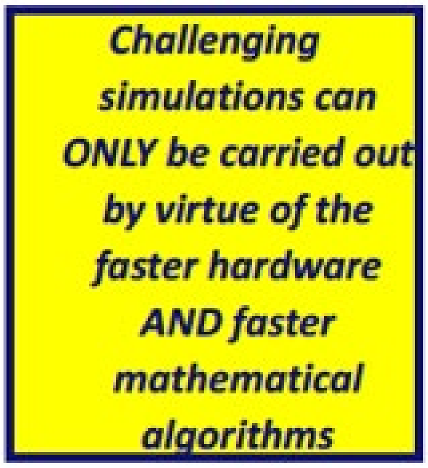
A famous law of computing is Moore's Law which states that computing hardware will double in speed and halve in cost every 18 months. This law has held true for a long time as can be seen below, along with a picture of Moore himself.





However, it is equally true that just as much a speed up in the operation of computers has been due to the creation of advanced mathematical algorithms, such as the conjugate gradient method. We can see this in the figure below




Challenging simulations can ONLY be carried out by virtue of the faster hardware AND faster mathematical algorithms

The title of this lecture asks the question of where computing is heading in the future. My personal opinion is that it will see growth in three main areas. One is in the growth of faster super computers, with ever increasing number of components and increasing complexity. Secondly, and closely related, is the possibility of quantum computers which will allow calculations of even greater complexity in fractions of the tome of conventional computers. Finally, and perhaps the biggest change, is the use of increasingly sophisticated mathematical algorithms in machine learning and artificial intelligence. This latter, highly mathematical field, is already impacting us in many ways. The future impact will be limitless. A tsunami is coming our way and we need to be ready to receive it. How are we going to make sense of it all? Well to quote a leading engineer from Boeing

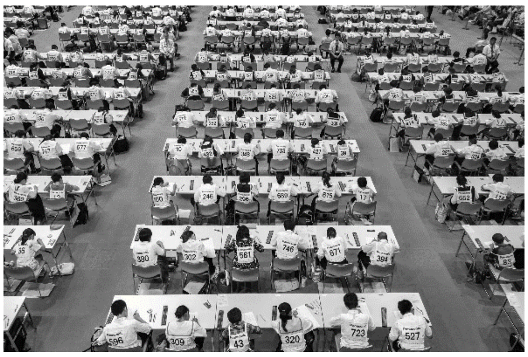
Taking full advantage of computing tools requires more mathematical sophistication not less

In this lecture I will start by looking at how modern computing arose and the role that mathematics has played in it. We will then look at the operation of modern-day computers, which includes the solution of complex mathematical problems. Then we will have a look at the role and operation of computers in the future.

2. The Earliest Computers

Ever since mathematics was invented (or discovered) we have been using practical tools to help us to do it. We can tell that this must be the case, by the fact that our number system is in base 10. This shows that we must have been using our fingers to do our mathematics with. Another very old mathematical computer is the tally stick, which was used to record mathematical calculations. An early example, dated from 9000 BC, is illustrated below.

Another early calculator was the abacus. This uses beads to represent numbers in base 10 and by using several lines can carry digits from one power of ten to the next. This certainly makes rapid calculation possible and has the additional advantage of storing the latest answer so that we can use if for later calculations. The abacus is still very much in use. Here you can see a picture of the 2019 Japanese abacus championship.



In this, the winning student had to solve a problem the answer of which was 8,186,699,633,530,061.

The next major advance in mechanical computation came from the invention of logarithms. This was due to Napier in 1614. Napier also invented a calculating tool now called Napier's bones. As far as I know these were not much used, however his discovery of logarithms was profoundly important for the future of mathematics and science. To quote the Wikipedia article on Napier:

> *"His invention of logarithms was quickly taken up at* **Gresham College** *and prominent English mathematician Henry Briggs visited Napier in 1615. Among the matters they discussed were a re-scaling of Napier's logarithms, in which the presence of the mathematical constant now known as e (more accurately, e times a large power of 10 rounded to an integer) was a practical difficulty. Neither Napier nor Briggs actually discovered the constant e; that discovery was made decades later by Jacob Bernoulli."*

The base-10 logarithm of a number x is the number a so that $x = 10^a$

Base 10 logarithms have many uses, for example the power of a signal is usually expressed in Deci Bels or dB, which is 10 times the logarithm of power expressed in milli Watts. However, their use in computation follows from the fact that if the logarithm of the number y is b, then

$$x\ y = 10^a\ 10^b = 10^{a+b}$$

and

$$x/y = 10^a/10^b = 10^{a-b}$$

Thus the (difficult) operations of multiplication and division become equivalent to the (much simpler) operations of the addition and subtraction of their logarithms. Hence, if the logarithm of a set of numbers could be computed then multiplication and division could be replaced by addition and subtraction. I am old enough to have made of this fact at school. If I wanted to multiply two numbers I would find their logarithms, add them and then find the antilog to give the solution. As an example, we want to find 13.45*23.56. Using 4 figure log tables we have

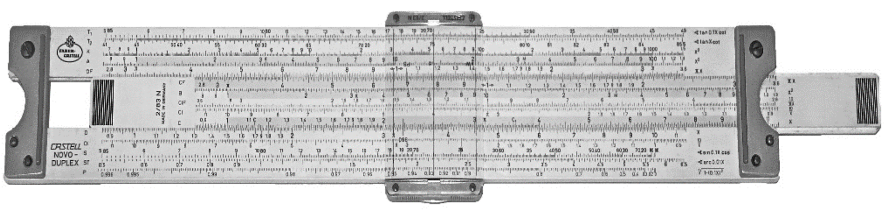
log(13.45) = 1.1287 and log(23.56) = 1.3722

Adding we get 2.5009 and the antilog of this is **316.9**

If we compare this with the exact answer of **316.882** we see that our calculation is pretty close.

This whole process could be automated, although with a loss of precision, by using a slide rule. This is simply a ruler in which the numbers on it are placed at a distance along the rule at a point proportional to its logarithm. Moving one rule along another automatically adds their logarithms and gives the answer.

In the picture below you can see an example of a slide rule



Whilst slide rules are great fun to use, and I used them at school up to and including my mathematics and physics O-levels, they have their disadvantages. The results are not very accurate, indeed it is hard to get precision to more than 3 decimal places. If calculations are repeated so the errors accumulate very rapidly. They are also slow to use so that they would be impractical for very long calculations. A further 'disadvantage' is that it could be ambiguous when using a slide rule whether the answer was, say, 316.9 or 3.169 or 3169. Thus to use a slide rule effectively you have to have a rough idea of what the answer is in advance. Actually this is no bad thing. One thing computers are VERY good at is producing the wrong answer very quickly if they are asked the wrong question. Having a rough idea of what the answer is in advance can help to check whether the computer is performing correctly. This is, of course, where mathematics is vital. I will return to this point when we look at the use of super computers to predict climate change.

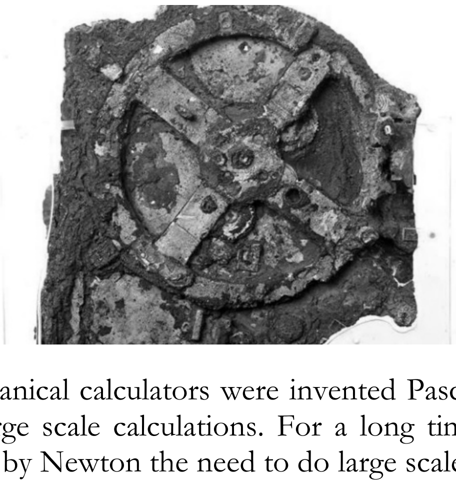
Slide rules have the advantage that once set up they can compare the ratio of many numbers at once. Thus, whilst out shopping, you can use a slide rule to compare the relative value of many goods at once. I think it would be good to see slide rules used again.

Logarithms still are very important in all areas of science, and computing the logarithm over a finite field lies at the heart of internet security.

