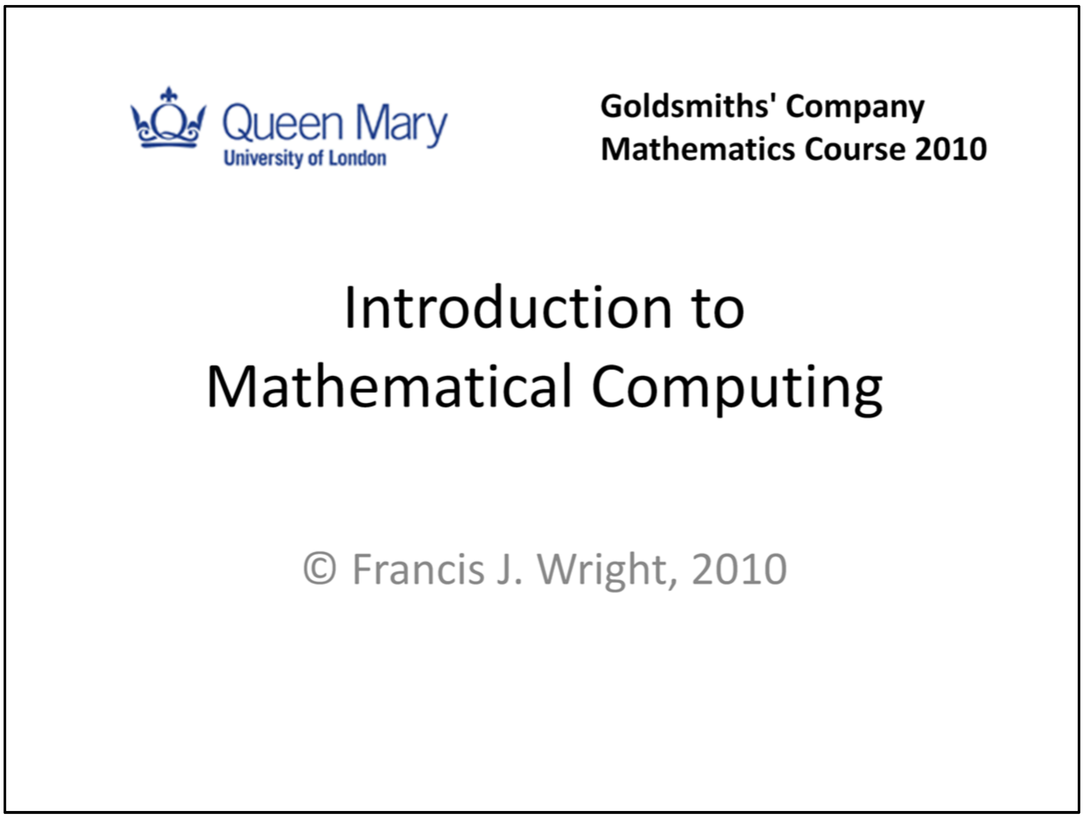


Queen Mary University of London

Goldsmiths' Company Mathematics Course 2010

Introduction to Mathematical Computing

Try to set up the 4-minute video clip at www.flixxy.com/babbage-mechanical-calculator.htm ready to show later.

This will be a broad introduction to some key landmarks in mathematical computing leading to a hands-on workshop to introduce Maple, a powerful mathematical computing system.

Images taken from the web are hyperlinked to their sources.

The diffraction catastrophe images are from
"The elliptic umbilic diffraction catastrophe" by M. V. Berry, J. F. Nye, F.R.S. and F. J. Wright, *Phil. Trans. R. Soc. Lond.* A **291**, 453–484, 1979.

What is a computer?

What does a computer look like these days?

Apart from the obvious desktop, laptop and netbooks computer, many modern appliances are essentially computers in different packaging; some even have wheels! Digital phones, digital TVs and radios, and games consoles are specialized digital computers. Modern cars have many subsystems that are specialized digital computers, such as electronic engine management, traction control, navigation, and security systems.

But probably only digital mobile phones are sufficiently general-purpose to be used for mathematical computing. Many already have a calculator of some description and they could run more sophisticated mathematical software. (TI-Nspire, developed from Derive, currently runs on a pocket calculator.)

Analogue versus digital: Clocks

There are two key computing paradigms – **analogue** and **digital** – which are well illustrated by clocks. **Is the pendulum clock analogue or digital?**

It really depends on your definitions.

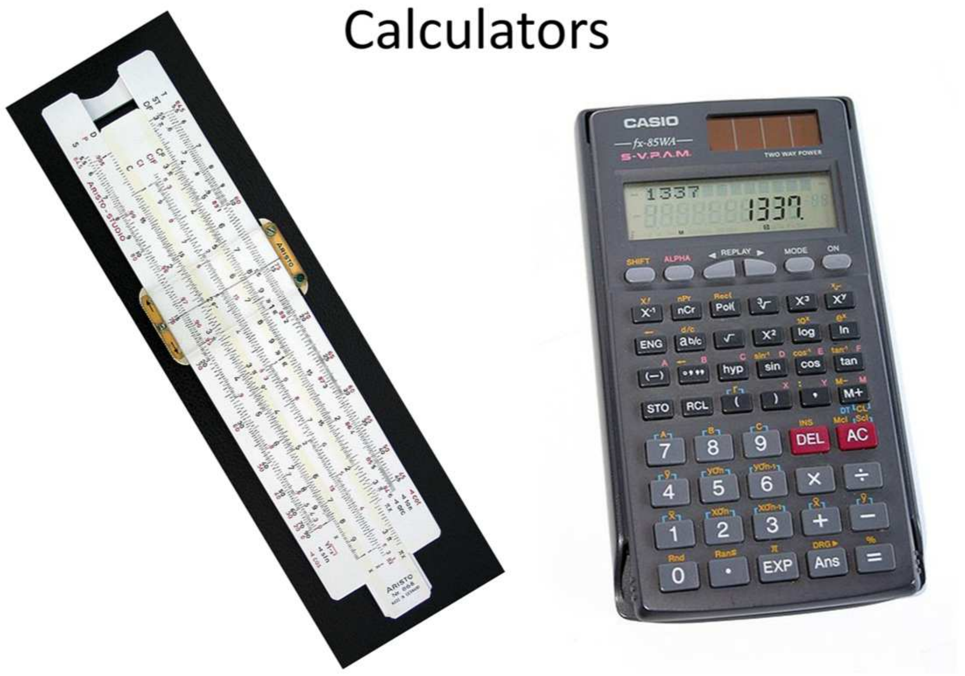
Analogue computing relies on representing the system being investigated by a more convenient **analogous** physical system that is governed by the same mathematics, e.g. the same differential equations. The apparent motion of the sun around the earth is an analogue of the flow of time, so a sundial is an analogue clock, as are water and sand clocks (egg timers).

Physical systems are continuous and smooth, so analogue has also come to mean continuous and smooth, or non-digital.

Digital computing relies on **deconstructing** the system being investigated into a discrete collection of subsystems that are investigated in turn and the results used to **reconstruct** the behaviour of the complete system. A digital clock is a simple example that just counts and displays time steps.

The two clocks on the left are very similar and both rely on an oscillator to provide an analogue of time and a device to count the oscillations. So digital computing relies on an analogue of time that is used to synchronise analysis and synthesis steps.

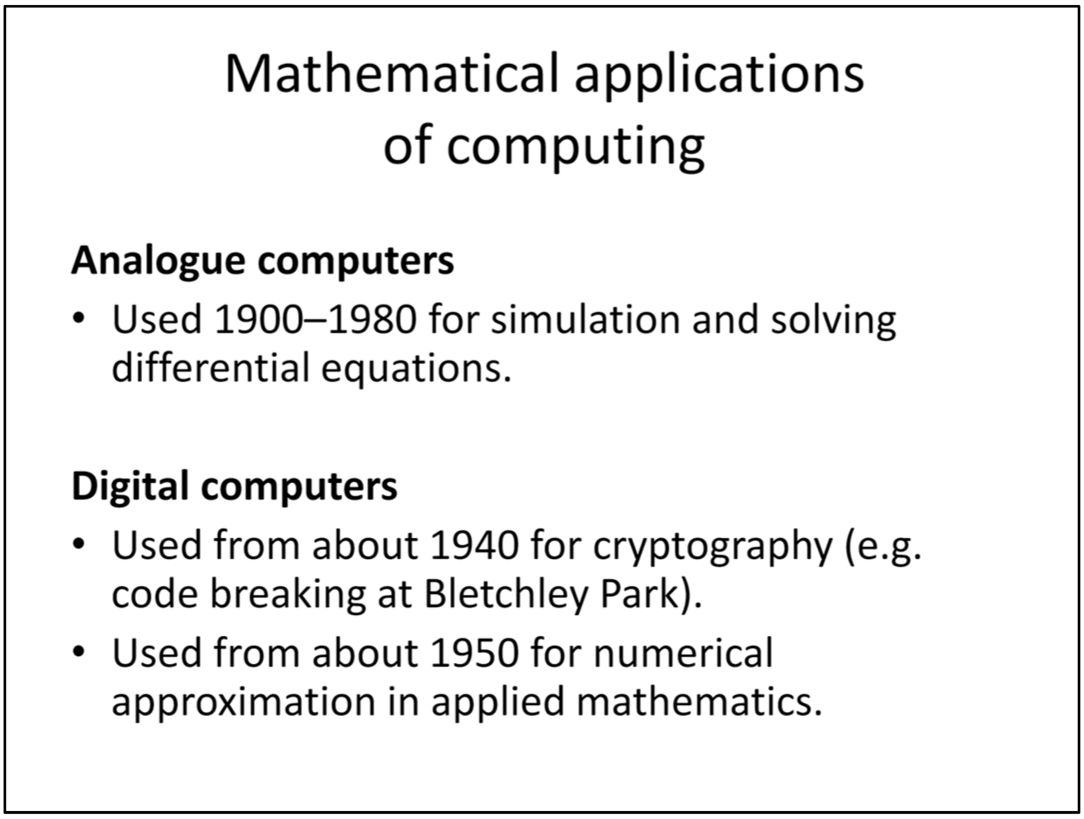
Analogue versus digital: Calculators



The first computers were analogue but digital computers have now completely taken over. This transition is well illustrated by hand-held computing devices.

A slide rule is a good example of an analogue calculator. Addition of physical distances provides an analogue of multiplication of real numbers via a logarithmic mapping and the slide rule relies on addition of logarithmic scales.

The digital calculator offers much better accuracy and flexibility and took over from the slide rule in the mid 1970s.



Mathematical applications of computing

Analogue computers

- Used 1900–1980 for simulation and solving differential equations.

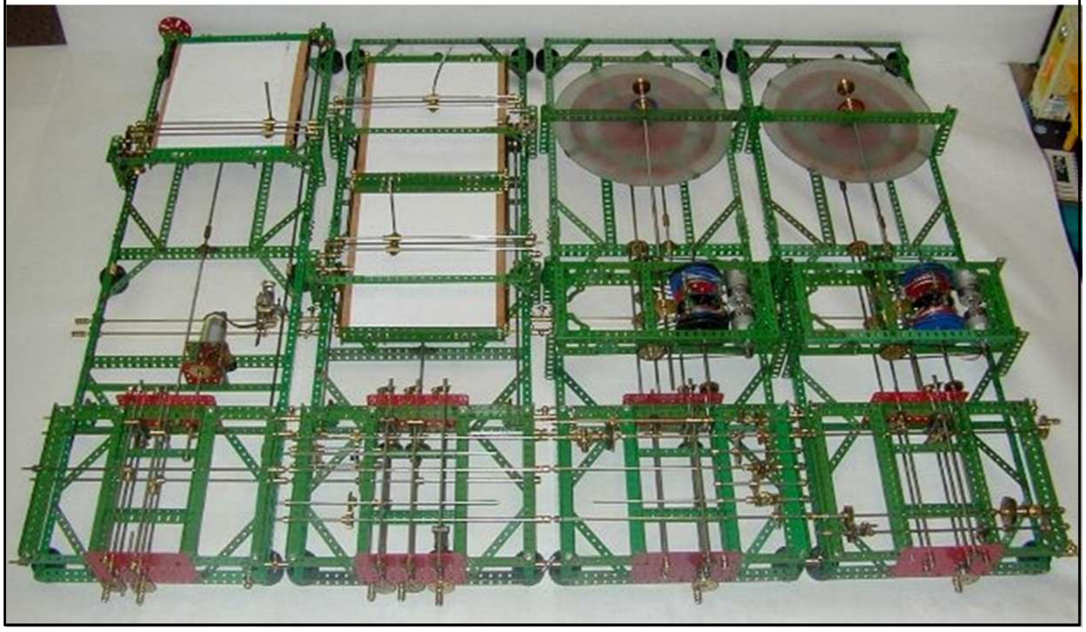
Digital computers

- Used from about 1940 for cryptography (e.g. code breaking at Bletchley Park).
- Used from about 1950 for numerical approximation in applied mathematics.

Analogue computers are limited to mathematical modelling by solving differential equations.

Digital computers are much more flexible and progressively took over from analogue computers during the second half of the last century. Initially their mathematical uses were mainly numerical approximation.

