Computational Mathematics
Lecture 1

Patrick E. Farrell

University of Oxford

Section 1

Computational mathematics by example

In 1781, William Herschel discovered Uranus with a telescope he had built in his back garden in Bath.

William Herschel, 1738–1822

In 1781, William Herschel discovered Uranus with a telescope he had built in his back garden in Bath.

Astronomers had a theory for predicting the spacing between the planets, the Titius–Bode law:

$$d(n) = 0.4 + 0.3 \times 2^n, \quad n = -\infty, 0, 1, \ldots.$$

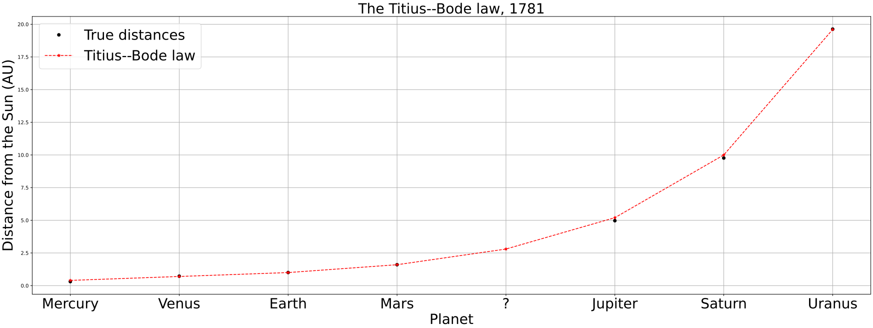
William Herschel, 1738–1822

In 1781, William Herschel discovered Uranus with a telescope he had built in his back garden in Bath.

Astronomers had a theory for predicting the spacing between the planets, the Titius–Bode law:

$$d(n) = 0.4 + 0.3 \times 2^n, \quad n = -\infty, 0, 1, \ldots.$$

William Herschel, 1738–1822



So where was the missing planet for $n = 3$? Astronomers around Europe furiously searched the skies between Mars and Jupiter.

So where was the missing planet for $n = 3$? Astronomers around Europe furiously searched the skies between Mars and Jupiter.

On 1 January 1801, Giuseppe Piazzi discovered Ceres, almost exactly where the Titius–Bode law predicted!

Giuseppe Piazzi, 1746–1826

So where was the missing planet for $n = 3$? Astronomers around Europe furiously searched the skies between Mars and Jupiter.

On 1 January 1801, Giuseppe Piazzi discovered Ceres, almost exactly where the Titius–Bode law predicted!

But he could only observe it for 41 days before it was lost behind the Sun—not long enough to compute its orbit. How could it be found again?

Giuseppe Piazzi, 1746–1826

In Brunswick, the 24-year old Gauss had invented the *method of least squares*, for estimating coefficients of mathematical expressions from partial and noisy data.

Carl Friedrich Gauss, 1777–1855

In Brunswick, the 24-year old Gauss had invented the *method of least squares*, for estimating coefficients of mathematical expressions from partial and noisy data.

This method works by finding the coefficients that minimise the distance between the expression and the data.

Carl Friedrich Gauss, 1777–1855

In Brunswick, the 24-year old Gauss had invented the *method of least squares*, for estimating coefficients of mathematical expressions from partial and noisy data.

This method works by finding the coefficients that minimise the distance between the expression and the data.

Carl Friedrich Gauss, 1777–1855

Gauss knew that orbits are approximately ellipses with the sun at one focus, which left him with six parameters to estimate from Piazzi's 22 observations.

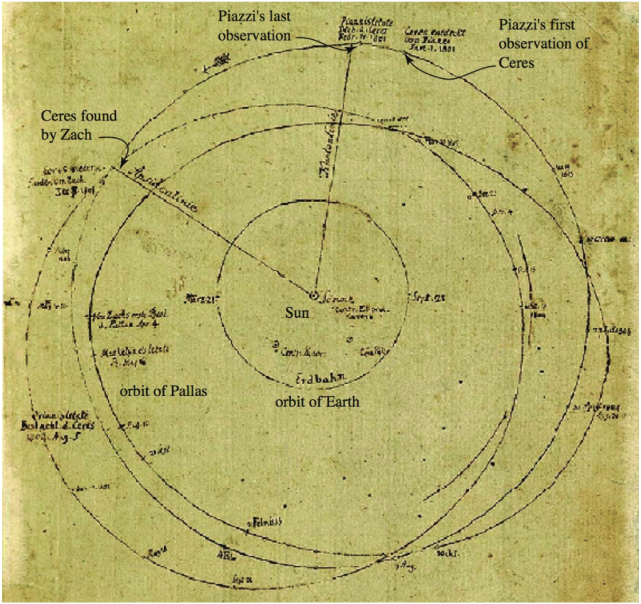
In Brunswick, the 24-year old Gauss had invented the *method of least squares*, for estimating coefficients of mathematical expressions from partial and noisy data.

This method works by finding the coefficients that minimise the distance between the expression and the data.

Carl Friedrich Gauss, 1777–1855

Gauss knew that orbits are approximately ellipses with the sun at one focus, which left him with six parameters to estimate from Piazzi's 22 observations.

Gauss calculated for weeks on end; he published his prediction for Ceres' location in September 1801. On December 7, astronomers found Ceres again, almost exactly where he predicted.



Annotated sketch from Gauss' papers. Courtesy Georg-August-Universität Göttingen.

It is possible to find two squares that sum to a square:

$$3^2 + 4^2 = 5^2,$$

and three cubes that sum to a cube:

$$3^3 + 4^3 + 5^3 = 6^3.$$

Leonhard Euler, 1707–1783

It is possible to find two squares that sum to a square:

$$3^2 + 4^2 = 5^2,$$

and three cubes that sum to a cube:

$$3^3 + 4^3 + 5^3 = 6^3.$$

Leonhard Euler, 1707–1783

But Euler could not find natural solutions to

$$a_1^3 + a_2^3 = b^3 \quad \text{or} \quad a_1^4 + a_2^4 + a_3^4 = b^4,$$

the first statement being Fermat's Last Theorem.