3. Mechanical Computers: Babbage, Lovelace, Kelvin and The Difference Engine

3.1 The Difference Engine

As stated above, one of the big problems with the slide rule is that it cannot carry out a lot of calculations automatically. This is almost a defining feature of a (modern) computer and it requires a mechanical or electronic approach. The notion of using a mechanical calculator for mathematical functions can be traced back to the Antikythera mechanism of the 2nd century BC. This used a (for the time) incredibly sophisticated set of gears and cog wheels to calculate celestial motions.



Other more modern examples of mechanical calculators were invented Pascal and Leibniz in the 17th century. However neither of these could do large scale calculations. For a long time, this did not matter very much, however, after the invention of calculus by Newton the need to do large scale calculations become very pressing. In Newton's time it centred around calculations of the movement of the celestial bodies, such as calculating the location of the moon. Newton himself complained about the need to do the tedious calculations, but as can be seen from his papers, he was quite prepared to get his hands dirty to do them.

Following on from Newton's work came the great developments in navigation, which I reported in my Gresham lecture 'Can Maths Tell You Where You Are?' By making measurements of the location of the celestial bodies at known times it was possible to determine your location provided that the location of these bodies could be predicted in advance. This required making careful calculations of their positions using Newton's laws. These had to be done by hand, and the results were the Ephemerides tables of the locations of the heavenly bodies. These calculations were done by 'computers' who were usually human beings equipped, if they were lucky, with mechanical calculating devices. Many of the navigational calculations were routine, and involved simple addition and subtraction. However, they had to be done a very large number of times. It occurred to several people, most notably the mathematician Charles Babbage, that the whole process could be automated. This would not only speed it up but would also minimise the possibility of error.

Babbage's approach used a method, still in use today, involving the use of *difference tables* and *Newton's method of divided differences*. Essentially this method starts with a series of numbers in a table, which arise from the evaluation of a polynomial, or other regular mathematical function that can be closely approximated by a polynomial. This included the (spherical) trigonometric functions related to navigation, as well as logarithms and anti-logarithms. If differences of these numbers are taken, and then differences of these differences, and so on, then eventually the result is a series of constants. If this process is reversed, then it gives a means of evaluating the original polynomial. This process reduces quite complex calculations, such as the evaluation of polynomials, to a large number of systematic additions. This is perfectly suited to the use of a (mechanical) computer

Below we can see the finite difference table for the polynomial

$$y = 1 + \frac{3}{2}x^2 + \frac{1}{2}x^3$$

For x=0,1,2,3,4 the values of the polynomial are in the first row. In the second row we see the differences between these values (first order differences). In the third row the (second order) differences of these, and in the final fourth row the differences of these. Note that the final row consists of a series of 3s only!!

```
1     3     11    28    57
   2     8     17    29
      6     9     12
         3     3
```

Now suppose that we want to find the value of the polynomial when x = 5. We write a 3 on the bottom line and build up the table as follows

12 + 3 = 15
29 + 15 = 44
57 + 44 = 101

We also have that $$1 + \frac{3}{2}\,5^2 + \frac{1}{2}\,5^3 = 101.$$

Thus we have calculated the polynomial at x = 5 without having to do any multiplications. To find more values we simply repeat the process starting with more 3s at the bottom. More generally a polynomial of degree n will have a series of constants in the (n+1) st row of differences. So by starting there we can build up any polynomial we wish.

This procedure is extremely straightforward, but it is also time consuming and prone to error.

In 1823 Charles Babbage came up with the revolutionary idea that the same routine difference calculations could be done mechanically. He designed a mechanical computer to do this calculation which was called the Difference Engine.

To achieve the calculation above the machine needed to do several things, *all of which are vital to the operation of the modern computer.*

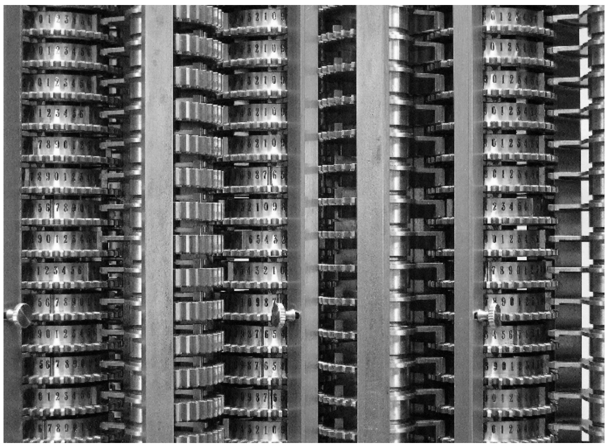
Firstly it had to be able to represent numbers to a certain precision. This is often thought of as the number of decimal digits used to represent the number. For example, the square root of 2 is given by 1.4142135623730950488…. If we represent it by two decimal digits it is given by 1.4 and if by 9 then it is given by 1.41421356. The more digits we use the more accurate the calculation, but the harder it is to store them, and the slower the resulting calculation.

Secondly it must be able to store these numbers. To calculate an nth degree polynomial, it must store (at least) n+1 numbers.

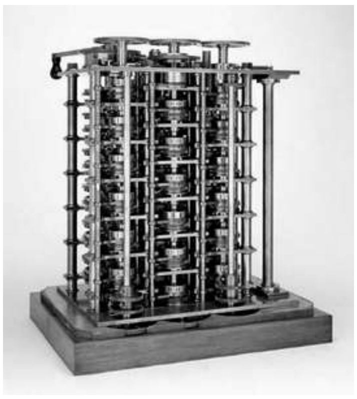
Thirdly it must be able to add these numbers quickly and accurately.

In the difference engine these operations were all done mechanically. The numbers in particular were represented as decimals by the position of cog wheels on a column. (The more cog wheels the greater the accuracy). The difference engine then had a set of columns numbered 1 to N. In the Nth column the constant (e.g. 3 above) was stored. This was then successively added to the columns N-1, N-2, N-3 etc. Finally, the results of the calculation appeared in column 1 and could then be printed. Before this could be done it was initialised by setting each column to the first column of the difference table. So, to work out the polynomial above it would use 4 columns, which would be set to the values 1,4,6,3. All of the additions would be synchronised by the turning of a central main shaft.

Here is an example of the columns, with the toothed wheels showing the stored decimal digits. These could then be added together mechanically using the same principles as the earlier mechanical calculators.



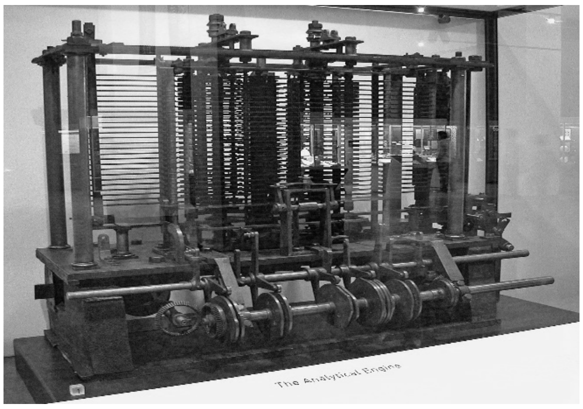
A substantial government grant of £17,000 led Babbage and his chief engineer Joseph Clement (with whom he later fell out) to produce in 1832 a small working model of the calculating section of Difference Engine No. 1. This operated on 6-digit numbers and second-order differences. Lady Byron (mother of Ada Lovelace) reported on seeing the working prototype in 1833 that "We both went to see the thinking machine (for so it seems) last Monday. It raised several Nos. to the 2nd and 3rd powers and extracted the root of a Quadratic equation." Unfortunately, the full project was not accomplished successfully, and it was abandoned in 1842.



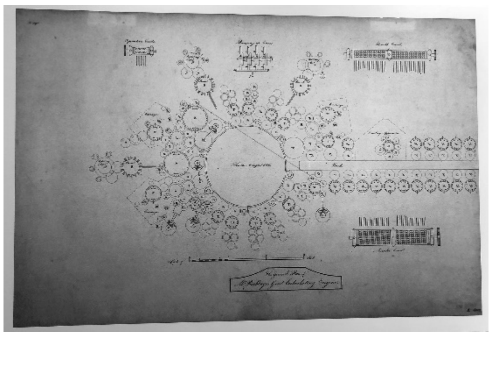
However, Babbage's difference engine No. 2, illustrated above, was finally built by Allan Bromley in 1991 and is on display at the Science Museum in London. This machine can store 8 numbers of 31 decimal digits each and can thus tabulate 7th degree polynomials. to that precision.