Analogue computers: The differential analyser



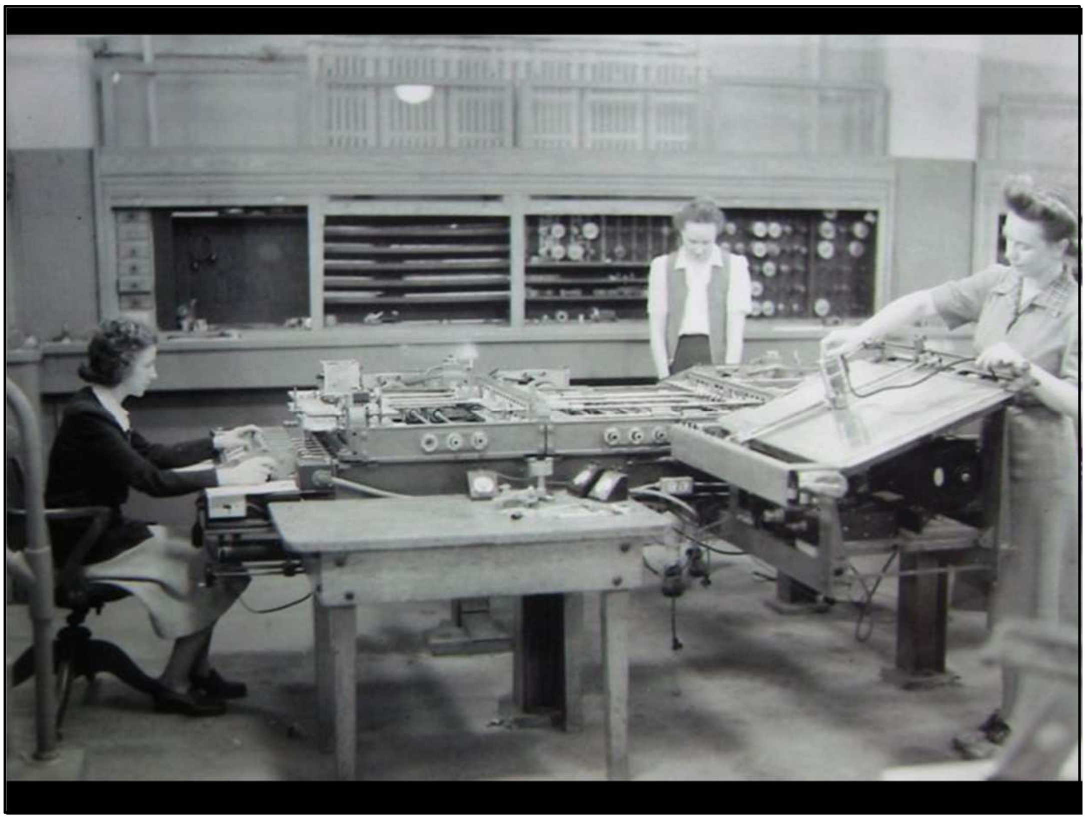
Let’s begin with a brief history of computers.

Computer technology has evolved from mechanical devices to valves, transistors, simple integrated circuits and finally very-large-scale integrated circuits with over a billion transistors per chip.

Let’s start with analogue computers.

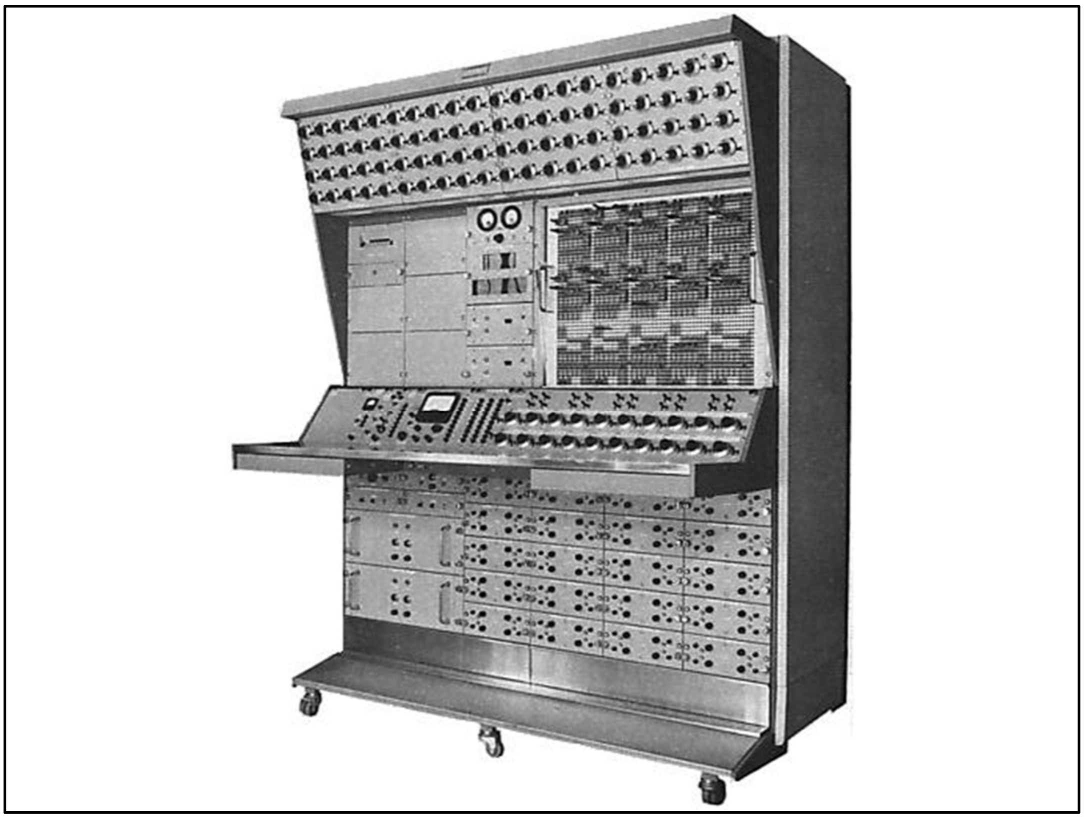
The **Differential Analyser** solves differential equations by **integration**. It was invented in 1876 by James Thomson, brother of Lord Kelvin. The discs are mechanical analogue integrators. The rotation of the small pick-up wheel that touches the disc surface is the integral of its distance from the disc centre as a function of angle (and hence time), since d(arc length) = radius × d(angle). Torque amplifiers are essential to make this work; they are the blue cylindrical devices this side of the discs.

This is a Meccano Differential Analyser built by Tim Robinson of California, USA. [www.dalefield.com/nzfmm/magazine/Differential_Analyser.html]



This is a differential analyser in operation circa 1942–1945. They were used to compute shell and bomb trajectories during the war.

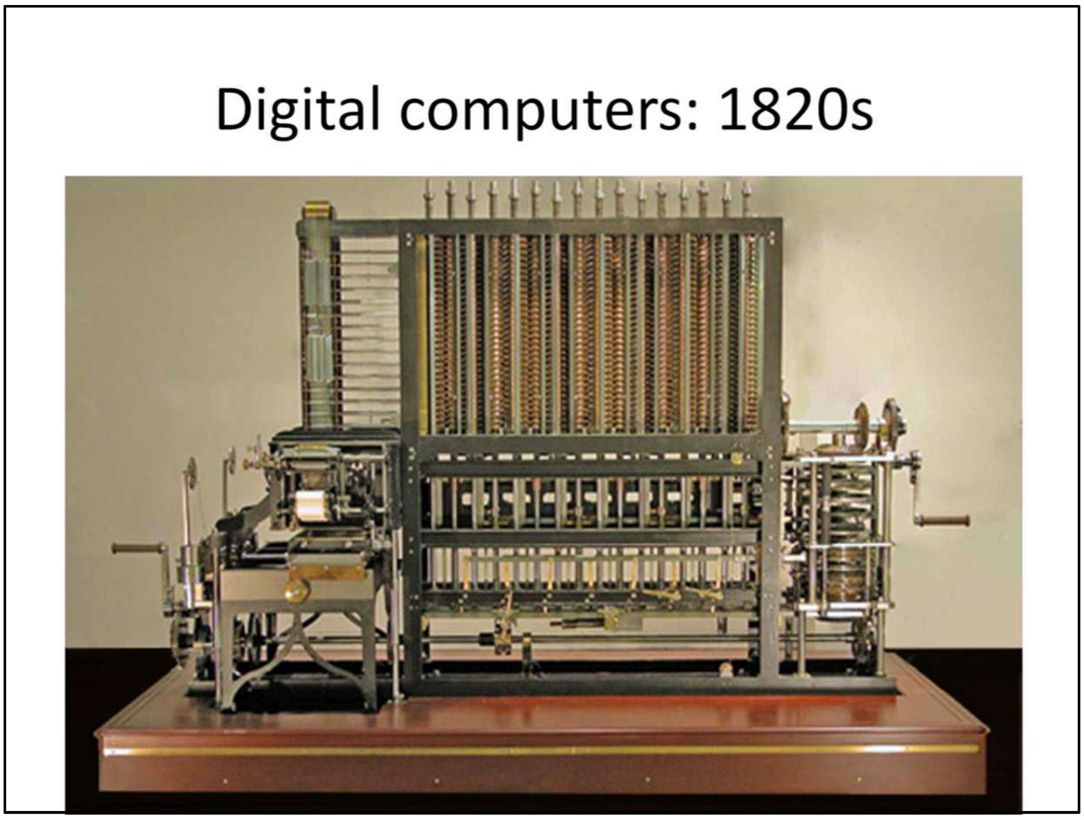
[http://en.wikipedia.org/wiki/Differential_analyser]



Later analogue computers were electronic: they were programmed by connecting differentiators, integrators, etc, using "patch cables“ as used in manual telephone exchanges. This one dates from around 1960.

The behaviour of different types of electronic circuits – resistive, capacitative, inductive, etc. – provide analogues of other physical systems.

A PACE 16-231R analogue computer. Photograph published in 1961. [www.dself.dsl.pipex.com/MUSEUM/COMPUTE/analog/analog.htm]



Digital computers: 1820s

The first digital computers were mechanical and this is **Babbage's Difference Engine**.

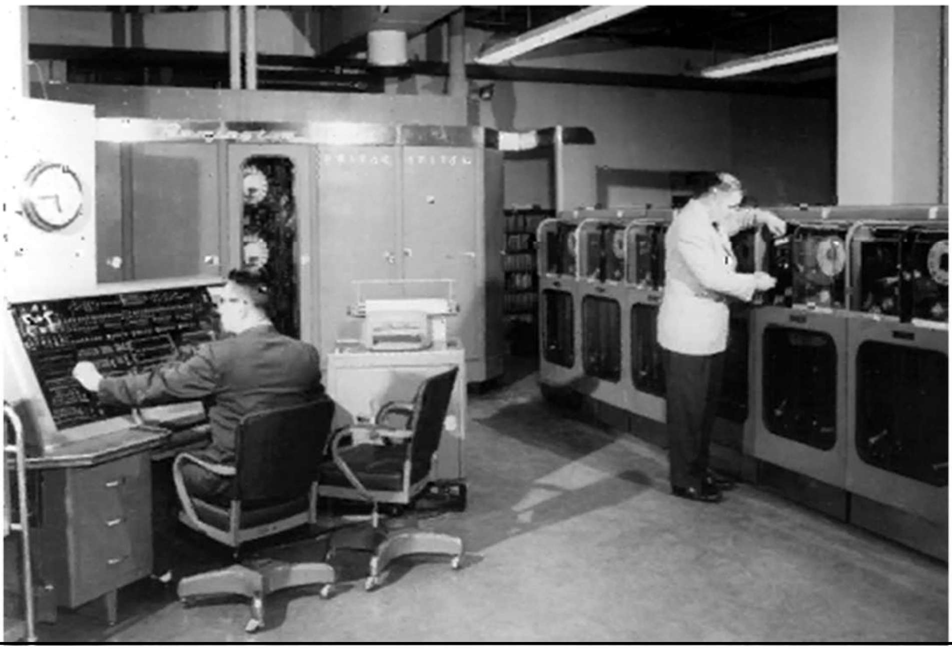
Charles Babbage (1791-1871) completed the design for this computer in 1849 but engineering technology was not adequate to build it. It was finally built in 2002 for the London Science Museum. It consists of 8,000 parts, weighs 5 tons, and measures 11 feet long.

The engine is operated by a crank handle and can evaluate trigonometric and logarithmic functions with 31 digits of precision. Its printer (on the left) stamps the results on paper and on a plaster tray, which could be used to create lead type for printing books of mathematical tables.

Ada Lovelace, daughter of the famous poet **Lord Byron**, published a detailed study of the difference engine and is credited today as the first computer programmer.

See the 4-minute video clip at www.flixxy.com/babbage-mechanical-calculator.htm.

Digital computers: 1950s



The first practical digital computers were electronic. They use stored programs: the so-called von Neumann architecture in which instructions and data are equivalent. Prototype electronic computers were built from about 1939 using **valves** or **vacuum tubes**.

We will see a British example, Colossus, built around this time at Bletchley Park.

This is the first commercially available electronic computer, **UNIVAC I**, designed by **Eckert** and **Mauchly**. The central computer is in the background, and in the foreground is the control panel. Remington Rand delivered the first UNIVAC machine to the US Bureau of Census in 1951.

Picture from THE BETTMANN ARCHIVE.

"UNIVAC Computer System," Microsoft(R) Encarta(R) 97 Encyclopedia. (c) 1993-1996 Microsoft Corporation. All rights reserved.

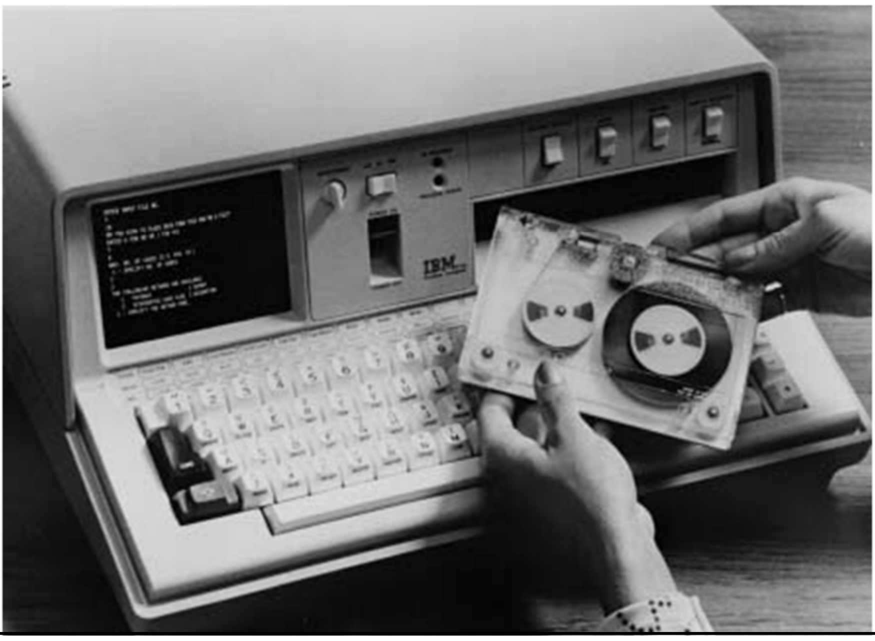
Digital computers: 1960s

Transistors were introduced in the late 1950s and integrated circuits in the late 1960s.

This is the IBM System/360 announced in April 1964 using **solid state technology**. Computers like this were widely used in universities from around 1960 to 1990.

Note the absence of any display screens! They were not available until the 1970s. Input and output was via punched cards and paper tape, teletypewriters and "line printers".