It is possible to find two squares that sum to a square:

$$3^2 + 4^2 = 5^2,$$

and three cubes that sum to a cube:

$$3^3 + 4^3 + 5^3 = 6^3.$$

Leonhard Euler, 1707–1783

But Euler could not find natural solutions to

$$a_1^3 + a_2^3 = b^3 \quad \text{or} \quad a_1^4 + a_2^4 + a_3^4 = b^4,$$

the first statement being Fermat's Last Theorem.

So, in 1769, Euler conjectured that

$$\exists\, k > 1, n > 1, a_1, \ldots, a_n, b \in \mathbb{N}_+ : a_1^k + a_2^k + \cdots a_n^k = b^k \implies k \leq n.$$

Euler's Conjecture remained open for nearly 200 years.

Euler's Conjecture remained open for nearly 200 years.

In 1966, Leon J. Lander and Thomas R. Parkin discovered a counterexample:

COUNTEREXAMPLE TO EULER'S CONJECTURE ON SUMS OF LIKE POWERS

BY L. J. LANDER AND T. R. PARKIN

Communicated by J. D. Swift, June 27, 1966

A direct search on the CDC 6600 yielded

$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

as the smallest instance in which four fifth powers sum to a fifth power. This is a counterexample to a conjecture by Euler [1] that at least n nth powers are required to sum to an nth power, $n > 2$.

REFERENCE

1. L. E. Dickson, *History of the theory of numbers*, Vol. 2, Chelsea, New York, 1952, p. 648.

Computers are not just useful for finding counterexamples, though. They can help us find *proofs*.

Computers are not just useful for finding counterexamples, though. They can help us find *proofs*.

In 1852, Francis Guthrie was colouring a map of the counties of England. He noticed he could satisfy the constraint that counties sharing a border were coloured differently with only four colours.

Francis Guthrie, 1831–1899

Computers are not just useful for finding counterexamples, though. They can help us find *proofs*.

In 1852, Francis Guthrie was colouring a map of the counties of England. He noticed he could satisfy the constraint that counties sharing a border were coloured differently with only four colours.

Was this true for all (reasonable) maps?

Francis Guthrie, 1831–1899

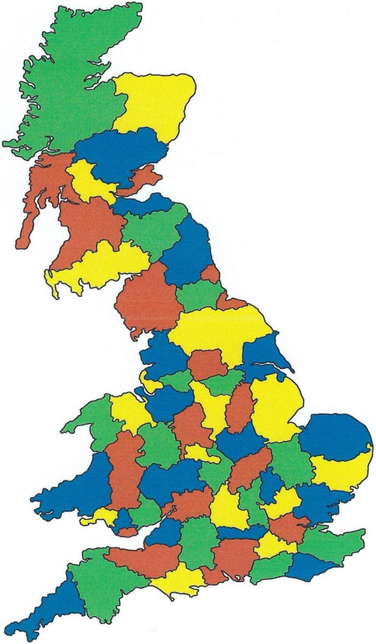
Computers are not just useful for finding counterexamples, though. They can help us find *proofs*.

In 1852, Francis Guthrie was colouring a map of the counties of England. He noticed he could satisfy the constraint that counties sharing a border were coloured differently with only four colours.

Francis Guthrie, 1831–1899

Was this true for all (reasonable) maps?

This is equivalent to colouring a *graph*: each region is a vertex, and adjacent regions are connected with an edge.



In 1879, Alfred Kempe published a clever proof. He introduced an 'unavoidable set', a set of six configurations that *any* graph *must* have at least one of.

Alfred Kempe, 1849–1922

In 1879, Alfred Kempe published a clever proof. He introduced an ‘unavoidable set’, a set of six configurations that *any* graph *must* have at least one of.

He then proved that for each element of the unavoidable set, a graph containing that fragment could be coloured with four colours. Done!

Alfred Kempe, 1849–1922

In 1879, Alfred Kempe published a clever proof. He introduced an ‘unavoidable set’, a set of six configurations that *any* graph *must* have at least one of.

He then proved that for each element of the unavoidable set, a graph containing that fragment could be coloured with four colours. Done!

However, in 1890, Percy Heawood showed Kempe’s proof was wrong for the last element of his unavoidable set.

Alfred Kempe, 1849–1922

Percy Heawood, 1861–1955

In 1879, Alfred Kempe published a clever proof. He introduced an ‘unavoidable set’, a set of six configurations that *any* graph *must* have at least one of.

He then proved that for each element of the unavoidable set, a graph containing that fragment could be coloured with four colours. Done!

However, in 1890, Percy Heawood showed Kempe’s proof was wrong for the last element of his unavoidable set.

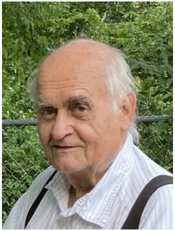
While the proof was wrong, the basic strategy was right.

Alfred Kempe, 1849–1922

Percy Heawood, 1861–1955

In 1976, Appel & Haken announced the first correct proof of the four colour theorem.

Kenneth Appel, 1932–2013



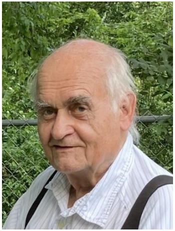
Wolfgang Haken, 1928–2022

In 1976, Appel & Haken announced the first correct proof of the four colour theorem.

With a computer, they found an unavoidable set with 1834 cases, and programmed it to mechanically check that in each case the graph can be coloured with four colours.

The proof took over 1000 hours of computer time.

Kenneth Appel, 1932–2013



Wolfgang Haken, 1928–2022

In 1976, Appel & Haken announced the first correct proof of the four colour theorem.