3.2 The Analytical Engine

Whilst sophisticated for its time, the difference engine could only do one thing, namely calculating polynomials. Babbage then went on from the design of the difference engine to the much more sophisticated analytical engine (illustrated below) which could be programmed to do many different tasks. It was this that marked the start of modern computing.



The Analytical Engine introduced many of the logical features of modern computer programming. In particular, it incorporated an arithmetic logic unit, a control flow of the flow of a calculation, in the form of conditional branching and loops, and had a memory to store the results of its calculations. By doing this he produced a machine which anticipated Turing’s work 100 years later. The program and the data for the engine were inputted to it on the same sort of punch cards which were then in use to control looms. (I should say that when I started as an undergraduate at Cambridge in 1979 we were still programming the main university computer by using punched cards!)



A key role in the development of the programming format and language was provided by Ada Lovelace (Lady Byron's daughter). She wrote what is now considered to be the first computer program, which was to compute Bernoulli Numbers B m. These are the coefficients of the Taylor series expansion of the expression

$$\frac{x}{e^x - 1} = \sum_{m=0}^{\infty} \frac{B_m \, x^m}{m!}$$

The resulting algorithm, often called the first ever computer program, and written in what we would now call assembler code for the analytical engine, is given below. More information about the life of Ada Lovelace is given in the book [1] and the lecture by Raymond Flood [2].

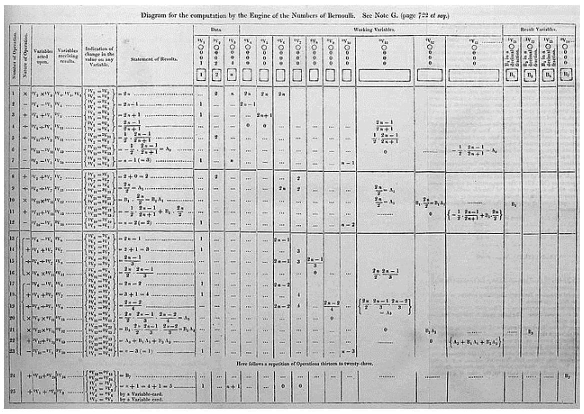
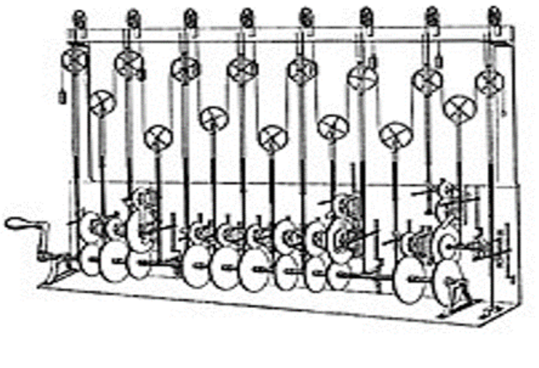


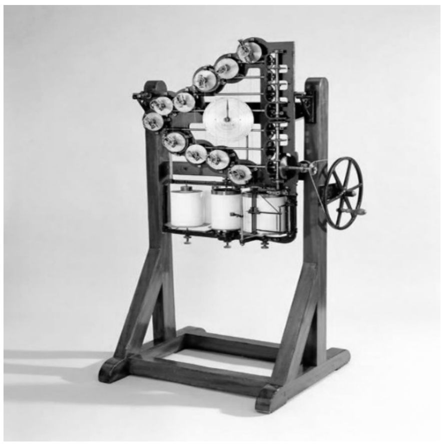
Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			Working Variables.										Result Variables.			
						$^1V_1$	$^1V_2$	$^1V_3$	$^0V_4$	$^0V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	$^1V_{21}$ B_1 in a decimal fraction.	$^1V_{22}$ B_3 in a decimal fraction.	$^1V_{23}$ B_5 in a decimal fraction.	$^0V_{24}$
						1	2	n											B_1	B_3	B_5	B_7
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\{^1V_2 = {}^1V_2, {}^1V_3 = {}^1V_3\}$	$= 2n$		2	n	2n	2n	2n											
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\{^1V_4 = {}^2V_4, {}^1V_1 = {}^1V_1\}$	$= 2n - 1$	1			2n − 1													
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\{^1V_5 = {}^2V_5, {}^1V_1 = {}^1V_1\}$	$= 2n + 1$	1				2n + 1												
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\{^2V_5 = {}^0V_5, {}^2V_4 = {}^0V_4\}$	$= \frac{2n-1}{2n+1}$				0	0						$\frac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\{^1V_{11} = {}^2V_{11}, {}^1V_2 = {}^1V_2\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$		2									$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\{^2V_{11} = {}^0V_{11}, {}^0V_{13} = {}^1V_{13}\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$											0		$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\{^1V_3 = {}^1V_3, {}^1V_1 = {}^1V_1\}$	$= n - 1 \; (= 3)$	1		n							n − 1							
8	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\{^1V_2 = {}^1V_2, {}^0V_7 = {}^1V_7\}$	$= 2 + 0 = 2$		2					2										
9	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\{^1V_6 = {}^1V_6, {}^0V_{11} = {}^3V_{11}\}$	$= \frac{2n}{2} = A_1$						2n	2				$\frac{2n}{2} = A_1$						
10	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\{^1V_{21} = {}^1V_{21}, {}^3V_{11} = {}^3V_{11}\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$											$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$		B_1			
11	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\{^1V_{12} = {}^0V_{12}, {}^1V_{13} = {}^2V_{13}\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$												0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\{^1V_{10} = {}^2V_{10}, {}^1V_1 = {}^1V_1\}$	$= n - 2 \; (= 2)$	1									n − 2							
13	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\{^1V_6 = {}^2V_6, {}^1V_1 = {}^1V_1\}$	$= 2n - 1$	1					2n − 1											
14	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\{^1V_1 = {}^1V_1, {}^1V_7 = {}^2V_7\}$	$= 2 + 1 = 3$	1						3										
15	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\{^2V_6 = {}^2V_6, {}^2V_7 = {}^2V_7\}$	$= \frac{2n-1}{3}$						2n − 1	3	$\frac{2n-1}{3}$									
16	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\{^1V_8 = {}^0V_8, {}^3V_{11} = {}^4V_{11}\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$								0			$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\{^2V_6 = {}^3V_6, {}^1V_1 = {}^1V_1\}$	$= 2n - 2$	1					2n − 2											
18	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\{^2V_7 = {}^3V_7, {}^1V_1 = {}^1V_1\}$	$= 3 + 1 = 4$	1						4										
19	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\{^3V_6 = {}^3V_6, {}^3V_7 = {}^3V_7\}$	$= \frac{2n-2}{4}$						2n − 2	4		$\frac{2n-2}{4}$								
20	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\{^1V_9 = {}^0V_9, {}^4V_{11} = {}^5V_{11}\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$									0		$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = A_3\right\}$						
21	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\{^1V_{22} = {}^1V_{22}, {}^0V_{12} = {}^2V_{12}\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$											0	$B_3 A_3$			B_3		
22	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\{^2V_{12} = {}^0V_{12}, {}^2V_{13} = {}^3V_{13}\}$	$= A_0 + B_1 A_1 + B_3 A_3$												0	$\{A_3 + B_1 A_1 + B_3 A_3\}$				
23	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\{^2V_{10} = {}^3V_{10}, {}^1V_1 = {}^1V_1\}$	$= n - 3 \; (= 1)$	1									n − 3							
Here follows a repetition of Operations thirteen to twenty-three.																						
24	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\{^4V_{13} = {}^0V_{13}, {}^0V_{24} = {}^1V_{24}\}$	$= B_7$																	B_7
25	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\{^1V_1 = {}^1V_1, {}^1V_3 = {}^1V_3, {}^5V_6 = {}^0V_6, {}^5V_7 = {}^0V_7\}$	$= n + 1 = 4 + 1 = 5$ by a Variable-card. by a Variable card.	1		n + 1			0	0										

3.3 Kelvin's Tidal Calculator

Whilst the difference engine and the analytical engine had tremendous future impact, neither of them were constructed during the lifetime of Babbage. However, a mechanical computer which did get built was designed by Lord Kelvin. This was used to calculate the tides. It worked by decomposing the function describing the height of the tides into a sum of a number of periodic cycles or different frequencies and amplitudes. (A very similar decomposition is used in a synthesiser to produce different sounds.) The cycles could then be reproduced by using cog-wheels of different cycles coupled together by a cable. Below you can see both the original drawing and a version of the tidal calculator, in the London Science Museum.