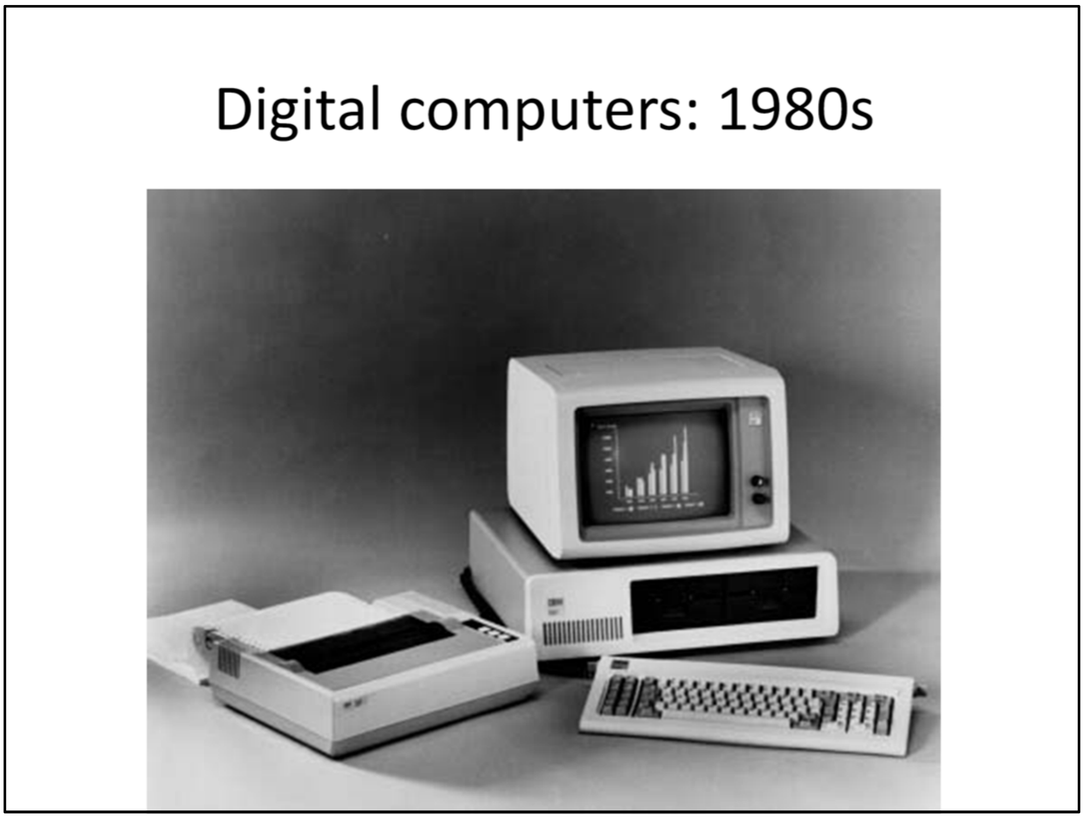
Digital computers: 1970s



This IBM 5100 Portable Computer was sold from 1975 to 1982.

It weighed about 50 pounds and was slightly larger than an IBM typewriter.

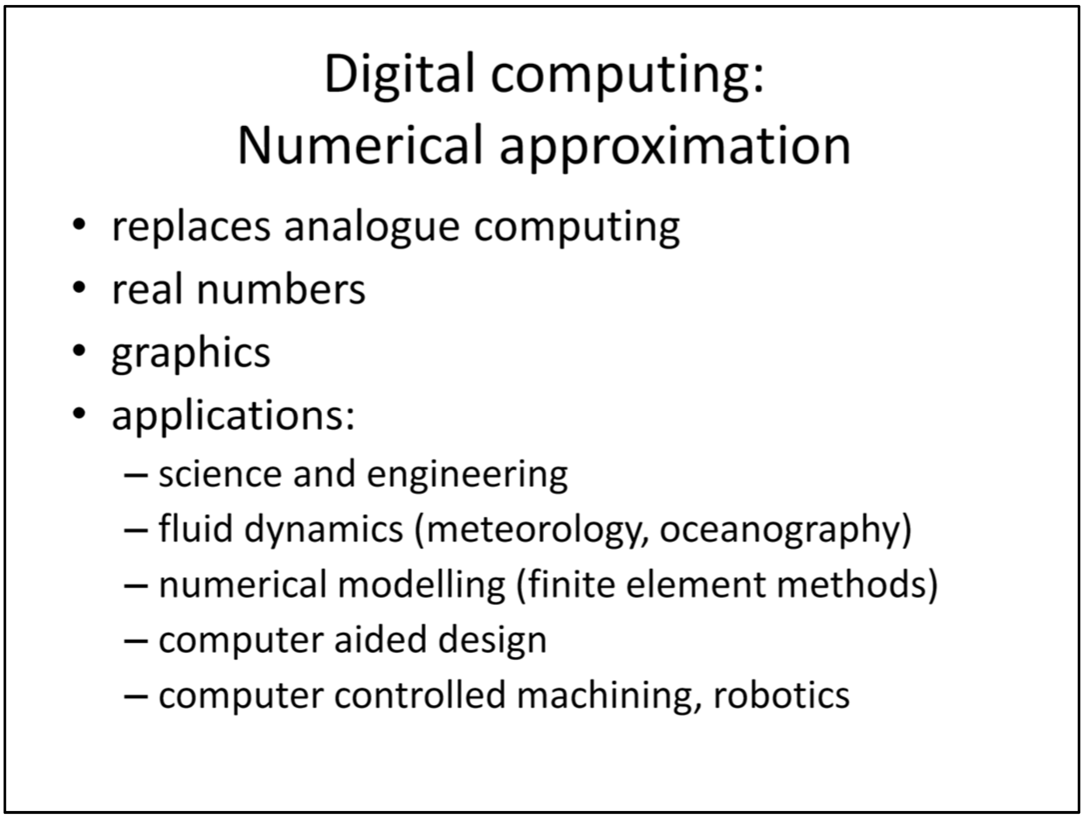
Note the very small display screen.



Digital computers: 1980s

The IBM Personal Computer was introduced in 1981 and heralded the modern desktop computer.

Computing became interactive via the keyboard and screen. Mice and windowed displays were introduced during the 1980s. They were first developed by Xerox at PARC – Palo Alto Research Centre – but first successfully commercialised by Apple with the Mac.



Digital computing: Numerical approximation

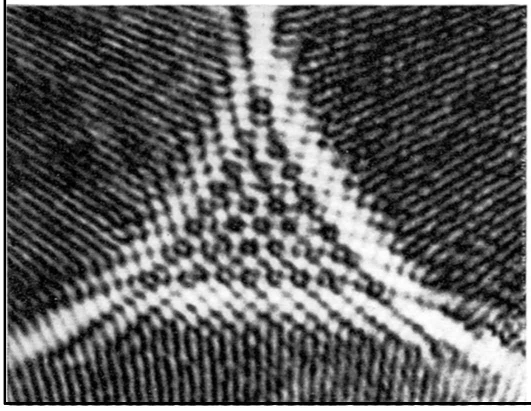
- replaces analogue computing
- real numbers
- graphics
- applications:
  - science and engineering
  - fluid dynamics (meteorology, oceanography)
  - numerical modelling (finite element methods)
  - computer aided design
  - computer controlled machining, robotics

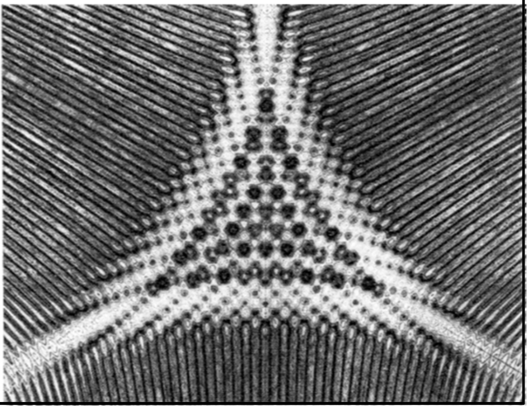
In digital computing there are two main paradigms: **numerical approximation** and **exact or symbolic computing**.

Numerical approximation is used to solve problems involving real numbers and often the results are best displayed graphically (for which quite low precision is sufficient). Computational fluid dynamics and stress analysis using finite element methods are major uses.

Spectacles in the rain!

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i\left\{\xi^3-3\xi\eta^2-z\left(\xi^2+\eta^2\right)-x\xi-y\eta\right\}}\,\mathrm{d}\xi\,\mathrm{d}\eta$$



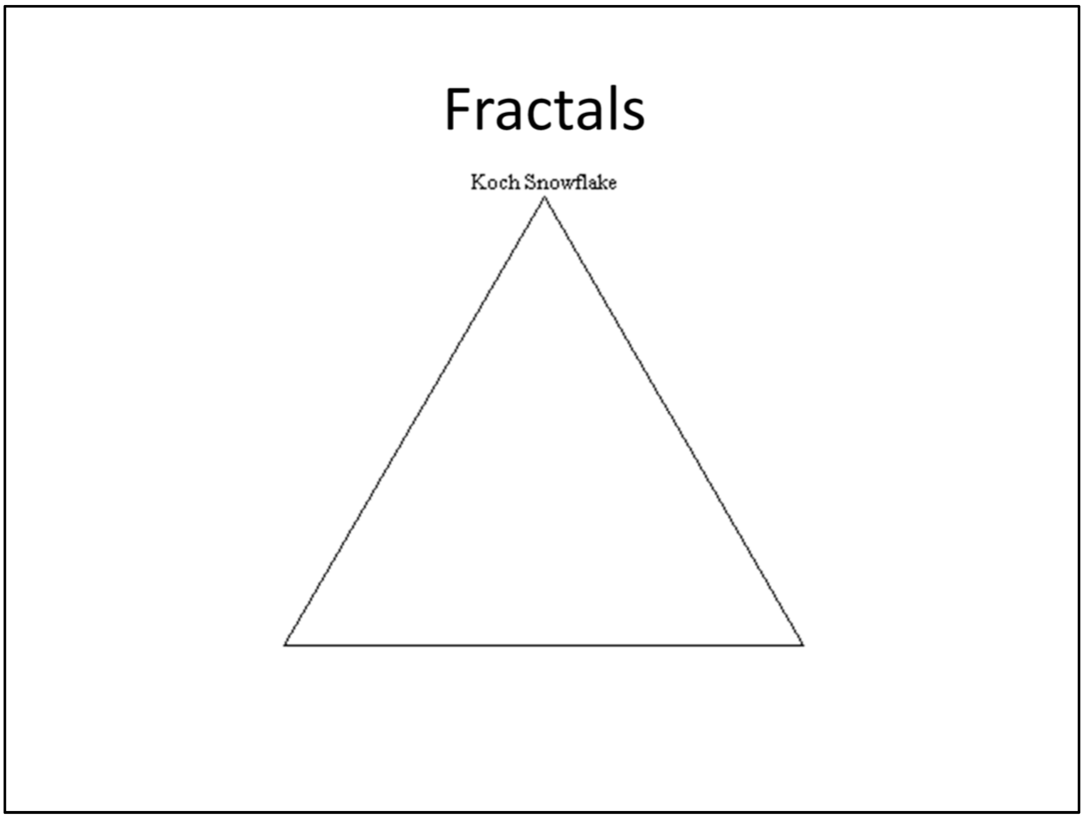


If you wear spectacles in the rain at night and look at a street lamp you can see patterns like that shown on the left, which was photographed through a water droplet using a microscope.

Only a small number of different types of pattern can typically occur, which are described by René Thom’s **Catastrophe Theory**. This is a theory that describes sudden changes in dynamical systems governed by a potential function; here the sudden changes are in light levels. The pattern is caused by diffraction around the classical focus or **caustic** and is called a **Diffraction Catastrophe**.

This diffraction catastrophe is described by the diffraction integral shown, which is a double integral over a highly oscillatory integrand. In general, this integral can only be approximated numerically, which would be necessary to display it graphically anyway.

The picture on the right is the modulus of this complex-valued integral in an (*x*,*y*)-plane at an appropriately chosen value of *z*. I computed this diffraction catastrophe as part of my PhD project in the mid 1970s using a mainframe computer similar to an IBM 360. Using the symmetry, I plotted contours for a 60 degree segment, shaded between the contours by hand, had the result photographed and mirrored, and then glued the pieces together. It would be much easier to do now!



Fractals

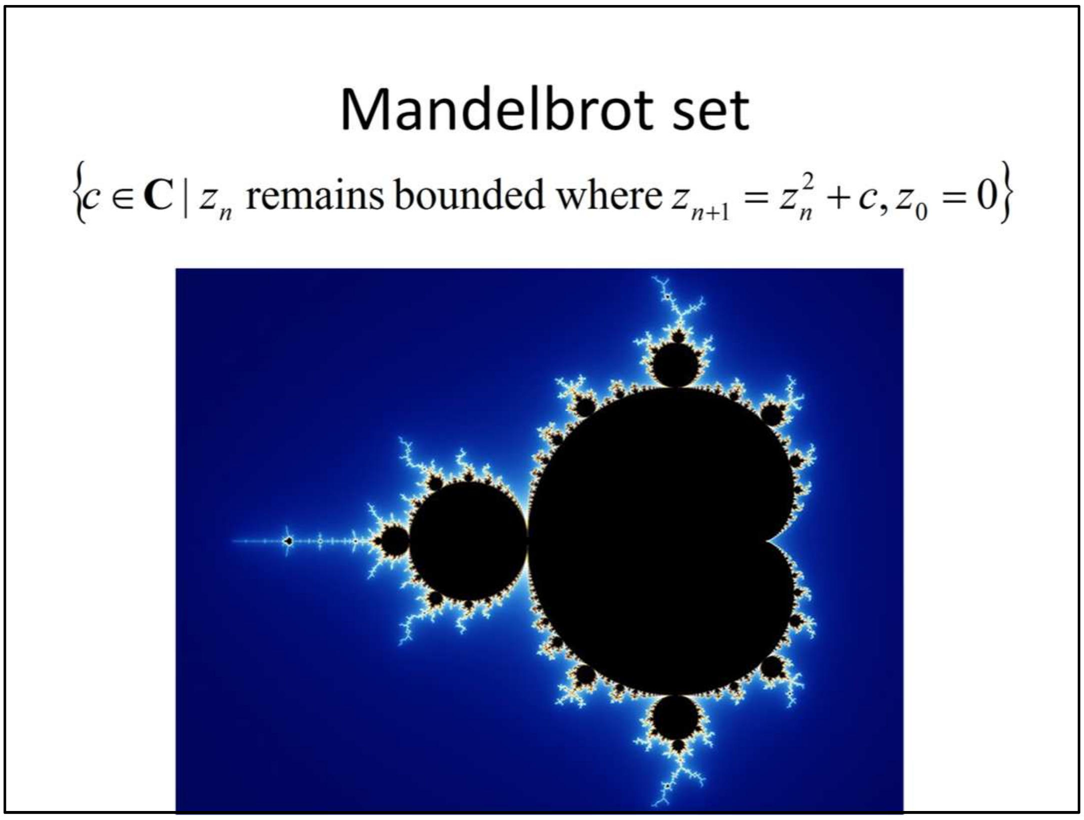
A fractal is a set that has **fractional dimension**. We will hear more about fractals later in the course. This animation shows the first few steps in the construction of the Koch snowflake fractal: continually replace the mid third of each line with an equilateral triangle minus its base.