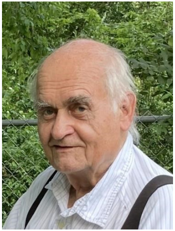
With a computer, they found an unavoidable set with 1834 cases, and programmed it to mechanically check that in each case the graph can be coloured with four colours.

The proof took over 1000 hours of computer time.

Since then, many theorems have been proven with computer-assisted proofs, among them

- Kepler's conjecture on packing cannonballs;
- Keller's conjecture on tiling Euclidean space;
- Feigenbaum's conjecture in dynamical systems.

Kenneth Appel, 1932–2013



Wolfgang Haken, 1928–2022

Dorothy Hodgkin was one of Oxford's pioneers of computational mathematics.

Dorothy Hodgkin, 1910–1994

Dorothy Hodgkin was one of Oxford's pioneers of computational mathematics.

In 1945, she identified the molecular structure of penicillin. She did this by passing X-rays through the atoms; the crystalline structure diffracts the beam in different directions.

Dorothy Hodgkin, 1910–1994

Dorothy Hodgkin was one of Oxford's pioneers of computational mathematics.

In 1945, she identified the molecular structure of penicillin. She did this by passing X-rays through the atoms; the crystalline structure diffracts the beam in different directions.

This allowed Hodgkin to compute the three-dimensional electron density function of the molecule from the two-dimensional diffraction patterns.

Dorothy Hodgkin, 1910–1994

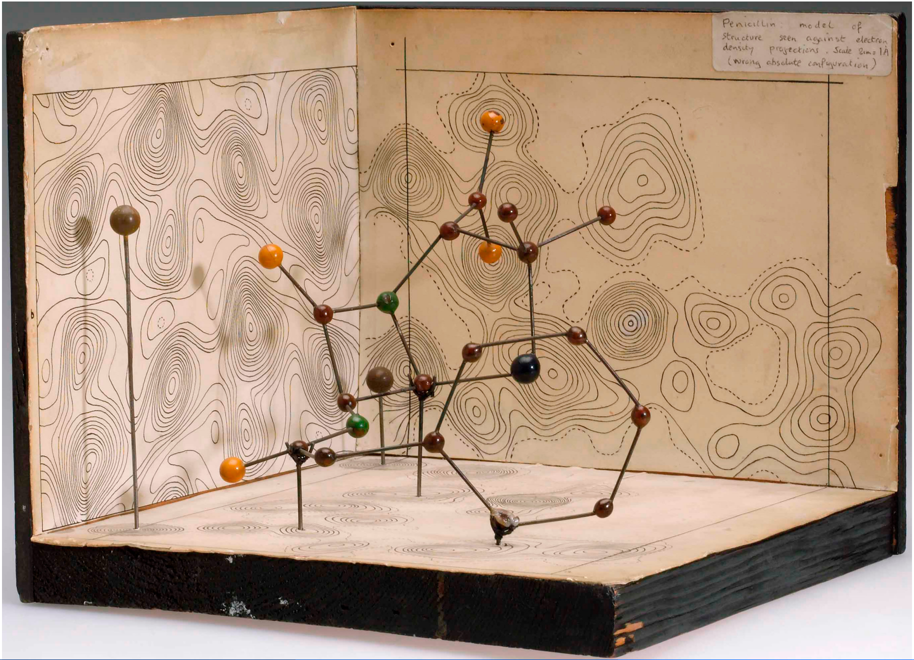
Dorothy Hodgkin was one of Oxford's pioneers of computational mathematics.

In 1945, she identified the molecular structure of penicillin. She did this by passing X-rays through the atoms; the crystalline structure diffracts the beam in different directions.

This allowed Hodgkin to compute the three-dimensional electron density function of the molecule from the two-dimensional diffraction patterns.

The calculations involved least squares, Fourier analysis, and extensive use of group theory.

Dorothy Hodgkin, 1910–1994


Penicillin: model of
structure seen against electron
density projections. Scale 2cm = 1Å
(wrong absolute configuration)

Hodgkin discovered penicillin by hand calculation; to tackle larger molecules, computers were required.

Hodgkin discovered penicillin by hand calculation; to tackle larger molecules, computers were required.

She chaired the committee overseeing Oxford's first computer purchase in 1952, and her group was involved in founding the Oxford University Computing Laboratory—now the Numerical Analysis Group.

Hodgkin discovered penicillin by hand calculation; to tackle larger molecules, computers were required.

She chaired the committee overseeing Oxford's first computer purchase in 1952, and her group was involved in founding the Oxford University Computing Laboratory—now the Numerical Analysis Group.

In 1964 she won the Nobel Prize in Chemistry for her identification of penicillin and vitamin B_{12}.

So what is this subject all about?

So what is this subject all about?

Computational mathematics

Computational mathematics is the subject that studies the use of computation to solve mathematical problems.

So what is this subject all about?

Computational mathematics

Computational mathematics is the subject that studies the use of computation to solve mathematical problems.

These problems might be in pure mathematics (Euler's Conjecture in number theory, the Four-Colour Theorem in graph theory), or in applied mathematics (Gauss' discovery of the orbit of Ceres, Hodgkin's work in crystallography).

So what is this subject all about?

Computational mathematics

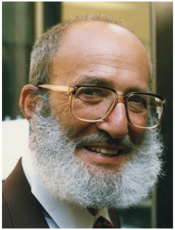
Computational mathematics is the subject that studies the use of computation to solve mathematical problems.

These problems might be in pure mathematics (Euler's Conjecture in number theory, the Four-Colour Theorem in graph theory), or in applied mathematics (Gauss' discovery of the orbit of Ceres, Hodgkin's work in crystallography).