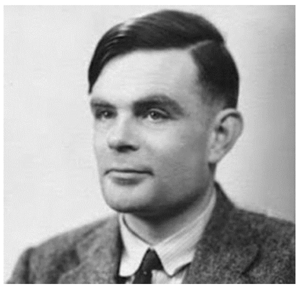


The Kelvin tidal computer is an example of what we now would call and 'analogue computer'. Once set up with data from previous tidal events, it could calculate with great accuracy the height of future tides. It was so good that it was used for the tidal calculations for the D-day landings, and similar computers were used until the 1960's. At this point they were replaced by the modern electronic computer which we will discuss next.

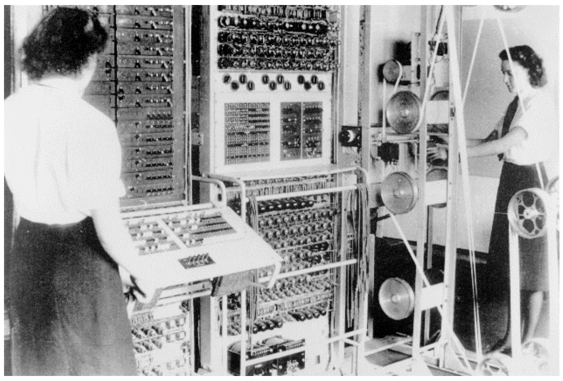
4. Turing, Von Neumann and the Invention of The Modern Electronic Computer

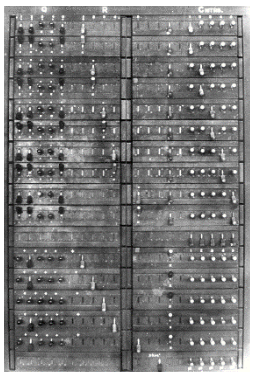
Whilst many people have contributed to the invention of the modern electronic computer, two names dominate: Alan Turing and John von Neumann. You can learn a lot more about the life and work of Turing and von Neumann in the lecture [3] by Raymond Flood my predecessor as Gresham Professor of Geometry. I will summarise some of their achievements here.

4.1 Alan Turing



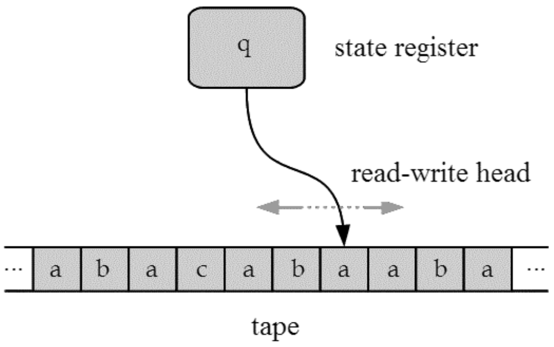
We now move forward 100 years to the invention of the modern electronic computer. The person most associated with this invention, at least in the minds of the British, is Alan Turing. Turing is perhaps most famous for his work in the Second World War in which he used electronic and electro-mechanical machines to crack the German Enigma and Lorenz codes. The first machine that he designed, around 1940, was the Bombe, which was based on earlier Polish machines. The Bombe had a large number of rotating drums, which simulated the wheels on the mechanical Enigma coding machine. It worked by rapidly changing the drum positions so that the received Enigma traffic could be compared with guesses (cribs) for its meaning. The Bombes were not programmable in the sense of the Analytical machine, but were set up in advance through the use of patch cables. In some ways the Bombe resembled the difference engine and is very different from the modern computer. Turing later worked on the Colossus computer, which was designed and built by Tommy Flowers in 1943. Flowers worked for the post office and is today celebrated by the BT Tommy Flowers Institute). Colossus was a fully electronic device, using 1700 valves and thyratrons to operate and to store data. It could be programmed using plug panels and switches, and it operated in binary, using the valves to perform Boolean and counting operations. Input was provided by paper tape (which was the medium that I used to store programs when I was at school). The operation of the Colossus was designed to replicate that of the German Lorenz coding machines. The operation of these had been worked out by Bill Tutte without ever seeing an original machine. (Tutte went on to become a prominent computer scientist). Below you can see the Colossus computer on the left and on the right the switch panel used to program it.





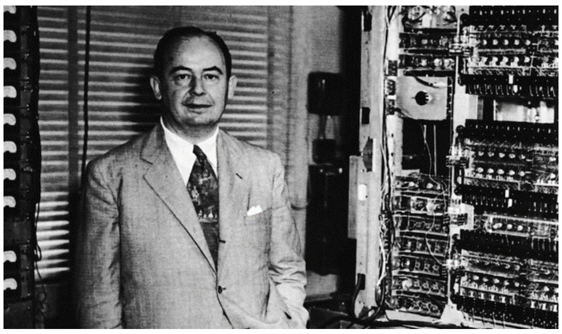
Although the Colossus was the first of the electronic digital machines with programmability, albeit limited by modern standards, it was not a general-purpose machine, being designed specifically for a range of precise cryptanalytic tasks, most involving counting the results of evaluating Boolean algorithms, closely linked to the design of the German Lorenz encryption machine.

Whilst it is Turing's work on code breaking that is best remembered, his main contribution to the development of the computer probably came earlier when he was working at Cambridge in 1936. During this time, he came up with the notion of what it now called a *Turing Machine*. This is a theoretical computer, which can exist in a number of states. In its operation it can access an infinitely long tape. It can read and write onto this tape and also change state according to what it reads and a series of rules. It then can move the tape. Turing was able to study the operation of this machine in detail, and was able to prove that it was capable of doing an arbitrary computation. By providing a mathematical description of this procedure, he was able to prove properties of computation on much more general machines, including fundamental limitations on what could be computed. See [4] for more details.



Turing completeness is the ability for a system of computer instructions to simulate a Turing machine. A programming language that is Turing complete is theoretically capable of expressing all tasks accomplishable by computers; nearly all programming languages are Turing complete if the limitations of finite memory are ignored. However, a Turing machine itself is never used for a practical computation. The format of the modern computer followed from the next set of developments in computing.

4.2 John Von Neumann and His Legacy

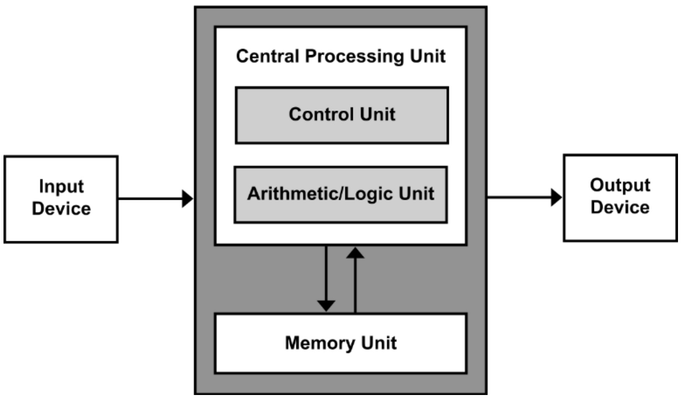


One of my favourite mathematicians of all time is John von Neumann. If anyone could be said to have made major contributions to the whole of 20th Century mathematics, then it is him. His original work was in set theory and the operator theory of Banach spaces, pure branches of pure maths, which he then applied to quantum physics. He was one of the developers of what is now called game theory and its applications to economics. He was a pioneer of the new subject of numerical analysis (which I will describe shortly). He developed the modern theory of bubbles and foams. He understood the basic physics of the shock waves involved in the atomic bomb. He was renowned for his lightning fast calculating abilities. And crucially for this lecture, he essentially invented the modern electronic computer. He was a legend at Princeton where he threw spectacular parties. Very sadly he died of cancer in 1957 at the age of only 53. John von Neumann's interest in computation came largely through his work on the Manhattan Project to design the atomic bomb. During World War II von Neumann worked on the Manhattan Project with theoretical physicist Edward Teller, mathematician Stanisław Ulam and others, problem solving key steps in the nuclear physics involved in thermonuclear reactions and the hydrogen bomb. He developed the mathematical models behind the explosive lenses used in the implosion-type nuclear weapon, and coined the term "kiloton" (of TNT), as a measure of the explosive force generated.