Although this object was first defined by the mathematician H. von Koch in 1904, it is far easier to construct and explore using a computer than by hand, which applies to all fractals, especially more complicated ones.

Fractals are normally defined recursively, so a recursive procedure is the natural way to plot them.

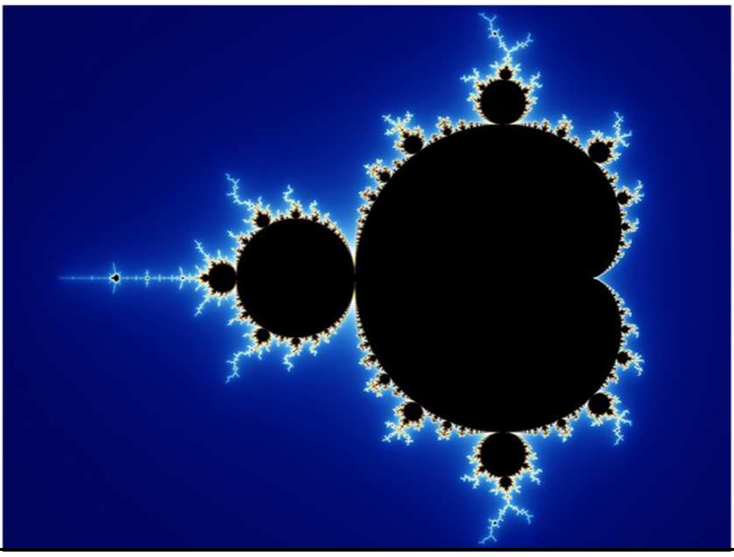
The numerical approximation arises because you have to terminate the infinite construction process and because the graphical display can only be approximate.

This animation is taken from my book *Computing with Maple* (CRC Press, 2001). [http://centaur.maths.qmul.ac.uk/CwM/]



Mandelbrot set

$$\{c \in \mathbf{C} \mid z_n \text{ remains bounded where } z_{n+1} = z_n^2 + c, z_0 = 0\}$$

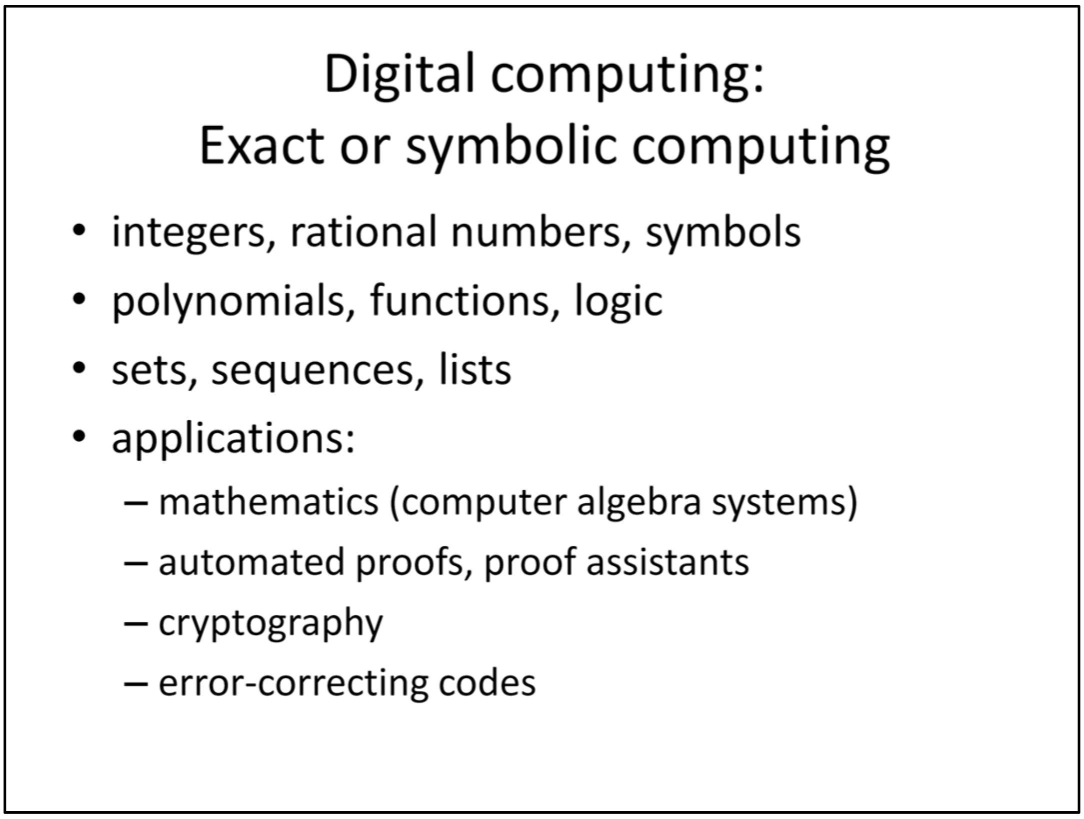


Named after Benoît Mandelbrot, this is the set of complex values of *c* for which the orbit of z_n defined by the iteration $z_{n+1} = z_n^2 + c$, $z_0 = 0$, remains bounded. The boundary of the Mandelbrot set is a fractal. Clearly, when $c = 0$, $z_n = 0$ for all n and so is in the Mandelbrot set, which is shown black here.

[http://en.wikipedia.org/wiki/Mandelbrot_set]

I defy anyone to construct such sets without using a computer!

In fact, Mandelbrot was on the research staff at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, when he did his main work on fractals.

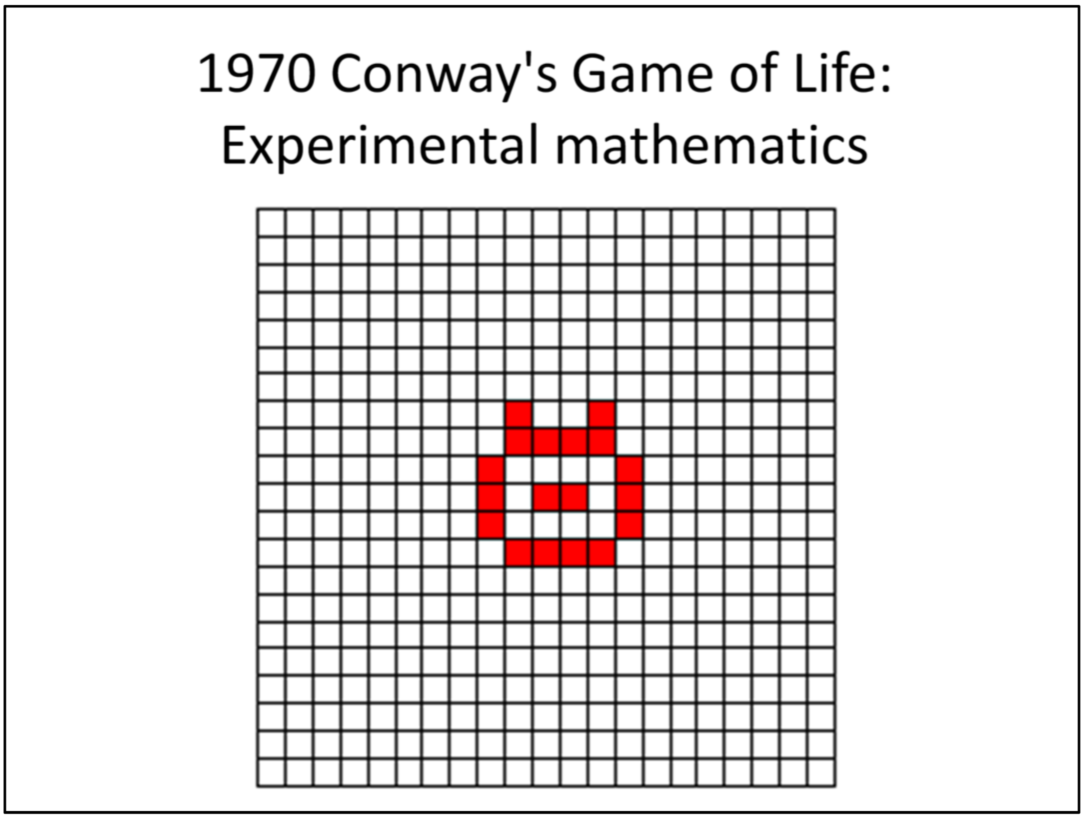


Digital computing: Exact or symbolic computing

- integers, rational numbers, symbols
- polynomials, functions, logic
- sets, sequences, lists
- applications:
  - mathematics (computer algebra systems)
  - automated proofs, proof assistants
  - cryptography
  - error-correcting codes

Exact or symbolic computing is used to solve problems in pure mathematics where approximation is not appropriate. Its most widespread use is in **computer algebra** or **algebraic computing**. This generally requires more computing power than numerical approximation and so did not become mainstream until the 1990s.

We will return to algebraic computing later, but first let me introduce two other exact computing landmarks.



1970 Conway's Game of Life: Experimental mathematics

This is what is now called a **cellular automaton** and was devised by the British mathematician John Horton Conway in 1970. This particular pattern is called the **Cheshire Cat** because the cat face disappears leaving just its grin!

Each cell of the grid is either alive (filled) or dead (empty). The grid is seeded and then at each step (generation), the following changes occur:

•Any live cell with fewer than two neighbours dies (through under population).

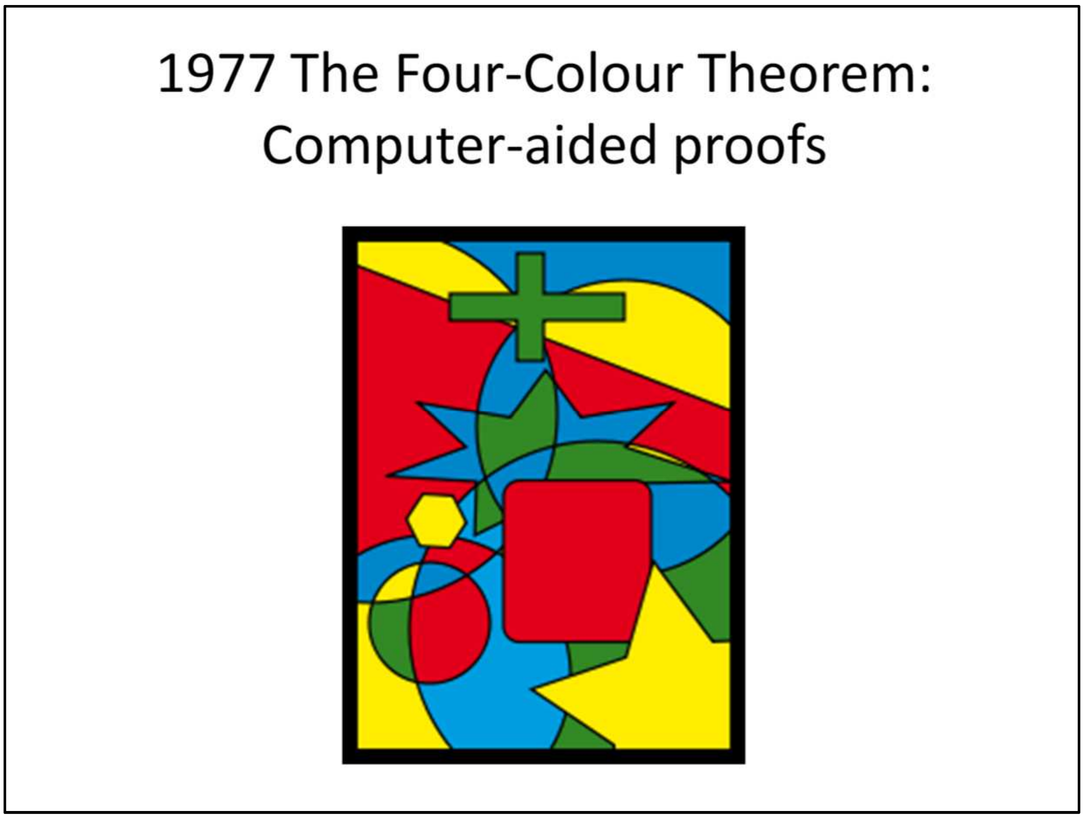
•Any live cell with more than three neighbours dies (through over population).

•Any empty cell with exactly three neighbours becomes a live cell.

The first Game of Life computer program was written by Guy and Bourne in 1970. Without its help some discoveries about the game would have been difficult to make.

[http://en.wikipedia.org/wiki/Conway's_Game_of_Life]

The game of life is easy to implement on modern computers and there are many programs available on the web. I generated this animation using Maple, originally for a third-year undergraduate course on Computational Problem Solving.