Computational mathematics is an ancient subject; it did not begin with the invention of computers. Instead, *computers were invented to speed up computational mathematics!*

In 1985, Paul Halmos wrote

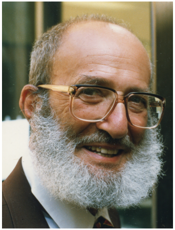
> *When you try to prove a theorem, you don't just list the hypotheses, and then start to reason. What you do is trial and error, experimentation, guesswork. You want to find out what the facts are.*



Paul Halmos, 1916–2006

In 1985, Paul Halmos wrote

> *When you try to prove a theorem, you don't just list the hypotheses, and then start to reason. What you do is trial and error, experimentation, guesswork. You want to find out what the facts are.*

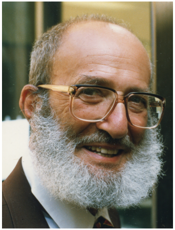


Paul Halmos, 1916–2006

Normally we present mathematics backwards from how it is done. We state a clean, general, abstract theorem, and give examples. But almost always the theorem was first conjectured based on experiments and calculations.

In 1985, Paul Halmos wrote

> *When you try to prove a theorem, you don't just list the hypotheses, and then start to reason. What you do is trial and error, experimentation, guesswork. You want to find out what the facts are.*



Paul Halmos, 1916–2006

Normally we present mathematics backwards from how it is done. We state a clean, general, abstract theorem, and give examples. But almost always the theorem was first conjectured based on experiments and calculations.

As G. H. Hardy wrote,

> *The theory of numbers, more than any other branch of mathematics, began by being an experimental science. Its most famous theorems have all been conjectured, sometimes a hundred years or more before they were proved; and they have been suggested by the evidence of a mass of computations.*

Godfrey Hardy, 1877–1947

Section 2

Practicalities

In this course, we will study the practical side of computational mathematics. Our objectives are

- to solve mathematical problems with computers;
- along the way to learn to program computers.

In this course, we will study the practical side of computational mathematics. Our objectives are

- to solve mathematical problems with computers;
- along the way to learn to program computers.

After this course, I hope to convince you that this subject can greatly aid your study and practice of mathematics. It is also great fun!

In this course, we will study the practical side of computational mathematics. Our objectives are

- to solve mathematical problems with computers;
- along the way to learn to program computers.

After this course, I hope to convince you that this subject can greatly aid your study and practice of mathematics. It is also great fun!

On a pragmatic point, a very large fraction of Oxford mathematics graduates will pursue careers where programming is useful, if not essential. These include

- mathematical research;
- scientific research;
- quantitative finance;
- teaching;
- data science;
- management consulting.

In this course, we will learn the Python programming language.

In this course, we will learn the Python programming language.

Python is one of the world's most popular programming languages. In particular, it is the leading language in scientific data analysis and machine learning.

In this course, we will learn the Python programming language.

Python is one of the world's most popular programming languages. In particular, it is the leading language in scientific data analysis and machine learning.

Python

- has simple and elegant syntax;
- is quick and easy to learn;
- is free software.

In this course, we will learn the Python programming language.

Python is one of the world's most popular programming languages. In particular, it is the leading language in scientific data analysis and machine learning.

Python

- has simple and elegant syntax;
- is quick and easy to learn;
- is free software.

Python was invented by Guido van Rossum in 1989.

Guido van Rossum, 1956–

Unlike most of your courses, this course is studied mainly in your own time, using the course handbook on

`https://courses.maths.ox.ac.uk/course/view.php?id=5471`

Unlike most of your courses, this course is studied mainly in your own time, using the course handbook on

`https://courses.maths.ox.ac.uk/course/view.php?id=5471`

Weeks	Chapters to read	Optional chapters	Problem sheet to start
1–2 MT	1–3	-	-
3–4 MT	4–5	-	I.1
5–6 MT	7	8	I.2
7–8 MT	10	-	I.3
1–2 HT	12–13	-	I.4

Unlike most of your courses, this course is studied mainly in your own time, using the course handbook on

`https://courses.maths.ox.ac.uk/course/view.php?id=5471`

Weeks	Chapters to read	Optional chapters	Problem sheet to start
1–2 MT	1–3	-	-
3–4 MT	4–5	-	I.1
5–6 MT	7	8	I.2
7–8 MT	10	-	I.3
1–2 HT	12–13	-	I.4

There are four two-hour demonstration sessions for this course; three this term, and one next term. In demonstration session n you start problem sheet n, and return it for marking in demonstration session $n+1$.

None of the work this term is formally assessed. Work collaboratively with your friends and ask your tutors for advice.

None of the work this term is formally assessed. Work collaboratively with your friends and ask your tutors for advice.

This term's work forms the basis for your projects in Hilary and Trinity terms. Three projects will be announced; you choose two of them. The projects are done in the same manner as the problem sheets for this term, but with more emphasis on interweaving coding, mathematics, and discussion.

None of the work this term is formally assessed. Work collaboratively with your friends and ask your tutors for advice.

This term's work forms the basis for your projects in Hilary and Trinity terms. Three projects will be announced; you choose two of them. The projects are done in the same manner as the problem sheets for this term, but with more emphasis on interweaving coding, mathematics, and discussion.

Your marks for computational mathematics form part of your marks for the Preliminary Examinations. In particular, getting a passing grade is necessary for passing the Preliminary Examinations.

None of the work this term is formally assessed. Work collaboratively with your friends and ask your tutors for advice.

This term's work forms the basis for your projects in Hilary and Trinity terms. Three projects will be announced; you choose two of them. The projects are done in the same manner as the problem sheets for this term, but with more emphasis on interweaving coding, mathematics, and discussion.

Your marks for computational mathematics form part of your marks for the Preliminary Examinations. In particular, getting a passing grade is necessary for passing the Preliminary Examinations.