Despite all of this work he was famously quoted as saying:

"I am thinking about something much more important than bombs. I am thinking about computers."

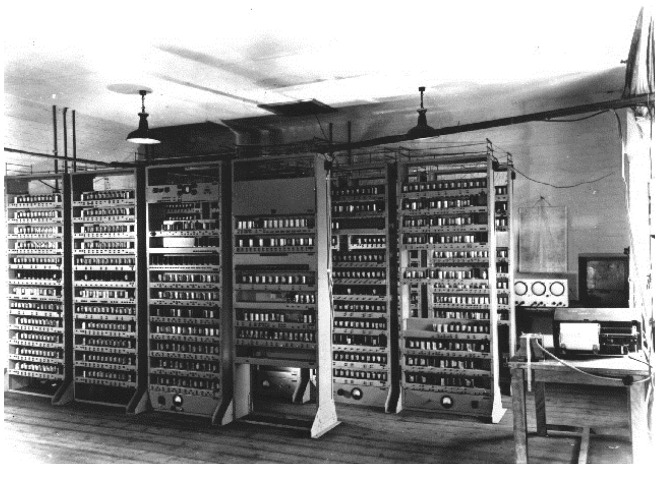
After the war von Neumann worked with the ENIAC and later EDVAC computers developed in America. As part of this work he was one of the pioneers of using computers both in weather forecasting and in the then very new science of forecasting the climate. As a result of these experience he wrote the monograph, *First Draft of a Report on the EDVAC* [5]. This monograph was seminal in the development of computing. In it he proposed the main ideas that embodied the "stored-program" concept that we now call the Von Neumann architecture. This is now the basis of all modern computers and is illustrated below



The key idea behind this architecture (which I was taught about in school in the early 1970s) was the storing of the program in the same memory as the data and distributing the tasks in the computer between a CPU (Central Processing Unit) which contained and arithmetic and a control unit, and memory. When I started computing in the 1970s the CPU was about the size of a filing cabinet and the memory was on magnetic cores, magnetics drums and magnetic tape. Now the CPU (a microprocessor) is tiny and on a chip, alongside the (flash) memory.

The **Electronic delay storage automatic calculator** (**EDSAC**) was an early British computer inspired by von Neumann. The EDSAC machine was constructed by Maurice Wilkes and his team at the University of Cambridge, Mathematical Laboratory. It is generally accepted that the EDSAC was the first *practical* general purpose stored program electronic computer. As head of the Mathematical Laboratory, Wilkes' brief was to provide 'mechanical' aids that would assist mathematicians, scientists and engineers at the university to perform complex and time-consuming calculations. He had observed research workers doing laborious computations with the aid of mechanical desk calculators and mathematical tables. His prime motive in building EDSAC was to provide them with faster and better facilities. Several of my lecturers at Cambridge worked on this project. EDSAC was based on (thousands of) valves, and these got very hot and often failed. All of the team working on it had to be skilled in putting out fires, and coping with the failure of the valves. The memory (as the title implied) was (acoustic mercury) delay lines, which could store 1024 words allowing a program of about 800 lines. The programs themselves were written on paper tape (I was still doing this 10 years later on my school computer). Work on EDSAC (see below) started during 1947, and it ran its first programs on 6 May 1949, when it calculated a table of square numbers and a list of prime numbers. The Star in a June 1949 news article about the EDSAC computer, said (with a good degree of insight):

"The 'brain' [computer] may one day come down to our level [of the common people] and help with our income-tax and book-keeping calculations. But this is speculation and there is no sign of it so far."



EDSAC was subsequently used to do many scientific computations in both pure and applied mathematics. Although EDSAC was modest in terms of modern-day computers (there were only 18 operation codes, just 1024 words of memory, later extended to 1024 and instructions were executed at a rate of approximately 650 per second.) it was far faster than any other machine in use in Cambridge at the time, and transformed work there, leading to several Nobel Prizes. It was finally shut down on 11 July 1958, having been superseded by EDSAC 2, which remained in use until 1965. EDSAC and its many uses won recognition and financial support from outside sources. In particular funds were provided by the catering company J Lyons which went on to build the LEO I computer, the world's first business computer, based on the EDSAC design.

4.3 What Happened Next

Since EDSAC the development of the electronic computer has been very rapid, and I have been privileged to see much of this happening in my own lifetime. Key to this development has been the invention of the transistor, and then the integrated circuit, which has allowed a rapid miniaturisation of electronic components. This has permitted a vast increase in the number of computations that can be made, along with the speed and reliability of these computations, whilst at the same time reducing the cost. At the same time, computers have become easier to use. The EDSAC and similar machines (including the BCL SUSIE machine I used at school, and illustrated below) were programmed in *machine code*. This was a set of numerical instructions and took a lot of time, practice and patience to use.

However, thanks in no small part to the work of (generations of) mathematical logicians, high level languages were rapidly developed which allowed computers to be programmed in a language much more like English. Early examples of these were COBOL, FORTRAN, PASCAL and (the wonderful) ALGOL. Later on came BASIC, which was the language many of us first used when we learned computing on the (legendary) BBC micro, and the now very widely used object oriented language C++. Other more sophisticated languages such as LISP and HASKELL are used for machine learning and AI applications.

We now have come to the point where every home has (probably) several computers. This gives us the expectation of boundless new ways that computing may be used in the future. We will now look at some of the ways that this development may happen.

5. Scientific Computing and The Future of Large-Scale Computing

5.1 Using Computers to Solve Mathematical Problems

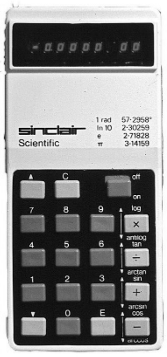
As advertised, my own field of research is in *scientific computing*, which is the business of computing the solution to problems posed in scientific terms. As well as its obvious applications in all areas of science, including physics, chemistry, biology, neuro-science, cosmology and climate science, scientific computing also plays a central role in all branches of engineering including aircraft and car design, in the drug industry, in medicine, and in genomics. It is also of major importance in the film animation and gaming industries. Some of the 'biggest' calculations currently being undertaken

Following the work of such great mathematicians as Newton and Euler we now know that one of the best ways to describe nature in mathematical terms is through (ordinary and partial) differential equations. It follows that a large part of scientific computing is involved with solving exactly such equations as accurately, reliably, and quickly, as possible. A very large part of these calculations then involves the analysis of very large systems of linear equations, the problems of inverting matrices and finding their eigenvalues.

In a sense, a scientific computer is a very large and sophisticated version of a programmable electronic calculator and one of the things that distinguishes scientific computing from many other branches of computing is that it has to work with *real numbers*. That is any number which could arise as part of a calculation. An example of this is

$$1/3 = 0.3333333333333333333333 \ldots.$$

This has to be represented as a *floating point number* in a computer for further calculations. However, this is not possible to do exactly as it has an infinite number of decimal digits. There is then a trade off in the representation of any number between accuracy and speed of calculation. Early calculators such as the 1974 Sinclair Scientific illustrated (and which I used at school), had very low accuracy, but modern computers typically work to about 20 decimal places of accuracy. A measure of computer speed is its ability to do floating point operations or *flops*. A modern computer can do many trillions of these every second.



The study of how to use a scientific computer to solve a mathematical problem accurately and efficiently is called *numerical analysis*. This is my own field, so I'm biased, but I regard it as a very important are of mathematics which combines the rigour of pure mathematics with direct applications to real world applications. Research in numerical analysis is both at the forefront of modern mathematics, and is leading the way in modern applications of computing to problems as diverse as climate modelling, oil prospecting, and medical imaging. The key question asked in numerical analysis is how to develop an algorithm which solves a mathematical problem as accurately and quickly as possible, knowing that there is always a trade-off (as we will see shortly) between speed and accuracy. Just as (if not more) importantly, a numerical analyst will try to determine rigorously both an estimate for the accuracy and the cost of any such calculation. See [6] for an excellent account of the subject of numerical analysis.