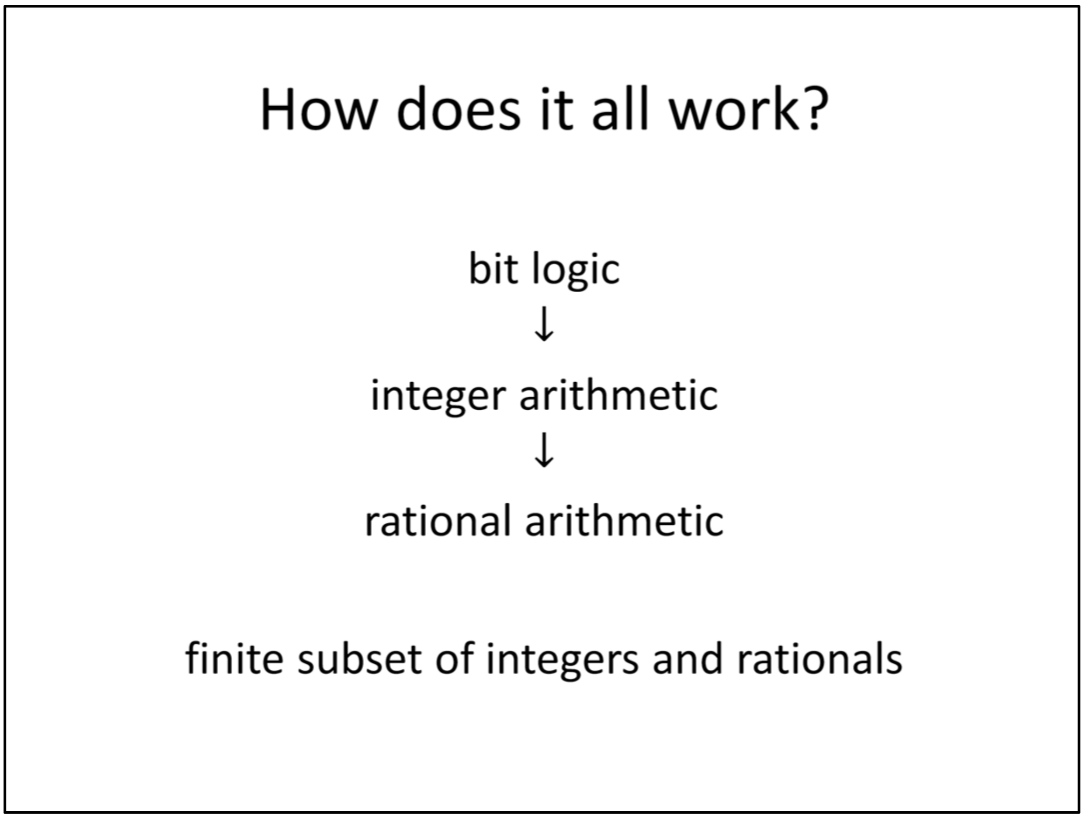
1977 The Four-Colour Theorem: Computer-aided proofs

This states (roughly) that four colours are enough to colour all the regions of any (planar) map and was first conjectured by Guthrie in 1853.

It was not proved until 1976, when Appel and Haken constructed a celebrated proof using an exhaustive analysis of many separate cases by computer. This led to a lot of controversy about what constitutes a proof.

An independent proof was constructed by Robertson et al. in 1997 and verified in 2004 by Gonthier of Microsoft Research in Cambridge using symbolic computing.

[Weisstein, Eric W. "Four-Color Theorem." From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/Four-ColorTheorem.html]



How does it all work?

bit logic

↓

integer arithmetic

↓

rational arithmetic

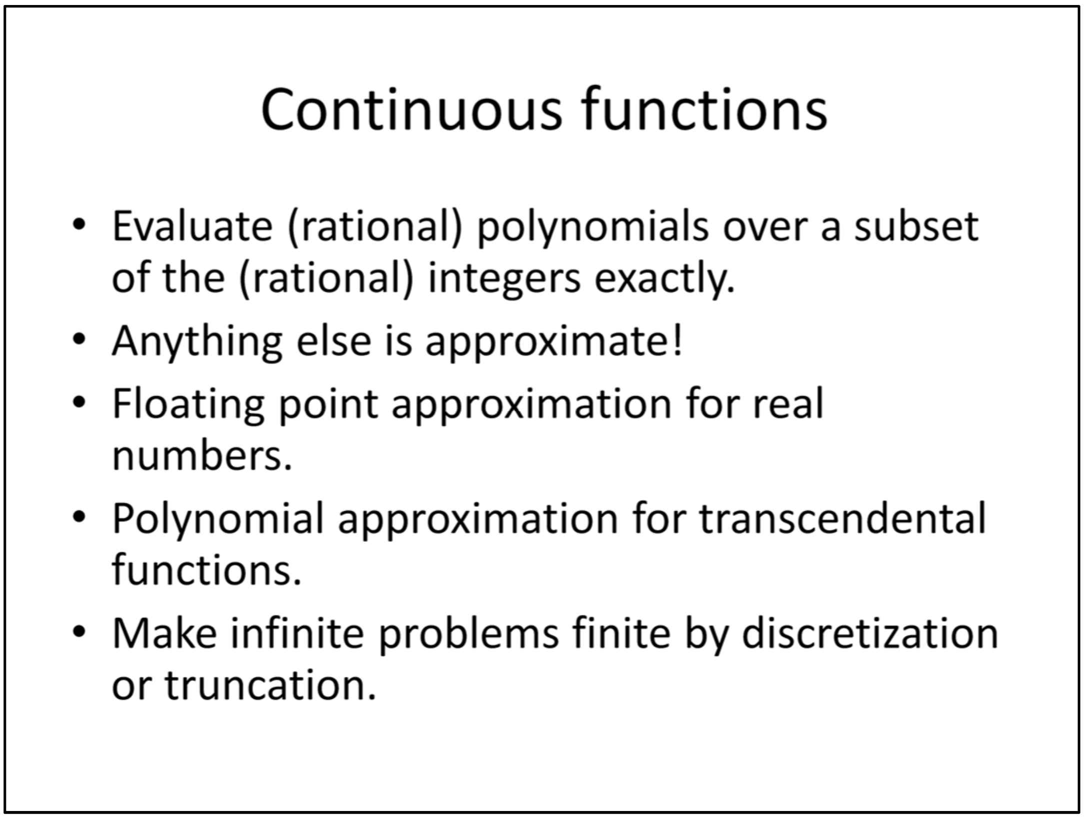
finite subset of integers and rationals

Fundamentally, computers can only perform logical operations on bits (binary digits). Everything else is built on this foundation.

Any integer can be represented in binary, which leads to integer arithmetic.

Any rational number can be represented as a pair of integers, which leads to rational arithmetic.

But the range of integers, and hence rationals, that can be represented is limited by physical memory limitations, so only a subset of the integers and rational can be represented. Fortunately, in modern computers this subset is large enough for most practical purposes.



Continuous functions

- Evaluate (rational) polynomials over a subset of the (rational) integers exactly.
- Anything else is approximate!
- Floating point approximation for real numbers.
- Polynomial approximation for transcendental functions.
- Make infinite problems finite by discretization or truncation.

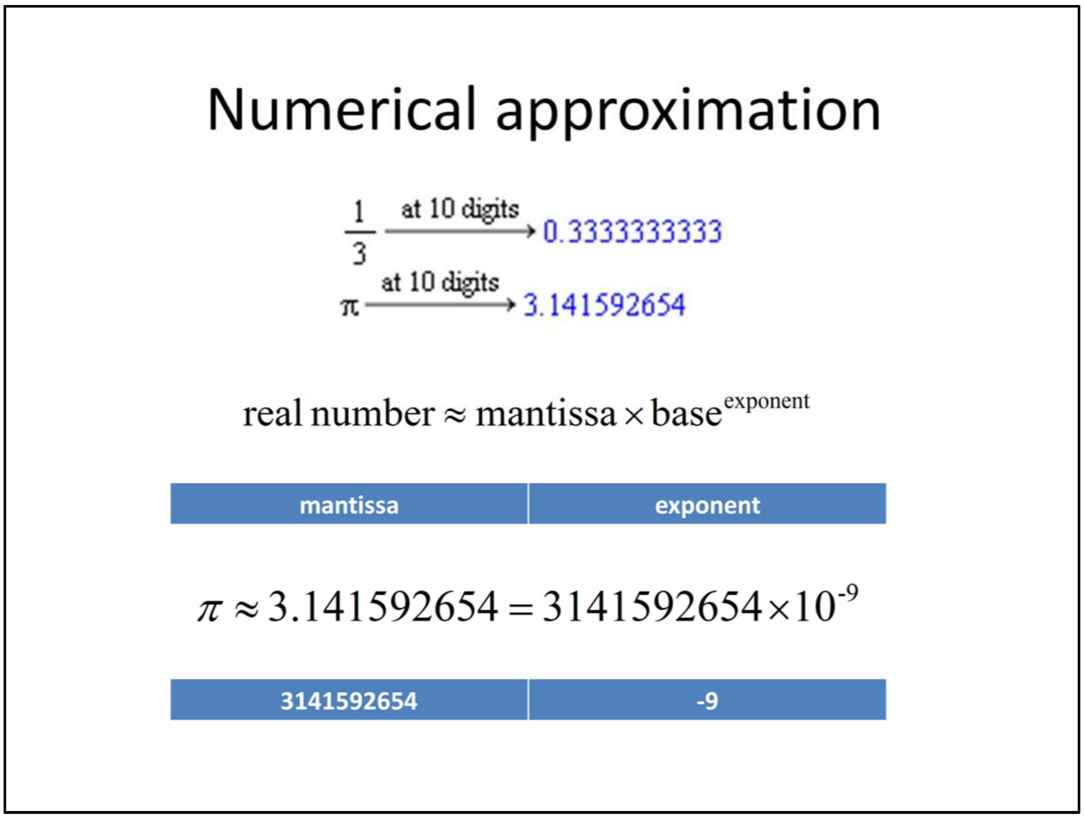
Numerical computing is about computing with smooth continuous functions. But only rational functions over the rational numbers can be evaluated exactly using the four arithmetic operations. Anything else involves approximation!

The continuous set of real numbers is normally approximated by a subset of the rational numbers using a special representation called **floating point**. This approximates a very large magnitude range with a fixed relative error, i.e. a fixed number of significant figures.

For a small number of applications, such as typesetting and graphics, a fixed point representation is used, which approximates a small range of magnitudes with fixed absolute error , i.e. a fixed number of decimal places.

Transcendental (i.e. logarithmic, exponential, trigonometric, etc.) functions are approximated by rational functions.

Continuous processes are discretized, continuous spaces become discrete lattices, and infinite processes are truncated. For examples, integrals become sums and differential equation become finite difference equation.



Numerical approximation

$$\frac{1}{3} \xrightarrow{\text{at 10 digits}} 0.3333333333$$

$$\pi \xrightarrow{\text{at 10 digits}} 3.141592654$$

$$\text{real number} \approx \text{mantissa} \times \text{base}^{\text{exponent}}$$

mantissa	exponent

$$\pi \approx 3.141592654 = 3141592654 \times 10^{-9}$$

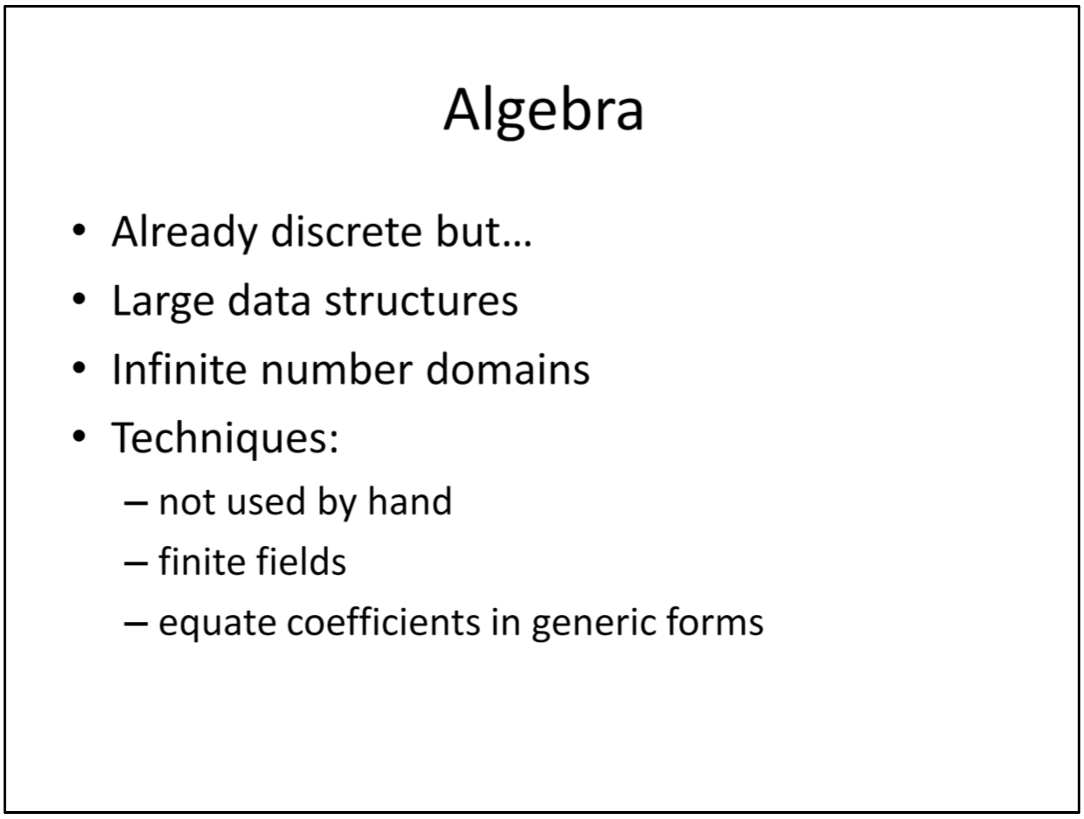
3141592654	-9

Numerical computing usually implies approximation of real numbers and approximate solution of problems involving real numbers. (The extension to complex numbers is trivial.) Generally, you can't prove anything this way, which is why it is not used much by pure mathematicians!

Irrational numbers **must** be approximated and it is normal in numerical computing also to approximate rational numbers. The normal way to approximate real numbers uses **floating point** representation. This is built into modern computer hardware with fixed accuracy, although a similar representation can be implemented in software with variable accuracy. Modern mathematical computing systems, such as REDUCE and Maple, offer software floating point arithmetic with variable and optionally very high accuracy.

Floating point representation uses two integers, the **mantissa** and the **exponent**, and the number of digits in the mantissa determines the number of significant figures in the approximation. In hardware floating point, the mantissa and exponent share one memory cell.

In hardware floating point the base is normally a power of 2, but in software floating point it is often 10, as illustrated here.