The deadlines for these projects are

- 1^{st} project: 12 noon on Monday of week 2 TT24
- 2^{nd} project: 12 noon on Monday of week 5 TT24

None of the work this term is formally assessed. Work collaboratively with your friends and ask your tutors for advice.

This term's work forms the basis for your projects in Hilary and Trinity terms. Three projects will be announced; you choose two of them. The projects are done in the same manner as the problem sheets for this term, but with more emphasis on interweaving coding, mathematics, and discussion.

Your marks for computational mathematics form part of your marks for the Preliminary Examinations. In particular, getting a passing grade is necessary for passing the Preliminary Examinations.

The deadlines for these projects are

- 1^{st} project: 12 noon on Monday of week 2 TT24
- 2^{nd} project: 12 noon on Monday of week 5 TT24

These submissions must be your own unaided work. **No AI.**

Your next steps:

- Download the course handbook.

Your next steps:

- Download the course handbook.
- Find out your schedule for demonstration sessions!

Your next steps:

- Download the course handbook.
- Find out your schedule for demonstration sessions!
- Install the required software **before** the demonstration sessions.

Your next steps:

- Download the course handbook.
- Find out your schedule for demonstration sessions!
- Install the required software **before** the demonstration sessions.

(Optionally) bring your laptops along to the next lecture to follow along with installation.

$\rightarrow$ the Lander–Parkin counterexample

Computational Mathematics
Lecture 2

Patrick E. Farrell

University of Oxford

How to submit problem sheets

A brief tour of the course

- Week 3–4 MT
- Week 5–6 MT
- Week 7–8 MT
- Week 1–2 HT

Software installation

Section 1

How to submit problem sheets

$\rightarrow$ using `publish.py`

Section 2

A brief tour of the course

Week 3–4 MT teaches

- arithmetic,
- conditionals,
- iteration.

Week 3–4 MT ends with a code for *bisection*, an algorithm for finding $x^\star$ such that $f(x^\star) = 0$.

It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Week 3–4 MT ends with a code for *bisection*, an algorithm for finding $x^\star$ such that $f(x^\star) = 0$.

It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

If $f : [a, b] \to \mathbb{R}$ is continuous with $f(a)f(b) < 0$, then there exists $x^\star \in (a, b)$ with $f(x^\star) = 0$.

The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

Bernhard Bolzano, 1781–1848

Week 3–4 MT ends with a code for *bisection*, an algorithm for finding $x^\star$ such that $f(x^\star) = 0$.

It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

If $f : [a, b] \to \mathbb{R}$ is continuous with $f(a)f(b) < 0$, then there exists $x^\star \in (a, b)$ with $f(x^\star) = 0$.

The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

Bernhard Bolzano, 1781–1848

We evaluate f at $c = (a + b)/2$. We then have three possibilities:

1. $f(c) = 0$, so we are done!

Week 3–4 MT ends with a code for *bisection*, an algorithm for finding $x^\star$ such that $f(x^\star) = 0$.

It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

If $f : [a, b] \to \mathbb{R}$ *is continuous with* $f(a)f(b) < 0$*, then there exists* $x^\star \in (a, b)$ *with* $f(x^\star) = 0$*.*

The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

Bernhard Bolzano, 1781–1848

We evaluate f at $c = (a + b)/2$. We then have three possibilities:

1. $f(c) = 0$, so we are done!
2. $f(c)$ has the same sign as $f(a)$, so there exists a root in (c, b).

Week 3–4 MT ends with a code for *bisection*, an algorithm for finding $x^\star$ such that $f(x^\star) = 0$.

It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

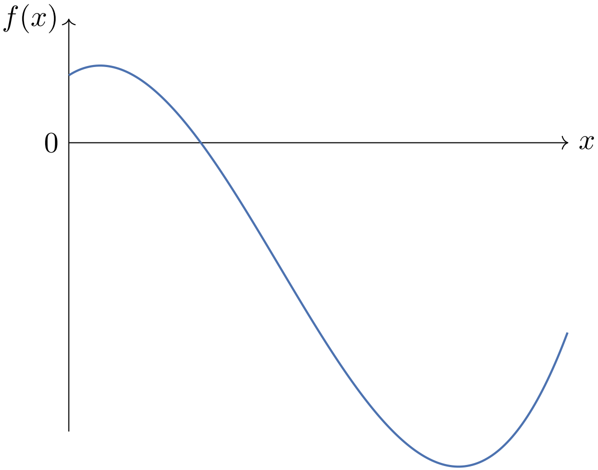
If $f : [a, b] \to \mathbb{R}$ is continuous with $f(a)f(b) < 0$, then there exists $x^\star \in (a, b)$ with $f(x^\star) = 0$.

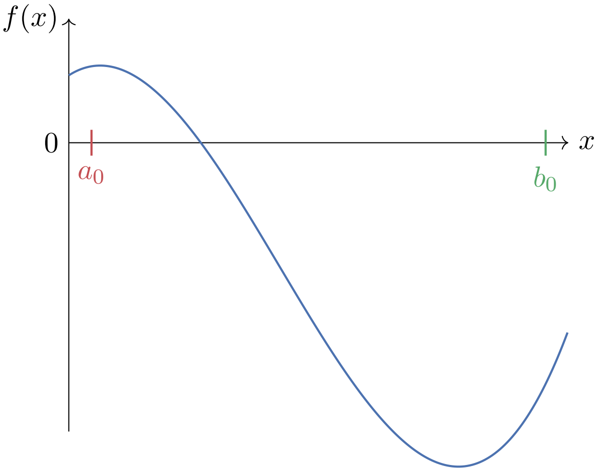
The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