Typical problem in numerical analysis might be how to solve the solve the ordinary differential equation system,

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(t, \mathbf{u})$$

or the partial differential (wave) equation $$\frac{\partial^2 u}{\partial t^2} = c(x)\frac{\partial^2 u}{\partial x^2}$$

or perhaps the matrix eigenvalue problem $$A\,\mathbf{x} = \lambda\,\mathbf{x}.$$

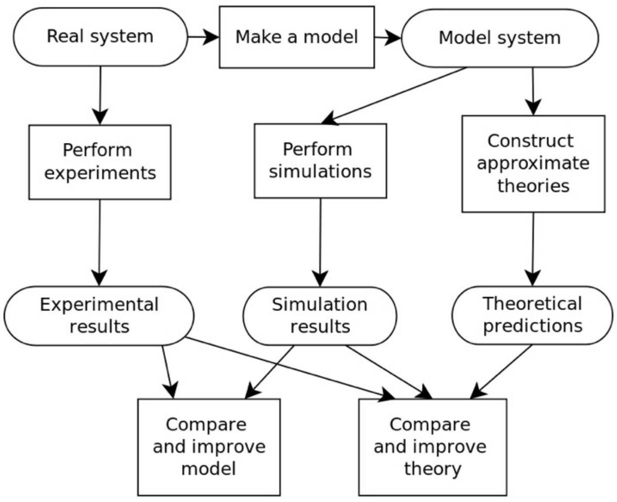
Such problems lie at the heart of many diverse scientific problems and hence the construction of efficient numerical algorithms to solve them is of critical importance. For example, von Neumann developed algorithms to solve the wave equation, which were first used to design the atomic bomb. Now (developments of the) same algorithms are used in such diverse applications as designing aircraft and cars, medical imaging, oil prospecting, curing cancer and saving the whales (see my 2020 Gresham lecture). The matrix eigenvalue problem arises both in the solution of the wave equation, and also (for a very large system) in the algorithms that Google uses to rank web pages.

5.2 Modelling the World Using Scientific Computing

I first came across the real power of scientific computing during my time as a research assistant at Oxford. On 18 November 1987, at approximately 7.30pm, a fire broke out at King's Cross tube station. The fire started under a wooden escalator due it is thought, to a dropped match. For a short while the fire burned slowly. However, at 7.45pm, it erupted in a sudden flashover into the underground ticket hall. The fire seemed minor until it suddenly increased in intensity, and shot a violent, prolonged tongue of fire, and billowing smoke, up into the ticket hall.

This sudden increase in the ferocity of the fire killed 31 people and injured a further 100. The devastation to the ticket hall is shown below.

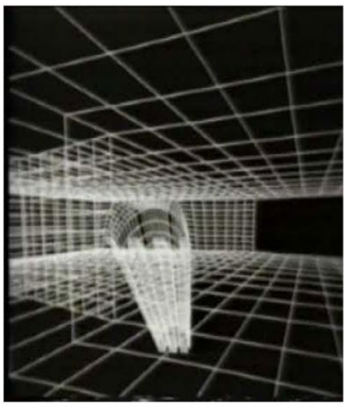
At the time, this incident affected me a great deal as I was due to pass through King's Cross just 24 hours later. I often thought that if my meeting had been a day earlier, I could have been caught up in the fire myself. Later on, it affected me again as I became involved (in a minor way) in the public inquiry, which was conducted from February to June 1988. The investigation comprised a series of experiments combined with a computer simulation conducted by AEA Technology, who I was working with at the time. The reason for the sudden transition in the fire intensity, was discovered by the computer simulation, and only confirmed later by the experiments. It was due to a previously unknown effect called the trench effect in which the fire spout was caused by the escalator stairway acting like a chimney. Computer models are often criticised for simply telling you what you knew already, however in this case the computer predicted something genuinely new. A full account of this study is given in [7]. To do this calculation the (super) scientific computer is part of a broader process of mathematical modelling which I will cover in more detail in a future lecture (on how maths saves the whales and cures cancer). The whole process can be summarised as follows:

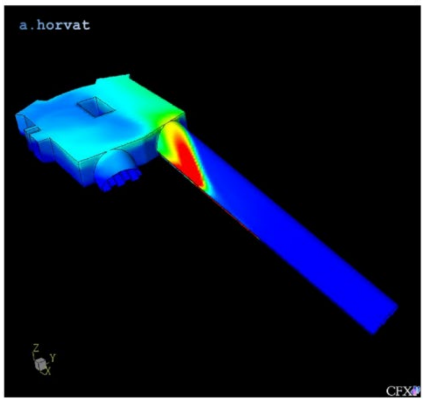


Critical to this calculation, and relevant to today's lecture is the box in the middle, which says '*perform simulations*'. In scientific computing this mainly involves solving the mathematical equations that are described in the model. The Kings Cross fire gives us a good example of the sort of equations that need to be solved to do this sort of calculation. To simulate the fire, you needed to write down equations for how the air moved around the station, and how the heat was conveyed by convection and radiation from one part of the station to another. These equations then had to be coupled with equations for how the materials in the station burned and otherwise reacted to the high temperatures of the fire. Finally the geometry of the station had to be accounted for, including the various exits and entrances.

These equations, whilst complex, are well understood, and are very similar to the equation which are used to predict the weather. Essentially, the system couples the Navier-Stokes equations of fluid motion, with the laws of thermodynamics, and Arrhenius' equations for chemical reactions. They are far too hard to be solved by hand. To find the solution on the computer the geometry of the underground station was divided up into a large number of cuboids called '*finite volumes*'. The equations were then expressed (in integral form) on each volume to produce what is called a discrete system. This final system could then be solved to give the results needed to understand the behaviour of the fire.

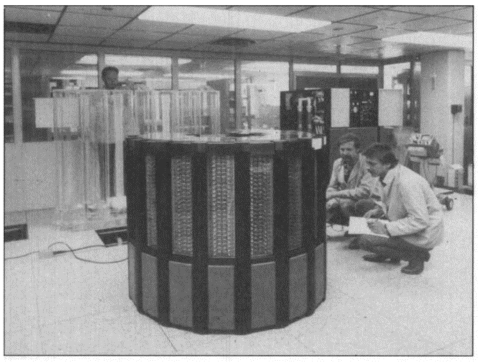
Below on the left you can see the division of the booking hall and the elevator into the finite volumes. On the right you can see the resulting temperature calculations.





In such a calculation there is a trade-off between the number of finite volumes, and the accuracy speed of the calculation. Roughly speaking, if there are N^3 such volumes (N in each spatial dimension), then the error decreases at a rate 1/N^2 and the computational time increases, often at a rate approaching N^6 or even worse. Thus, if N is large we pay a big time penalty for any increase in accuracy.

The calculations at the time were done on a Cray 2 machine at Harwell. This was one of the first ever super computers and had the (at the time) unheard of speed of 1.7 G Flops and a 2G byte RAM (see later to see how things have evolved since). Below is a cutting from the April 1987 edition of Electronics and Power [8] in which we can see the Cray 2 being installed. Note the 1980s hair styles.



Even though this was a very powerful computer indeed for the time, the rapid increase in the computational time as N above increases, meant that the discretisation of the overall geometry was fairly coarse.

If we now fast forward 30 years, the power of a Cray 2 is what most of us will have in our laptops. At the same time the power of the supercomputer has advanced significantly and we will look at this in the next section.

However, all supercomputing whether it is for the computation of fires, the calculation of flow over an aircraft, or the simulation of the brain, suffers from the same trade-off between accuracy and speed. This means that even on a supercomputer some big and calculations can take months to do accurately.

5.3 The Future of Scientific Computing

Scientific computing is a field where we will continue to see huge increases in computing power in the future. The growth will in part be one of sheer computing power, coupled with better algorithms to make full use of that power.