Algebra

- Already discrete but…
- Large data structures
- Infinite number domains
- Techniques:
  - not used by hand
  - finite fields
  - equate coefficients in generic forms

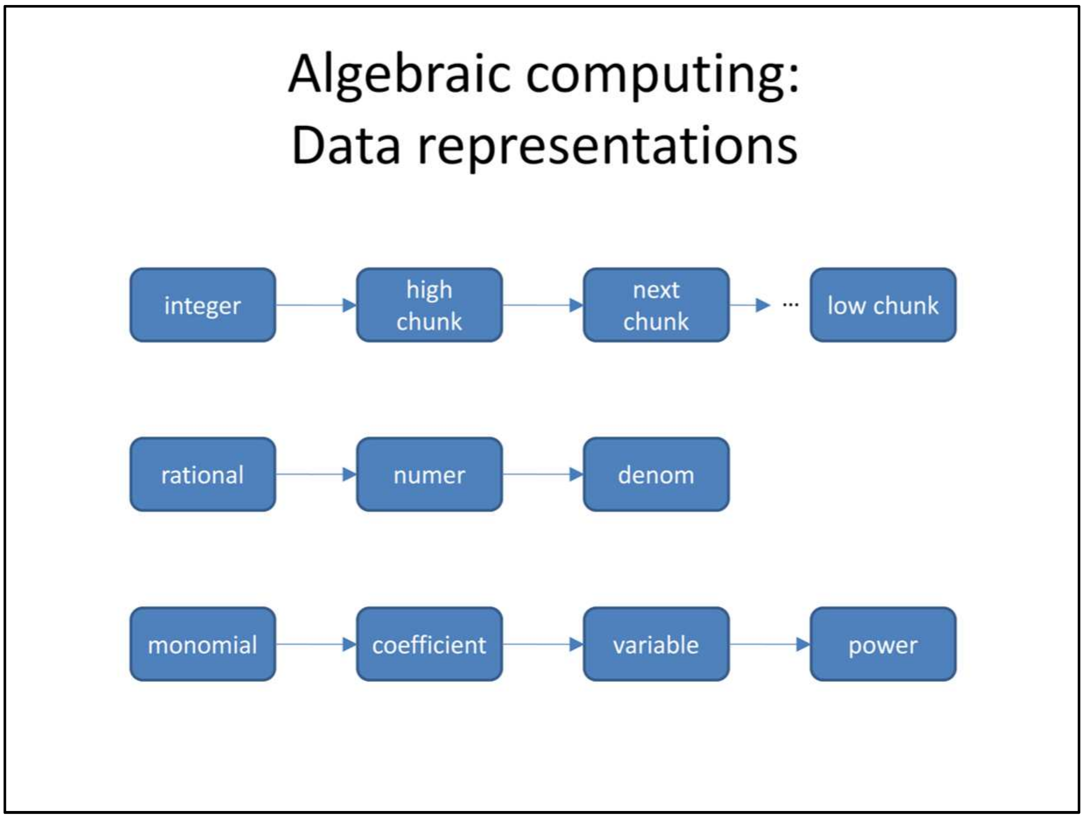
An arbitrarily complicated expression involving numbers with a particular accuracy simplifies to one number of the same accuracy, and hence memory requirement.

But algebraic computation normally means exact computation, in which neither the algebraic structure nor the numerical coefficients can be approximated. An algebraic expression may not simplify at all. So algebraic computation may involve very large data structures.

Some algebraic computation techniques are very different from those normally used by hand. For example, using the integers modulo a prime, which form a finite field.

Algebraic computation usually involves infinite coefficient domains. One way to work with the infinite domain of rational numbers without approximation, used in some factorisation algorithms, is to **project** the rationals to a finite field, solve the problem and then **lift** the result back to the rationals. One lifting technique involves Newton-Raphson iteration, in which the iterates are larger and larger integers representing the solution in larger and larger finite fields, until the field is large enough to include the numerators and denominators of all the exact rational solutions.

Another technique, used for some types of integral, is to construct the most general solution and then equate coefficients.



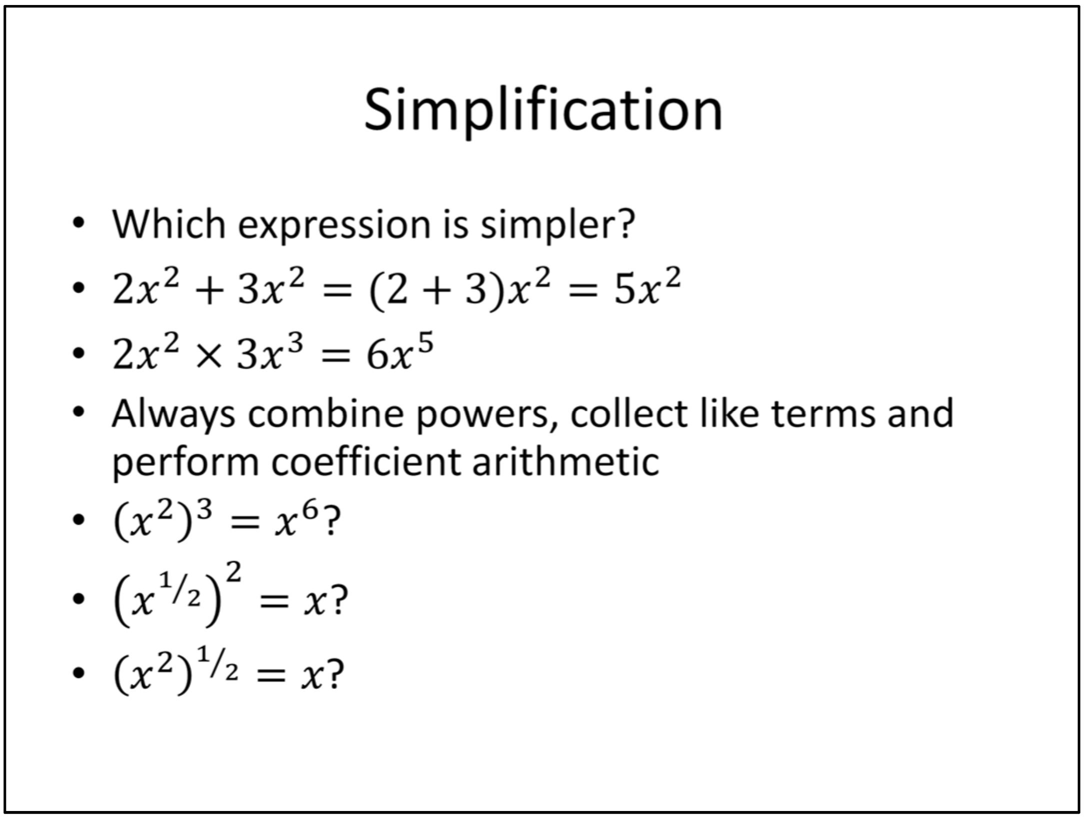
Algebraic computing: Data representations

Each object requires a special data structure, such as a table, array or list. Hence algebraic data representations are more complicated and require more memory than hardware floating point representations.

An arbitrarily large integer is split into a sequence of fixed-size integers (using a fixed base such as a large power of 10).

A rational numbers is represented as a pair of integers.

A monomial is represented as the triple [coefficient, variable, power].
A polynomial is just a collection of monomials.



Simplification

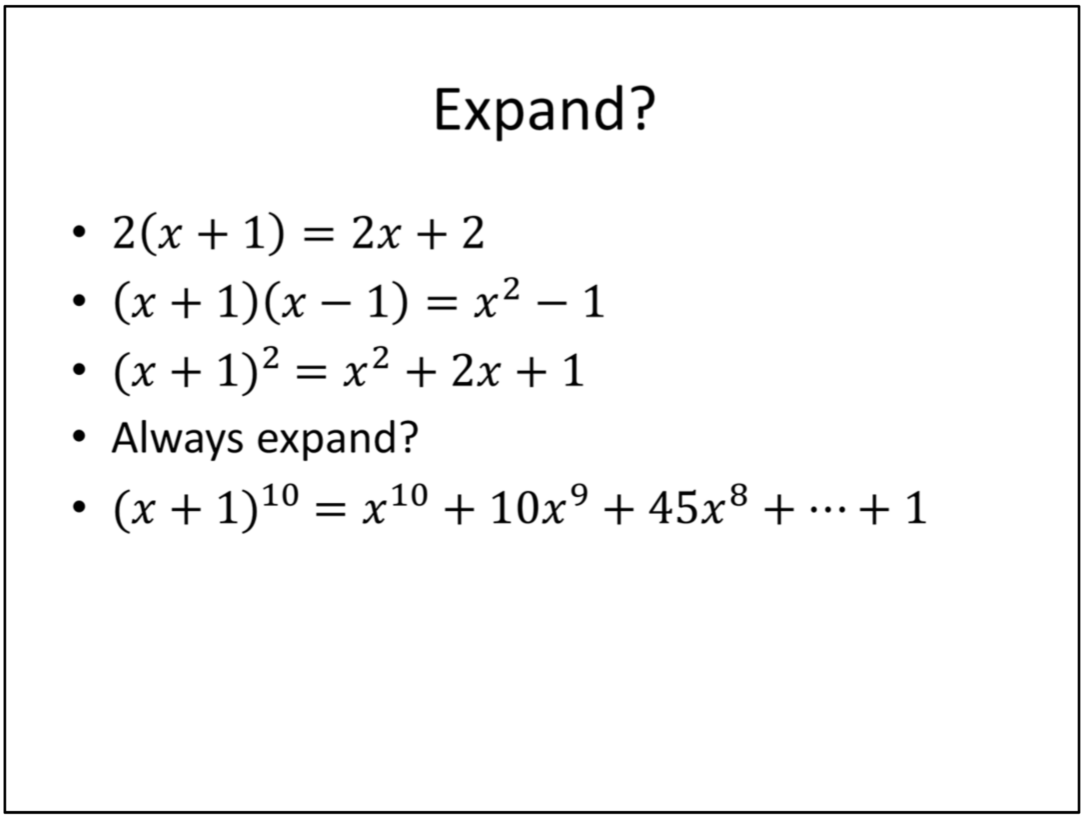
- Which expression is simpler?
- $2x^2 + 3x^2 = (2+3)x^2 = 5x^2$
- $2x^2 \times 3x^3 = 6x^5$
- Always combine powers, collect like terms and perform coefficient arithmetic
- $(x^2)^3 = x^6$?
- $\left(x^{1/2}\right)^2 = x$?
- $(x^2)^{1/2} = x$?

A-Level Maths questions often say “express the result in its simplest form”. But what does this mean? Is “simplest form” a well defined concept?

I’m sure you will agree that the far right hand sides of the first two equations is simplest. But what about the last three?

Fractional powers “on the outside” are tricky.

Maple automatically performs all but the last of these simplifications, which is true only if you take the right branch of the square root. Maple will perform it only as a **symbolic** or **formal** simplification.



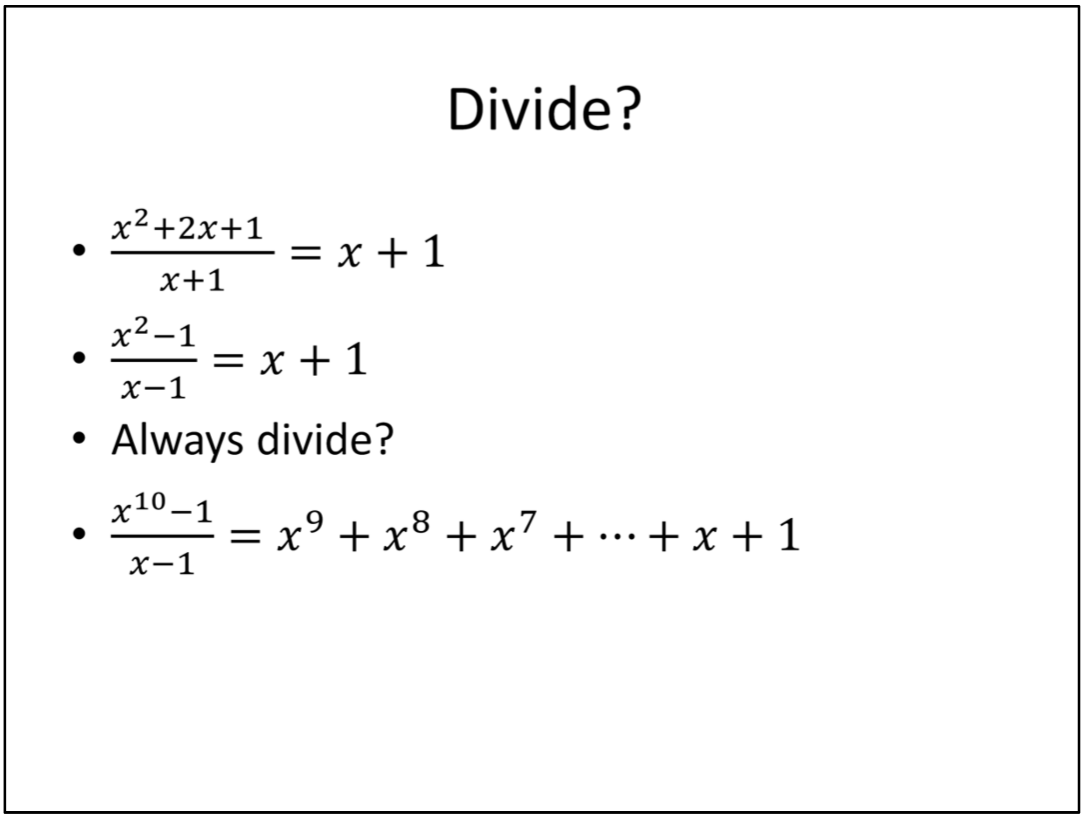
Expand?

- $2(x+1) = 2x+2$
- $(x+1)(x-1) = x^2 - 1$
- $(x+1)^2 = x^2 + 2x + 1$
- Always expand?
- $(x+1)^{10} = x^{10} + 10x^9 + 45x^8 + \cdots + 1$

Which side of these equations is simpler?

Is it best always to expand products?

Not necessarily. Some “simplifications” can produce much larger expressions. But perhaps having no brackets is simpler.



Divide?