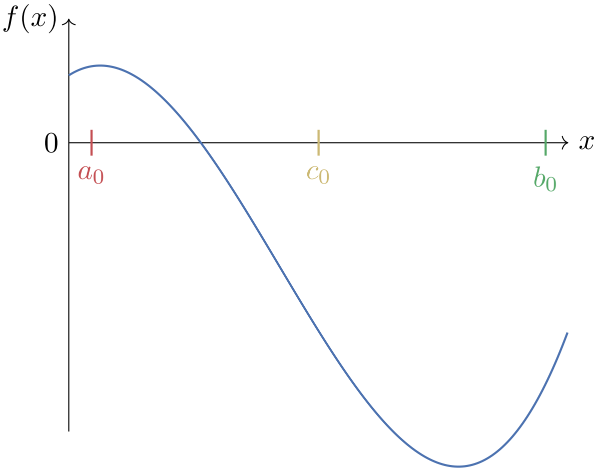
Bernhard Bolzano, 1781–1848

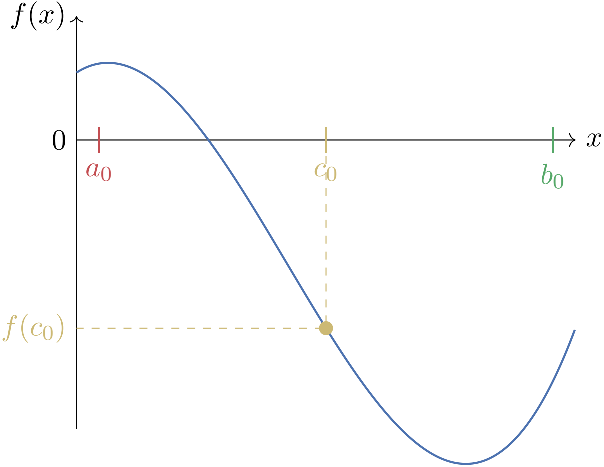
We evaluate f at $c = (a + b)/2$. We then have three possibilities:

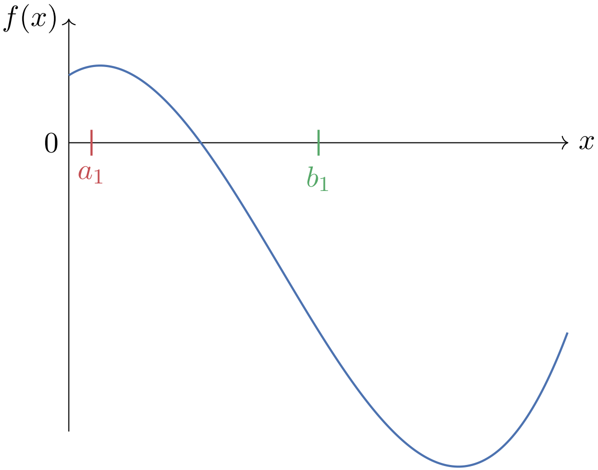
1. $f(c) = 0$, so we are done!
2. $f(c)$ has the same sign as $f(a)$, so there exists a root in (c, b).
3. $f(c)$ has the same sign as $f(b)$, so there exists a root in (a, c).