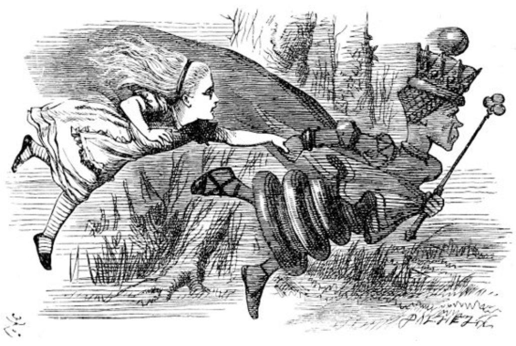
5.3.1 Hardware

Moore said in 1965, that the number of transistors that would fit in a given circuit space was doubling every year. Later this slowed down to every two years. Electronic components based on Silicon can only shrink so much before they run into the limits of physics. The smallest transistors today consist of about a hundred atoms. Silicon transistors are down to 10 nano meters, and there is now talk of 5-nanometer electronics, but that may be the limit, in part due to the effects of quantum physics. More speed will have to come from a different material or a different basic technology. One way is to use Carbon. Carbon nanotubes are smaller than 10 nano meters and offer faster switching speeds than silicon. They dissipate heat better. Producing them is hard and existing technology just can't create good-quality nanotubes. Another possibility is to use graphene, a hexagonal lattice of carbon atoms just one atom thick. Scientists have developed a graphene transistor, which requires lower power than silicon transistors and therefore can run faster. Three-dimensional construction of processors might get more speed out of components without reducing their size, since it can bring them closer together. Another possibility will be to use 'light based' computers.

The big problem in making things smaller is then heat dissipation, and a real problem with modern computing is keeping them cool. At present this is a major limitation to their usage, as is the sheer amount of energy that they need to use. This is in the order of Mega Watts. (Ironically, computers to compute climate change contribute to global warming.) One good way to reduce the amount of energy is simply to do calculations more efficiently. This can involve developing better mathematical algorithms, using reduced precision when you can get away with it, or replacing the solution of a physical problem by using a machine learning alternative (see later). All of these are the subject of intensive research.

5.3.2 Software

The rate at which computing power grows might decrease over the next few years, as chip makers work for the next breakthrough. However, if we're heading into a few years with less hardware growth, computer developers will be under pressure to get more performance out of the available power. *Moore's Law has encouraged ever more wasteful coding.* Software developers have often assumed that if their code runs a little slow on the current generation of machines, the next one will take care of the problem. The result is that computers don't seem any faster to the user. As the Red Queen said in Alice Through the Looking Glass, *it takes all the running you can do just to stay in the same place.* Anyone with a home computer will be very familiar with this problem.



Thus, the future of computing must also see a change in the way that we use and develop computer software.

A very significant extra area of advance will be in the use of parallel processing methods in which a lot of operations are carried out at the same time. Modern computers are multi-core machines, with a lot of processors all working together simultaneously. For example, the latest Met Office Cray XC40 computer, has 460,000 separate cores.

A nice example of a parallel computation is that of calculating the factorial of a number defined by

$$n! = n(n-1)(n-2)(n-3)\ldots 3.2.1.$$

This is usually calculated recursively via repeating the instructions

o! = 1,
n! = n * (n-1)!

If this is repeated n times then we can calculate n! This process takes n operations.

However, with a parallel computer we can do a lot better. Let's suppose that n is a power of 2, so that n = 2^m. Now suppose we have a computer with at least n/2 separate processors. We can then calculate n! in only

m = log_2(n) operations

We do this by using a *divide and conquer algorithm.*

In the first stage of this we calculate the n/2 separate products

a1 = n(n-1), a2 = (n-2)(n-3), a3 =(n-4)(n-5), a4 = (n-6)(n-7), a5 = (n-8)(n-9), … , a(n/2) =3.2

These calculations are all done in parallel on the n/2 processors and this takes *one time unit.*

Next we calculate (again in one operation) the n/4 numbers

b1=a1.a2, b2 = a3.a4, b3=a5.a6, ….

We repeat to calculate c1 = b1.b2 c2 = b3.b4 etc etc.

If we do this process m times then we have managed to find n!

To give an example of how fast this is, suppose that we want to use the Met Office computer to calculate 542,288! (I admit this is not a great use of the computer.)

The conventional algorithm takes 542,288 calculations, whereas the parallel calculation takes only 19. This is a speed up of 28,541 times.

Multiple core computers reach their full potential only when they run in parallel at full speed. Some kinds of tasks, such as the inversion of large-scale matrices by iterative methods allow a lot of parallel processing and this speeds things up a great deal. Special computer codes have been written especially to make this possible. But many kinds of operations don't. These are the sort of calculations in which there has to be a lot of communication between the different parts of the calculation. This process is called *message passing*, and the effective construction of codes to do this by dividing up the calculation is called *load balancing.* At present the development of effective algorithms to do this is a bottle neck to progress of scientific computing. It is an area I work in myself, and which mathematics has much to contribute to.

A significant, and relatively recent advance, which has made scientific computing a lot easier, has been the invention of dedicated programming languages, such as Matlab and Python, to express scientific computations in simple terms. Such languages have hard wired and very fast algorithms for specific mathematical calculations which can then be used very easily. For example to solve the matrix eigenvalue problem in Matlab you simply type

- S = eig(A)

This compares with several lines of code, together with a call to a subroutine library, which would be needed when using a language such as FORTRAN. Similarly, all of these languages make plotting the answers (even as movies) much easier than before. The development of such languages makes scientific computing much more accessible to the non-specialist and, to my mind, is every bit as important in terms of saving time and improving productivity, as any other recent advance in computing.

5.3.3 Peta scale computing and beyond

Both the hardware and the software developments described above are incorporated into the current generation of scientific super computers.

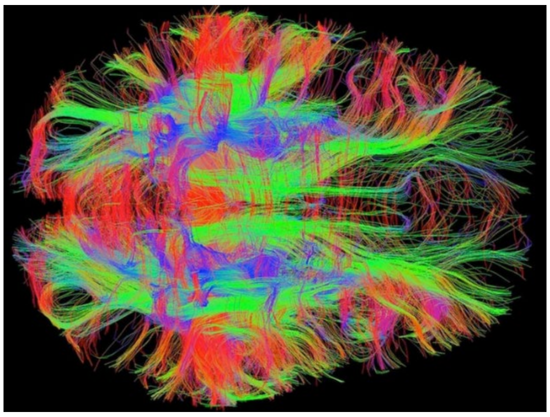
A Peta is 10^{15} and the three Met Office Cray computers are example of a Peta scale computers. This means that they are capable of over 14 Peta arithmetic operations per second (14 Peta flops), or in other words 14, 000, 000, 000, 000, 000 floating point calculations per second. Compare this with the Cray 2 which I was using 30 years ago. Similarly it has 2 Peta bytes of RAM memory and 24 Peta bytes of storage.

This power allows the Met Office to take in 215 billion weather observations from all over the world every day, which it then takes as a starting point for running an atmospheric model with many billions of unknown variables. This allows both more accurate, and faster, weather and climate forecasts

The first Peta scale computer went online in 2008 and there are now around twenty Peta scale computers currently in use. One of which is the Oak Ridge Jaguar supercomputer illustrated below.

In general Peta scale computing is being used to do advanced computations in many fields such as weather and climate simulation, nuclear simulations, cosmology, genomics, quantum chemistry, medical imaging, remote sensing, space flight, lower-level brain simulation, molecular dynamics, drug design, aerodynamics, and the science behind fusion reactors. Below are some examples of weather and climate calculations obtained by solving the Navier-Stokes equations by using a discretisation process very similar to that used for the Kings-Cross fire.

And here is a computation of brain activity.



To see more about what is possible with such large-scale computing read [9] (from which the above images are taken).