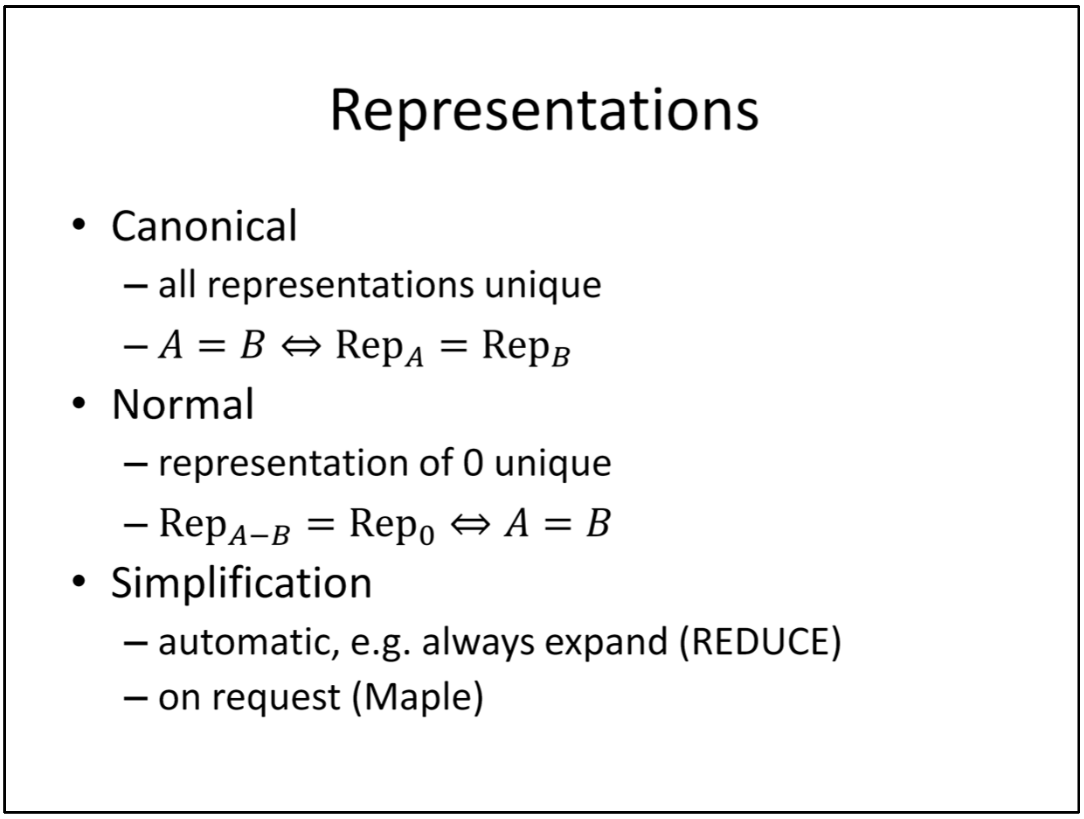
- $\frac{x^2+2x+1}{x+1} = x + 1$
- $\frac{x^2-1}{x-1} = x + 1$
- Always divide?
- $\frac{x^{10}-1}{x-1} = x^9 + x^8 + x^7 + \cdots + x + 1$

Which side of these equations is simpler?

Is it best always to divide if possible?

Not necessarily. Again, some “simplifications” can produce much larger expressions. But perhaps having no fractions is simpler.

“Simple” depends on the context.



Representations

- Canonical
  - all representations unique
  - $A = B \Leftrightarrow \mathrm{Rep}_A = \mathrm{Rep}_B$
- Normal
  - representation of 0 unique
  - $\mathrm{Rep}_{A-B} = \mathrm{Rep}_0 \Leftrightarrow A = B$
- Simplification
  - automatic, e.g. always expand (REDUCE)
  - on request (Maple)

A computer works with **representations** of mathematical objects, not with the abstract objects themselves.

There are two main classes of algebraic representation: **canonical** and **normal**. A canonical representation makes subsequent calculations easier.

Canonical $\Rightarrow$ Normal, Normal $\nRightarrow$ Canonical

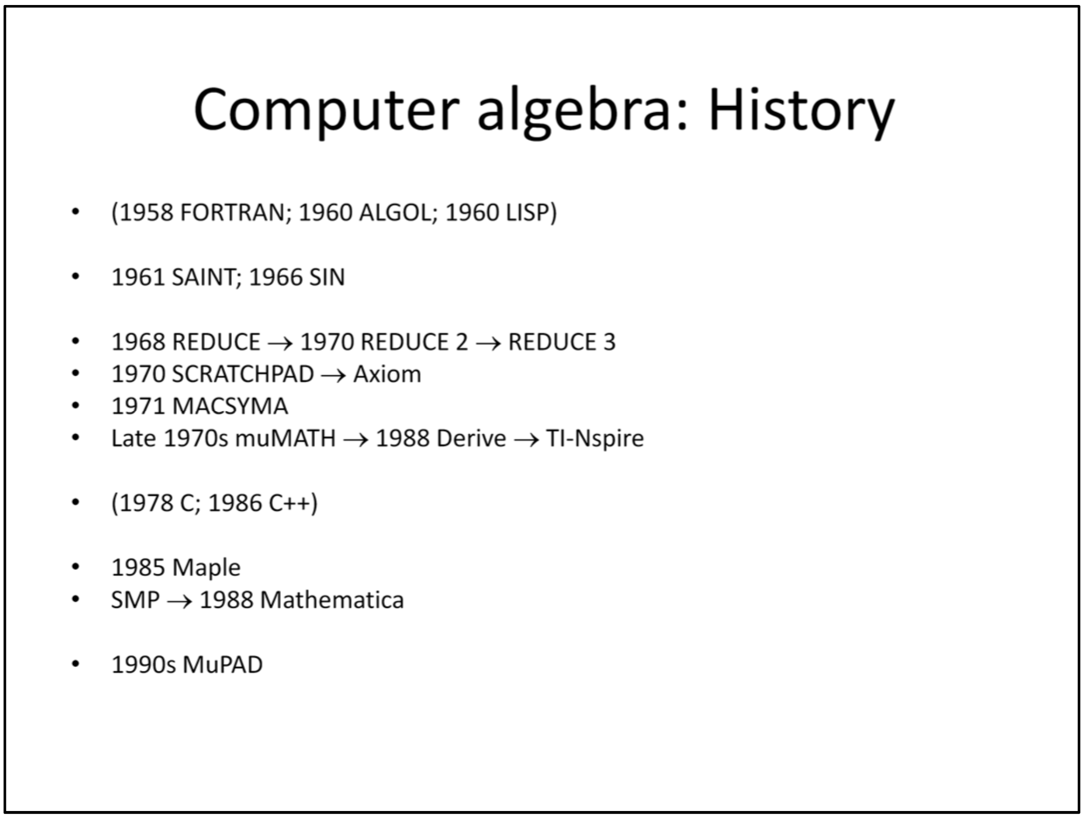
Simplification involves choosing or changing the representation, which should be at least normal.

All CASs perform some simplification automatically, such as collection of like terms and numerical arithmetic.

REDUCE uses a canonical form that is fixed at any given time but can be changed by switches.

Maple performs non-trivial simplifications only on request and the user specifies what kind of simplification to perform, as you will see later.

Simplification over a complex number domain is complicated by issues of which complex branch to use.



Computer algebra: History

- (1958 FORTRAN; 1960 ALGOL; 1960 LISP)
- 1961 SAINT; 1966 SIN
- 1968 REDUCE → 1970 REDUCE 2 → REDUCE 3
- 1970 SCRATCHPAD → Axiom
- 1971 MACSYMA
- Late 1970s muMATH → 1988 Derive → TI-Nspire
- (1978 C; 1986 C++)
- 1985 Maple
- SMP → 1988 Mathematica
- 1990s MuPAD

The first high-level programming languages appeared around 1960. LISP (using a LISt Processing paradigm) was the favourite for **symbolic** computing and was used to write **SAINT** (Symbolic Automatic INTegration) for heuristic integration and **SIN** (Symbolic INtegrator) for algorithmic integration. (Computer programmers have a strange sense of humour!)

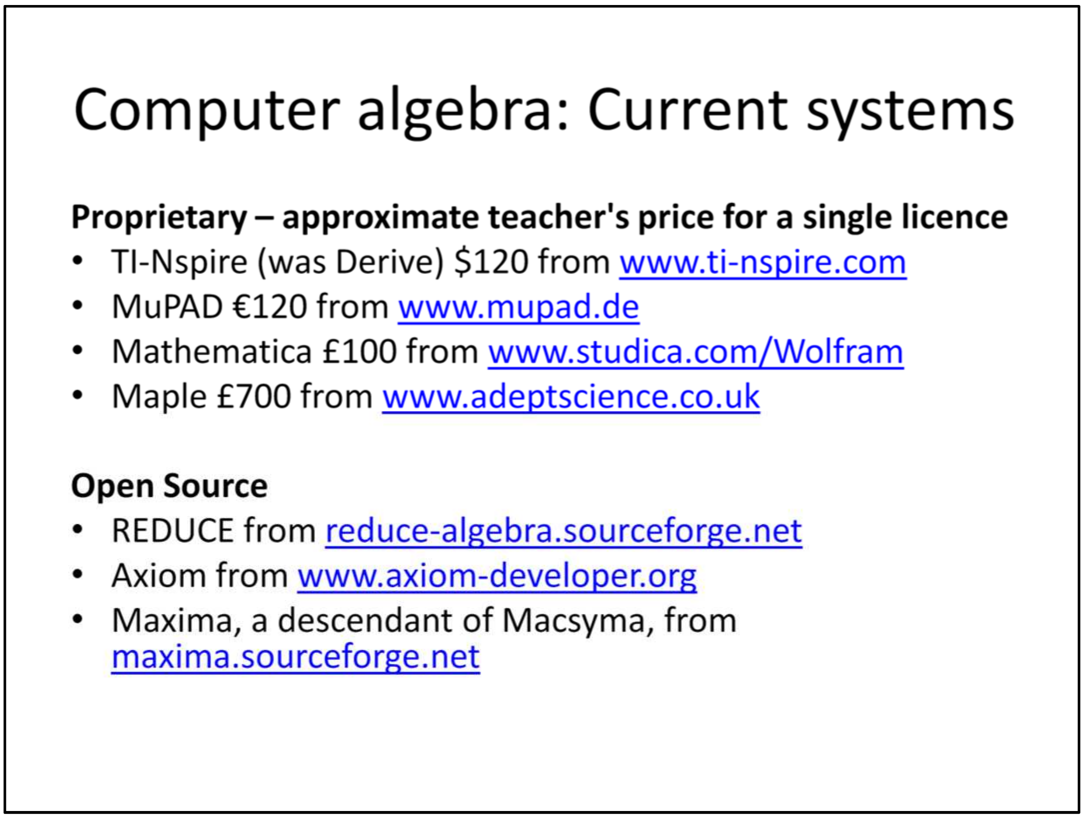
The first general-purpose computer algebra systems were also written in LISP around 1970: **REDUCE**, initially for high-energy physics; **SCRATCHPAD**, an IBM research project for general algebra; and **MACSYMA** for general algebra as an application of Artificial Intelligence. All were relatively large and used only for academic research.

In the late 1970s **muMATH** was written in LISP for the IBM PC and was the only system that would run on a PC. Also in the late 1970s the programming language C was developed, followed by C++. All later new computer algebra systems were implemented in C and/or C++ and marketed commercially.

Maple was developed with teaching in mind; **SMP** (Symbolic Manipulation Program) evolved into **Mathematica** in 1988 and became popular for its Graphical User Interface and graphics. Also in 1988, **DERIVE** evolved from muMATH for PCs and Texas Instruments calculators.

NAG took over SCRATCHPAD and marketed it as **AXIOM**; it has impressive technical facilities but was commercially unsuccessful.

The most recent commercial computer algebra system is **MuPAD**. It was originally developed at the University of Paderborn, Germany, but since 1997 it has been developed commercially by SciFace. It is similar to Maple, but more object-oriented.



Computer algebra: Current systems

Proprietary – approximate teacher's price for a single licence

- TI-Nspire (was Derive) $120 from www.ti-nspire.com
- MuPAD €120 from www.mupad.de
- Mathematica £100 from www.studica.com/Wolfram
- Maple £700 from www.adeptscience.co.uk

Open Source

- REDUCE from reduce-algebra.sourceforge.net
- Axiom from www.axiom-developer.org
- Maxima, a descendant of Macsyma, from maxima.sourceforge.net

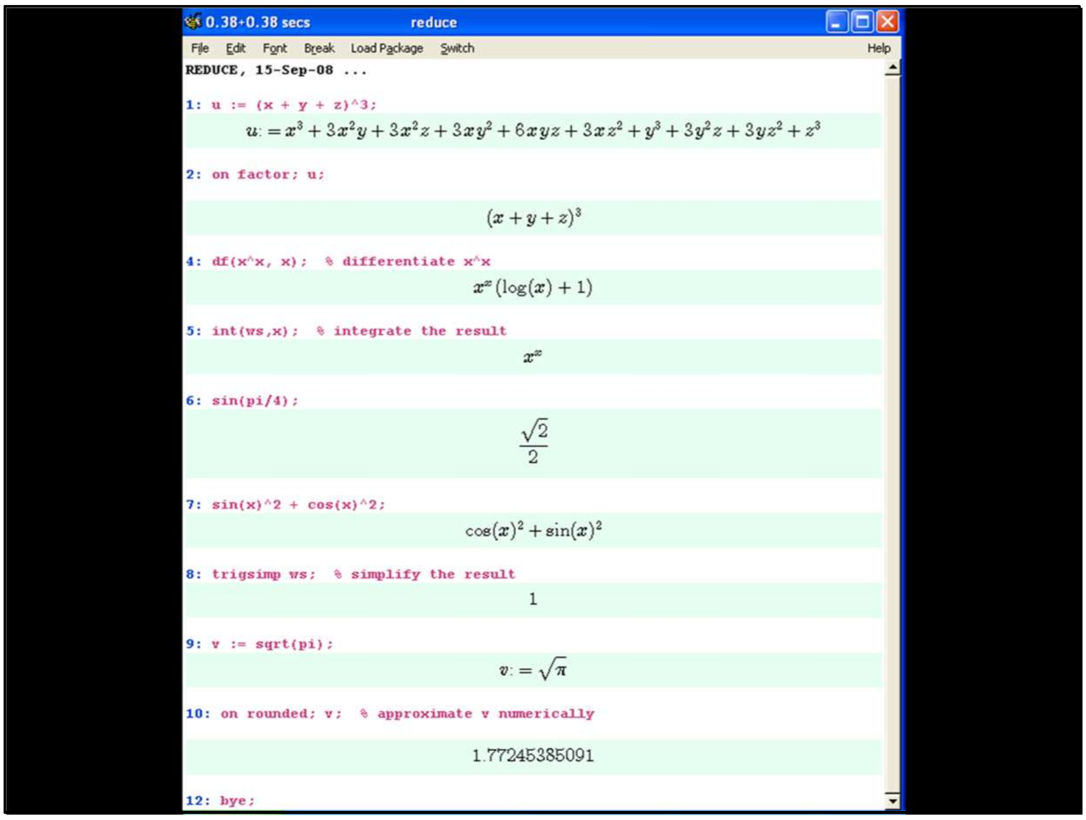
Computer algebra systems are alive and well, but can be expensive!

Derive has evolved into TI-Nspire and is specifically intended for school/college use. I have no experience of it, but Derive was a nice system!

MuPad, Mathematica and Maple are very powerful and primarily intended for university use, although as I hope you will see soon they can also be used at lower levels.

The older mainstream LISP-based computer algebra systems are all available as Open Source Software. REDUCE went Open Source at the beginning of 2009. It is the smallest and probably the easiest for the casual user. It is very easy to run under Windows: just download the latest Windows zip file from SourceForge, unzip it into a convenient folder, and open the application file (reduce.exe).