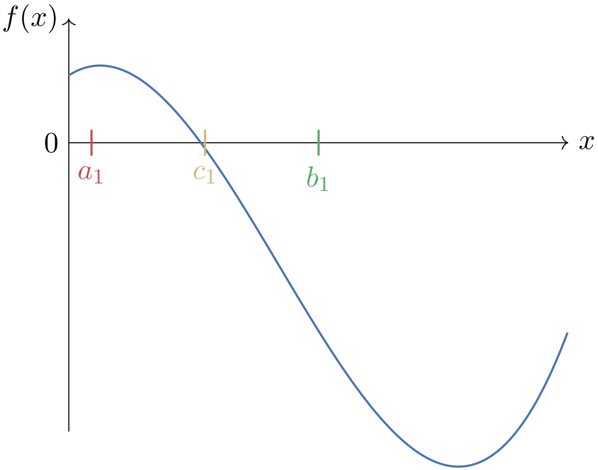

$f(x)$
0
x

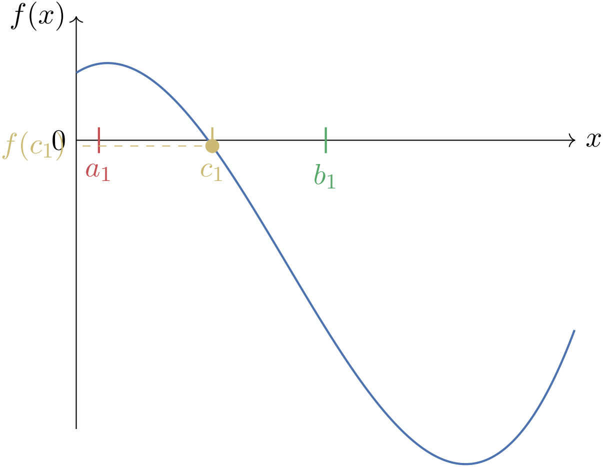

$f(x)$
0
x
a_0
b_0

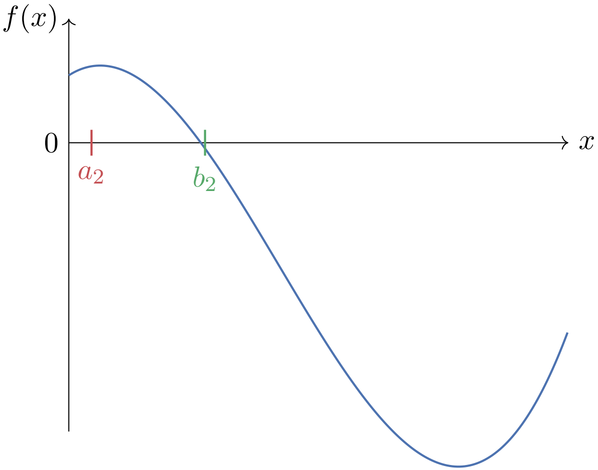

$f(x)$
0
x
a_0
c_0
b_0

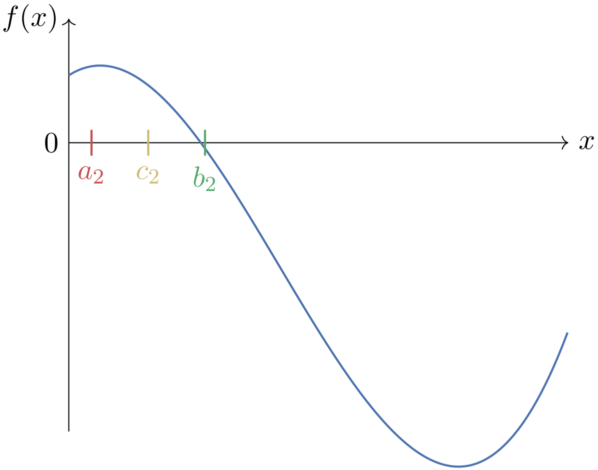

$f(x)$
0
x
a_0
c_0
b_0
$f(c_0)$

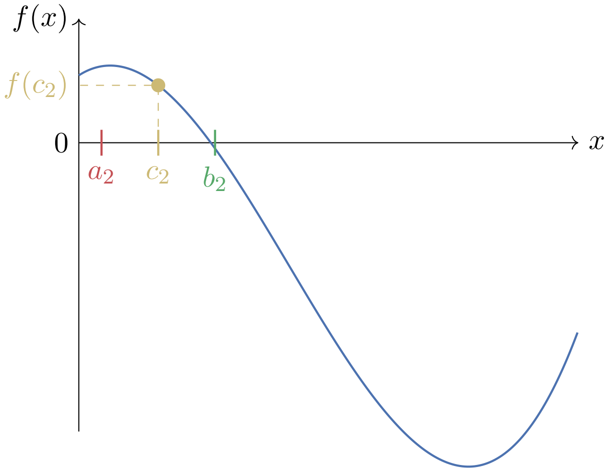

$f(x)$
0
x
a_1
b_1

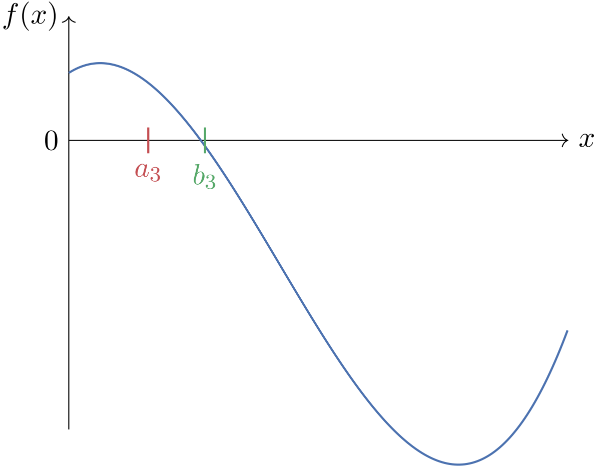

$f(x)$
0
x
a_1
c_1
b_1


$f(x)$
x
0
$f(c_1)$
a_1
c_1
b_1

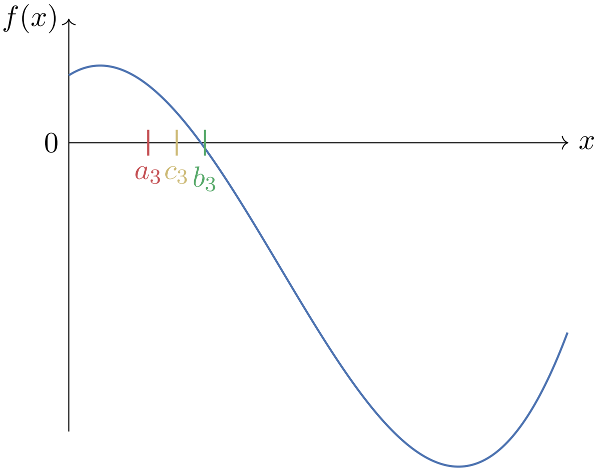

$f(x)$
0
x
a_2
b_2

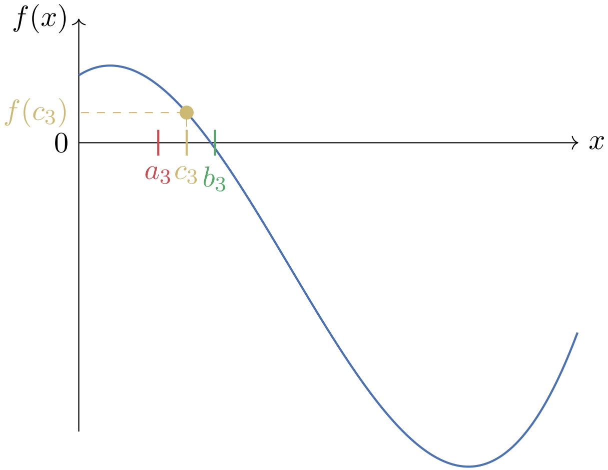

$f(x)$
0
x
a_2
c_2
b_2

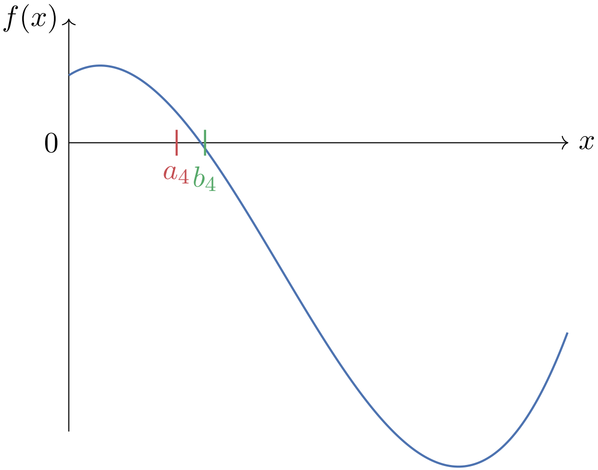

$f(x)$
$f(c_2)$
0
x
a_2
c_2
b_2



$f(x)$
0
x
a_3
b_3


$f(x)$
0
x
a_3
c_3
b_3


$f(x)$
$f(c_3)$
0
x
a_3
c_3
b_3


$f(x)$
0
x
a_4
b_4

$\rightarrow$ `bisection.py`

How can we use this to compute an approximation to π?