The next stage in this development will be *Exa scale computing*, which is 1000 times faster. Exa scale computing would be considered to be a significant achievement in computer engineering, for it is estimated to be the order of processing power of the human brain at a neural level. It is, for instance, the target power of the Human Brain

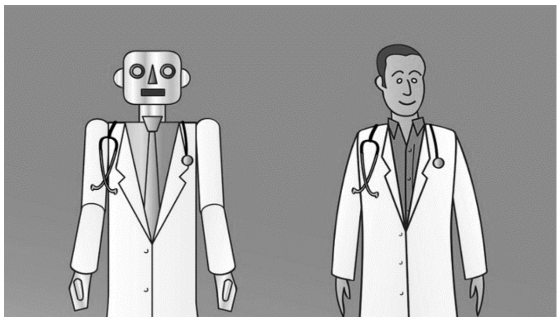
Project. So far this has been achieved (once in 2018) at Oak Ridge National Laboratory which performed a 1.8 exaflop calculation on the Summit OLCF-4 Supercomputer while analysing genomic information in 2018. According to the national plan for the next generation of high performance computers, China will develop an Exa scale computer during the 13th Five-Year-Plan period (2016–2020) project, which is planned to be named Tianhe-3.

Zetta scale computing will the next 1000 fold increase in computing power. It is estimated that this may happen in 2030. Watch this space!

This lecture is called the Mathematics of Future Computing. It is clear to me that a major impact on humanity will come through the combination of mathematics and large-scale computing as described above. Certainly, this is one of the best ways that we will see advances in climate science, genomics, maybe this will allow us to calculate the weather two weeks in advance, and possibly even predict the behaviour of cities or humans in large crowds. However, as with all things, I must urge a bit of caution. The larger the computation, the easier it is for it to go wrong. This we must always be careful to make sure that the algorithms developed to do these tasks are fully supported by mathematical theory.

6. The rise of the algorithms and the role of computers in modern life

Whilst the history of computing as described above, from Babbage to the present day, has been one of steady advance in the solution of scientific problems, it is now fair to the real impact it is making on our lives now is through the use of algorithms which we access through our home computers. By definition the Internet would not be possible without computers, and the mathematical algorithms that they rely on. These are used not only to control the operation of the Internet, typically through the use of queuing algorithms, but also to search it for information. As I said above, every time that you use Google you are solving a matrix eigen value problem. The information on the Internet is (as I described in my earlier lecture 'Maths is coded in your Genes') transmitted without error using mathematically based algorithms, in a compressed way, which makes such transmission quick and efficient. When we pay for any form of goods on the Internet, we use an encryption algorithm, which transmits our credit card details securely. This algorithm (usually some sort of public key encryption) is also based on sound mathematical ideas.



Whilst the computer acting through the Internet has certainly transformed our lives, I suspect that we will see even bigger changes coming with the widespread use of machine learning algorithms. I described these in the lecture 'Is a mathematician a robot'. A machine learning algorithm is one which can be trained to do a task by looking at past examples, and which can then be used to do similar tasks in the future. Examples of machine learning include relatively benign examples such as playing chess and Go, speech recognition, image recognition, weather forecasting, and recommendations for books on Amazon. But the uses of machine learning also include medical diagnosis, dating sites, personnel recruitment, driverless cars, and even making legal judgments. Potentially we are looking at a future without doctors, car drivers or judges. This will radically change our society.

Machine learning is based on mathematical ideas such as function approximation, data analysis, statistical inference, linear algebra, and nonlinear optimization. These are then combined with computer architectures such as Convolutional Neural Networks (CNNs) to give the algorithms described above. These algorithms are then frequently used as 'black boxes' to make decisions. Unfortunately, little is known about how these decisions are actually arrived at, and what forms of bias the machine might have used to come up with them. Would you really want a machine to replace a doctor for example? A great deal of mathematical work has to be done in these areas before we should feel at all confident about using a machine learning algorithm for anything other than routine tasks. I see this as a big aspect of the mathematics of future computing.

7. Quantum computing and other developments

Quantum computers are often given as the key area for future computing development. They can offer vast increases in computing power, at least in theory. Adding just one "qubit" doubles a computer's power, so in principle they could grow much faster than Moore's law would predict. So far they are just a laboratory curiosity, but it is possible that they will become very important for real computations very soon. I described some of this field in my earlier lecture 'The quantum mathematician' and I'll summarise some of this here.

The field of quantum computing was initiated by the work of Paul Benioff and Yuri Manin in 1980, Richard Feynman in 1982, and David Deutsch in 1985. Quantum computers could eventually allow work to be done at a speed almost inconceivable today. As an example of how significant this is, let's consider the fact that most of our money is held in banks. The security of the world's banking system relies on the rather abstract concept of the difficulty of factorising large integers. Whilst it is easy factorise a number such as 143 it is harder to factorise a number such as 262417 and very hard to factorise 97605751. Try doing these yourself. In general if a number has N digits then the best factorisation algorithms on a conventional computer take a time proportional to the exponential of N. If N is large, then this is very large indeed. The reason that this matters is that modern cryptography systems rely on the celebrated RSA algorithm to deliver the key used to encrypt or decrypt a message. Such systems are used to encode information about your credit card and bank account. The RSA algorithm relies for its security on properties of prime numbers, and in particular to the current fact that it is possible to let everyone know the product N of two of these numbers p and q, whilst keeping the values of p and q secure. If a computer could be made which could factorise N in a much shorter time, say in a time proportional to a power of N (called *polynomial time*) then the security of the RSA algorithm would be fatally compromised. A quantum computer works by operating on qubits using quantum gates. Quantum computer algorithms have a different computational complexity than classical algorithms. The most famous example of this is *Shor's factoring algorithm* developed in 1994 [10] (which in turn uses the quantum Fourier transform or QFT) for finding the factors of the number N. It is fast because it relies heavily on the ability of a quantum computer to be in many states simultaneously, and to compute the period of a certain function *f*, which is a necessary part of the algorithm, it evaluates the function at all points simultaneously. As a result of this speed up. Shor's algorithm can factorise a number with N digits in *polynomial time*, which is far faster than a classical algorithm. In 2001, a group at IBM, factored 15 into 3×5, using Shor's algorithm on a quantum computer with 7 qubits. In 2012, the factorisation of 21 was achieved, and in November 2014, adiabatic quantum computation had also factored 56153. So quantum computers are on their way. Besides factorisation and computing discrete logarithms, quantum algorithms offering a very significant speedup over the best-known classical algorithms, have been found for several important problems. These include the simulation of quantum physical processes from chemistry and solid-state physics, some NP-hard problems such as the travelling salesperson problem, and possibly even to artificial intelligence. Indeed, since chemistry and nanotechnology rely on understanding quantum systems, it is thought that the simulation of chemical processes involving quantum mechanics will be one of the most important applications of quantum computing.

As of today, the development of actual quantum computers is still in its infancy (for example we are a long way from factorising the really large numbers used in cryptography), but as we have seen experiments have been carried out in which quantum computational operations were executed on a small number of quantum bits. The problem that all current quantum computers face is the de-coherence of the super positions of the qubits before the algorithm is complete. Practical and theoretical research continues fast, and many national governments and military agencies are funding quantum computing research in additional effort to develop quantum computers for civilian, business, and national security purposes. A small 20-qubit quantum computer exists and is available for experiments via the *IBM-Q quantum experience* project.

Exciting times certainly lie before us with the increased power of computing. But in case we ever get complacent I leave you with one final quote from the great John von Neumann who said in the late 1940s.

It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years.

References

[1] C. Hollings, U. Martin and A. Rice, *Ada Lovelace: The making of a computer scientist*, (2018), The Bodleian Library

[2] R. Flood, *Gresham Lecture on Babbage and Lovelace*, (2016),

https://www.youtube.com/watch?v=1TprEGyo9ts

[3] R. Flood , *Gresham lecture on Turing and Von Neumann*, (2016), https://www.youtube.com/watch?v=fJltiCjPeMA

[4] Wikipedia article on Turing machines.

https://en.wikipedia.org/wiki/Turing_machine

[5] J. von Neumann *First draft of a report on the EDVAC*

https://en.wikipedia.org/wiki/First_Draft_of_a_Report_on_the_EDVAC

[6] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*, (1996), Cambridge.

[7] AEA Technology simulation of the Kings Cross Fire, (1987), https://www.youtube.com/watch?v=6cE9Ud6GgE4

[8] Press cutting of the installation of the Cray 2 at Harwell

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5187997

[9] D. A. Bader (ed.), *Peta scale Computing: Algorithms and Applications*, (2008), Chapman & Hall/CRC Computational Science.

[10] Shor's factoring algorithm

https://en.wikipedia.org/wiki/Shor%27s_algorithm