I predict that REDUCE will be the first Open Source general-purpose computer algebra system to become available on a mobile phone or PDA!



```
REDUCE, 15-Sep-08 ...

1: u := (x + y + z)^3;
```

$$u := x^3 + 3x^2y + 3x^2z + 3xy^2 + 6xyz + 3xz^2 + y^3 + 3y^2z + 3yz^2 + z^3$$

```
2: on factor; u;
```

$$(x + y + z)^3$$

```
4: df(x^x, x);  % differentiate x^x
```

$$x^x\,(\log(x) + 1)$$

```
5: int(ws,x);  % integrate the result
```

$$x^x$$

```
6: sin(pi/4);
```

$$\frac{\sqrt{2}}{2}$$

```
7: sin(x)^2 + cos(x)^2;
```

$$\cos(x)^2 + \sin(x)^2$$

```
8: trigsimp ws;  % simplify the result
```

$$1$$

```
9: v := sqrt(pi);
```

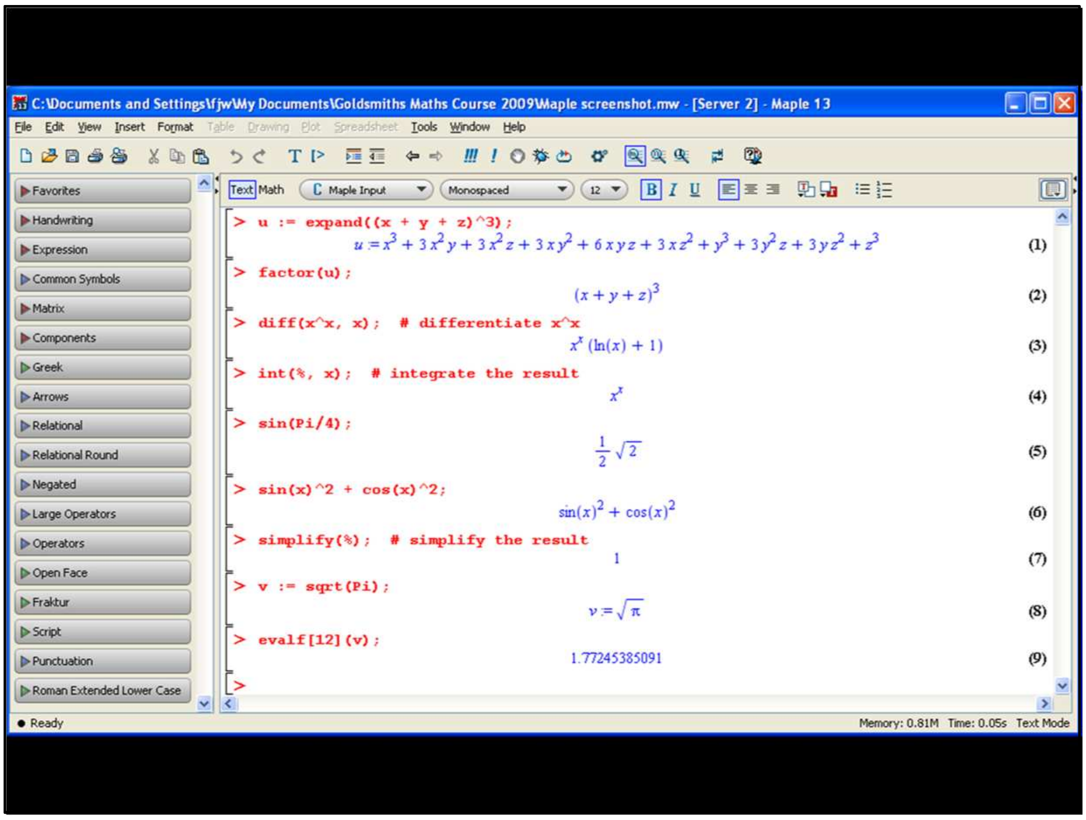
$$v := \sqrt{\pi}$$

```
10: on rounded; v;  % approximate v numerically
```

$$1.77245385091$$

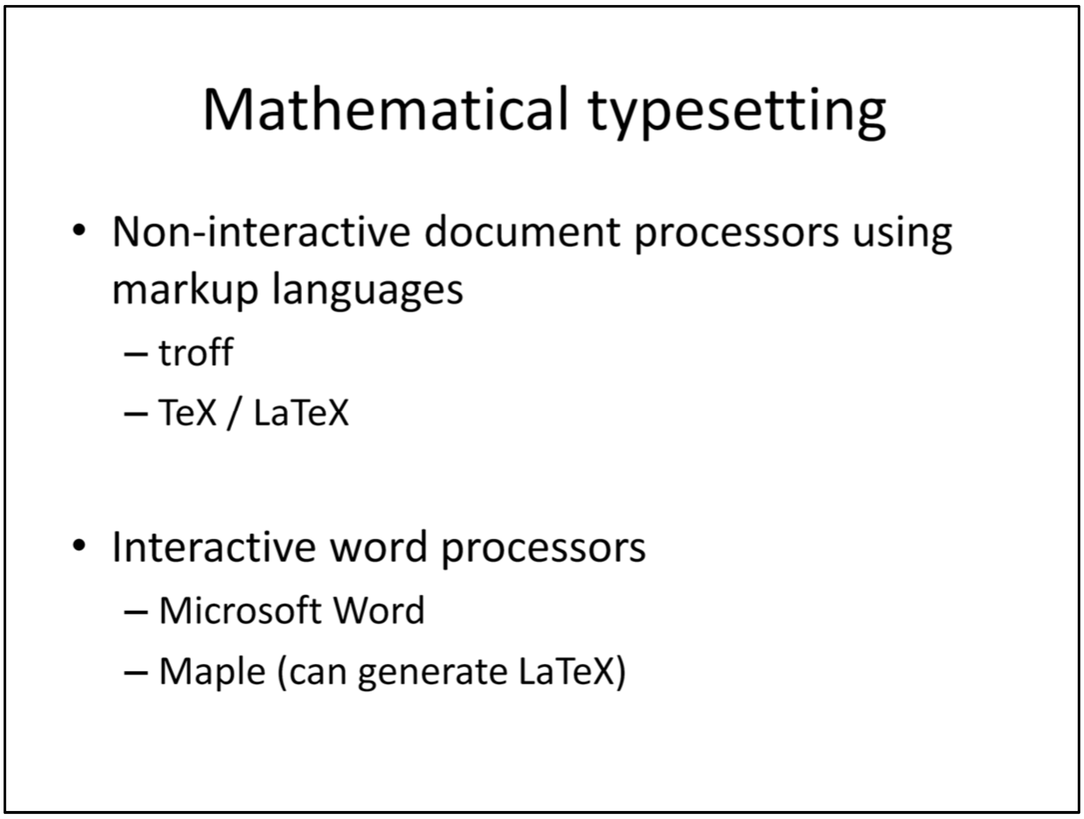
```
12: bye;
```

This screen-shot shows the standard Windows interface for CSL-REDUCE. Input is typed using the standard linear syntax that is common to most computer programming languages. On pressing the *Enter/Return* key, the input is executed and the result displayed.


C:\Documents and Settings\fjw\My Documents\Goldsmiths Maths Course 2009\Maple screenshot.mw - [Server 2] - Maple 13
File Edit View Insert Format Table Drawing Plot Spreadsheet Tools Window Help
Favorites
Handwriting
Expression
Common Symbols
Matrix
Components
Greek
Arrows
Relational
Relational Round
Negated
Large Operators
Operators
Open Face
Fraktur
Script
Punctuation
Roman Extended Lower Case
Text Math
Maple Input
Monospaced
12
Ready
Memory: 0.81M Time: 0.05s Text Mode

```
> u := expand((x + y + z)^3);
```
$$u := x^3 + 3x^2y + 3x^2z + 3xy^2 + 6xyz + 3xz^2 + y^3 + 3y^2z + 3yz^2 + z^3 \quad (1)$$
```
> factor(u);
```
$$(x + y + z)^3 \quad (2)$$
```
> diff(x^x, x);  # differentiate x^x
```
$$x^x (\ln(x) + 1) \quad (3)$$
```
> int(%, x);  # integrate the result
```
$$x^x \quad (4)$$
```
> sin(Pi/4);
```
$$\frac{1}{2}\sqrt{2} \quad (5)$$
```
> sin(x)^2 + cos(x)^2;
```
$$\sin(x)^2 + \cos(x)^2 \quad (6)$$
```
> simplify(%);  # simplify the result
```
$$1 \quad (7)$$
```
> v := sqrt(Pi);
```
$$v := \sqrt{\pi} \quad (8)$$
```
> evalf[12](v);
```
$$1.77245385091 \quad (9)$$
```
>
```

This screen-shot illustrates Maple using the same linear input syntax and performing the same computations as shown in the previous REDUCE screen-shot. Maple offers a more sophisticated Graphical User Interface (but costs a lot more; REDUCE is free). We will use Maple’s mathematical input mode later in this course.



Mathematical typesetting

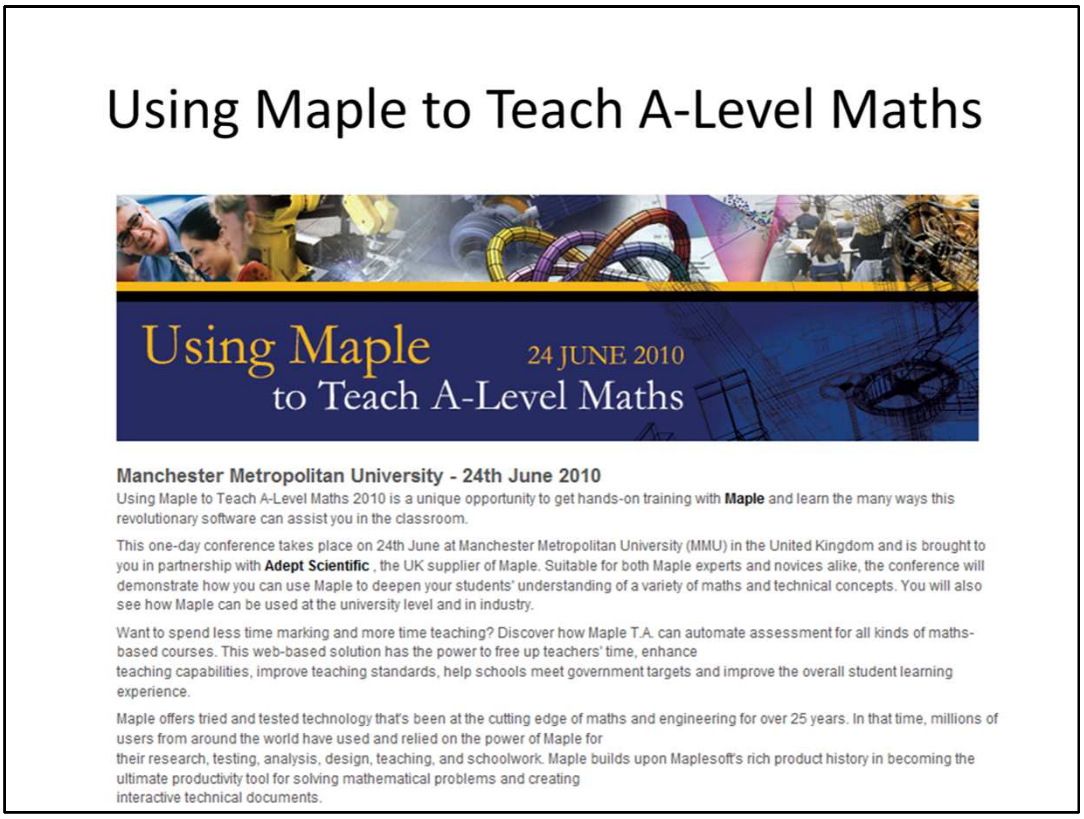
- Non-interactive document processors using markup languages
  - troff
  - TeX / LaTeX
- Interactive word processors
  - Microsoft Word
  - Maple (can generate LaTeX)

Discuss only if time (which is unlikely)
Mathematical typesetting is still often done by hand very badly!

Troff was originally written at Bell Labs for Unix by Joe Ossanna in about 1973, to drive the Graphic Systems CAT typesetter. In 1979 Brian Kernighan modified it to produce output for a variety of typesetters. Macro packages and pre-processors, such as tbl and eqn, provide more user-friendly input languages for tables and mathematics. Open Source versions of troff etc. are available as GNU troff (groff). Troff is primarily used for Unix manuals.

TeX was first developed in 1977 by Donald Knuth at Stanford and in the early 1980s Leslie Lamport began developing a more user-friendly version of TeX called LaTeX. Support for tables, mathematics and device-independent output are core features. TeX and LaTeX (and many related packages) are freely available Open Source software that run on all modern computers; MiKTeX is a good implementation for Microsoft Windows. Lyx is a useful freely available Open Source WYSIWYG editor for LaTeX. TeX and especially LaTeX are widely used by university mathematicians.

Using Maple to Teach A-Level Maths



Manchester Metropolitan University - 24th June 2010

Using Maple to Teach A-Level Maths 2010 is a unique opportunity to get hands-on training with **Maple** and learn the many ways this revolutionary software can assist you in the classroom.

This one-day conference takes place on 24th June at Manchester Metropolitan University (MMU) in the United Kingdom and is brought to you in partnership with **Adept Scientific** , the UK supplier of Maple. Suitable for both Maple experts and novices alike, the conference will demonstrate how you can use Maple to deepen your students' understanding of a variety of maths and technical concepts. You will also see how Maple can be used at the university level and in industry.

Want to spend less time marking and more time teaching? Discover how Maple T.A. can automate assessment for all kinds of maths-based courses. This web-based solution has the power to free up teachers' time, enhance teaching capabilities, improve teaching standards, help schools meet government targets and improve the overall student learning experience.

Maple offers tried and tested technology that's been at the cutting edge of maths and engineering for over 25 years. In that time, millions of users from around the world have used and relied on the power of Maple for their research, testing, analysis, design, teaching, and schoolwork. Maple builds upon Maplesoft's rich product history in becoming the ultimate productivity tool for solving mathematical problems and creating interactive technical documents.

Computers and the web are being used increasingly in schools.

Look out for future conferences such as this one.

How does any mathematical software you may be using compare with the most advanced systems?

Should you be considering the use of a computer algebra system such as Maple?

Find out this afternoon when you can try solving A-Level Maths problems yourself using Maple.

[http://www.docm.mmu.ac.uk/maple_maths_conference/]