Week 5–6 MT teaches

- lists, tuples
- dictionaries, sets,
- functions,
- plotting.

Week 5–6 MT ends with a naïve code for *primality testing*, checking whether a given integer is prime or not.

Week 5–6 MT ends with a naïve code for *primality testing*, checking whether a given integer is prime or not.

$\rightarrow$ `isprime.py`

Can we make `isprime(9999991111111)` faster?

Week 7–8 MT introduces *symbolic computing*, the use of computers to automate the kind of mathematical manipulations you do on paper.

This includes expanding and simplifying expressions, differentiating and integrating functions, calculating limits, and solving equations.

Week 7–8 MT introduces *symbolic computing*, the use of computers to automate the kind of mathematical manipulations you do on paper.

This includes expanding and simplifying expressions, differentiating and integrating functions, calculating limits, and solving equations.

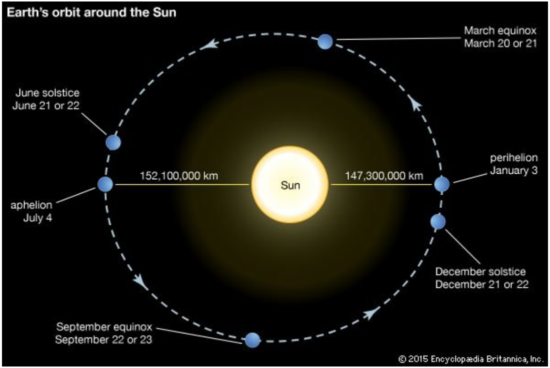
In 1843, describing Charles Babbage's Analytical Engine, Ada Lovelace wrote

> *Many persons who are not conversant with mathematical studies imagine that because the business of the engine is to give its results in numerical notation, the nature of its processes must consequently be arithmetical and numerical rather than algebraic and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraic notation were provisions made accordingly.*

Ada Lovelace, 1815–1852

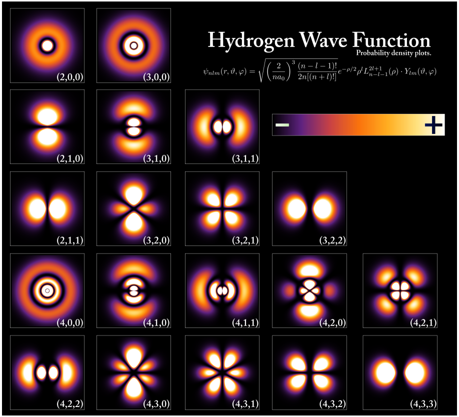
In the associated problem sheet, we use symbolic computing to

- derive the equations for the orbit of the Earth around the Sun;



In the associated problem sheet, we use symbolic computing to

- derive the equations for the orbit of the Earth around the Sun;
- explore the wave function of the hydrogen atom.

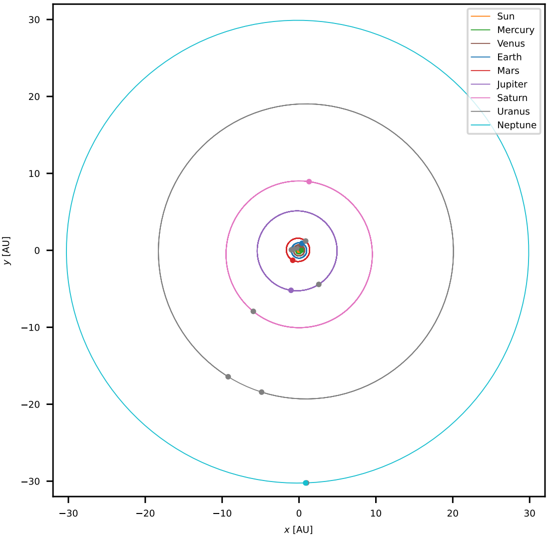


Week 1–2 HT introduces *numerical* computing, a powerful expansion of the conception of what it means to solve a mathematical problem.

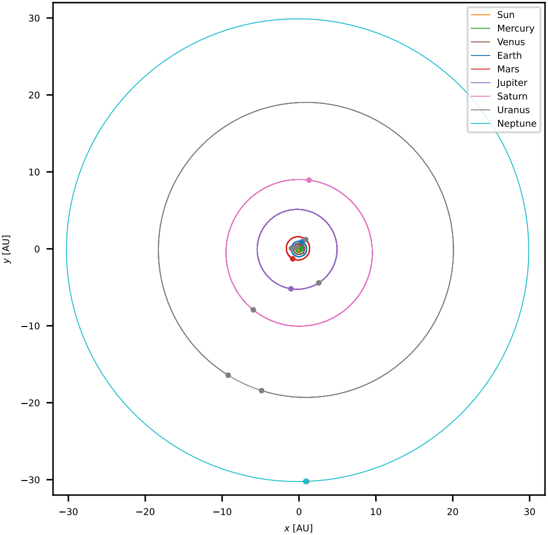
We will study

- numerical linear algebra,
- numerical quadrature of integrals,
- least squares and curve-fitting,
- numerical solution of ODE initial value problems.

Week 1–2 HT ends with a code for numerically simulating the solar system.



Week 1–2 HT ends with a code for numerically simulating the solar system.



→ `solar.py`

Section 3

Software installation

$\rightarrow$ Windows