# 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:

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

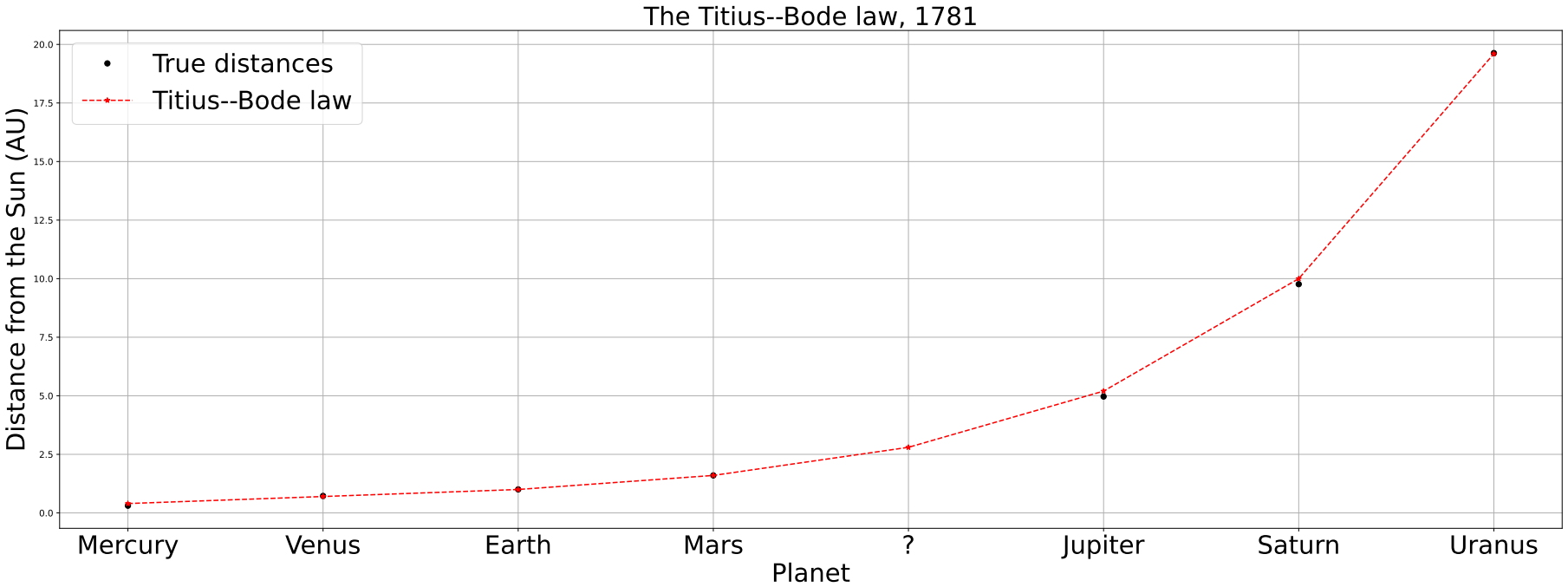
William Herschel, 1738–1822

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.

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.

Carl Friedrich Gauss, 1777–1855

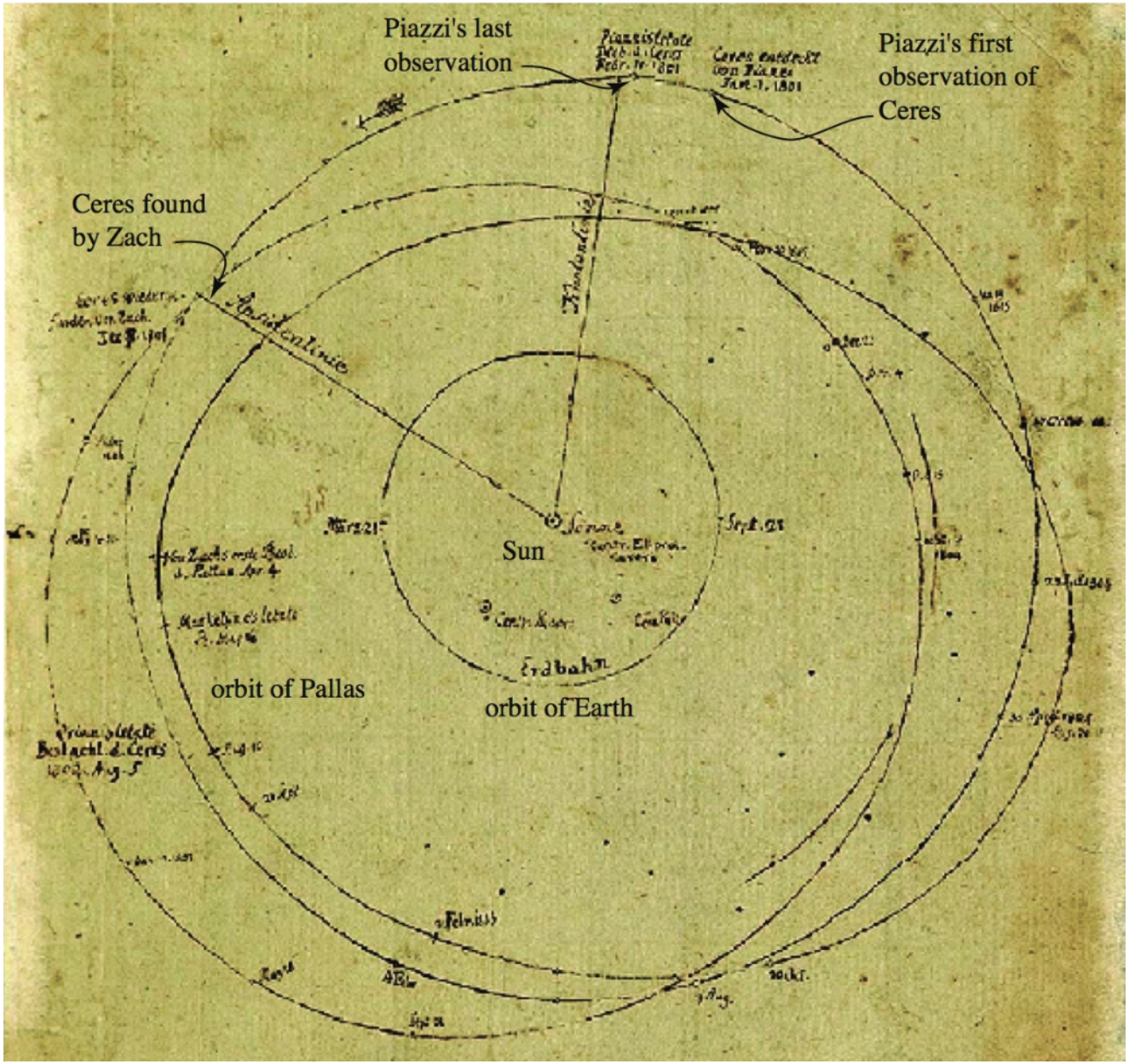
In Brunswick, the 24-year old Gauss had invented the *method of least squares*, for estimating coefficients of mathematical expressions from partial and noisy data.

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.

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 $n$ $n$th powers are required to sum to an $n$th 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

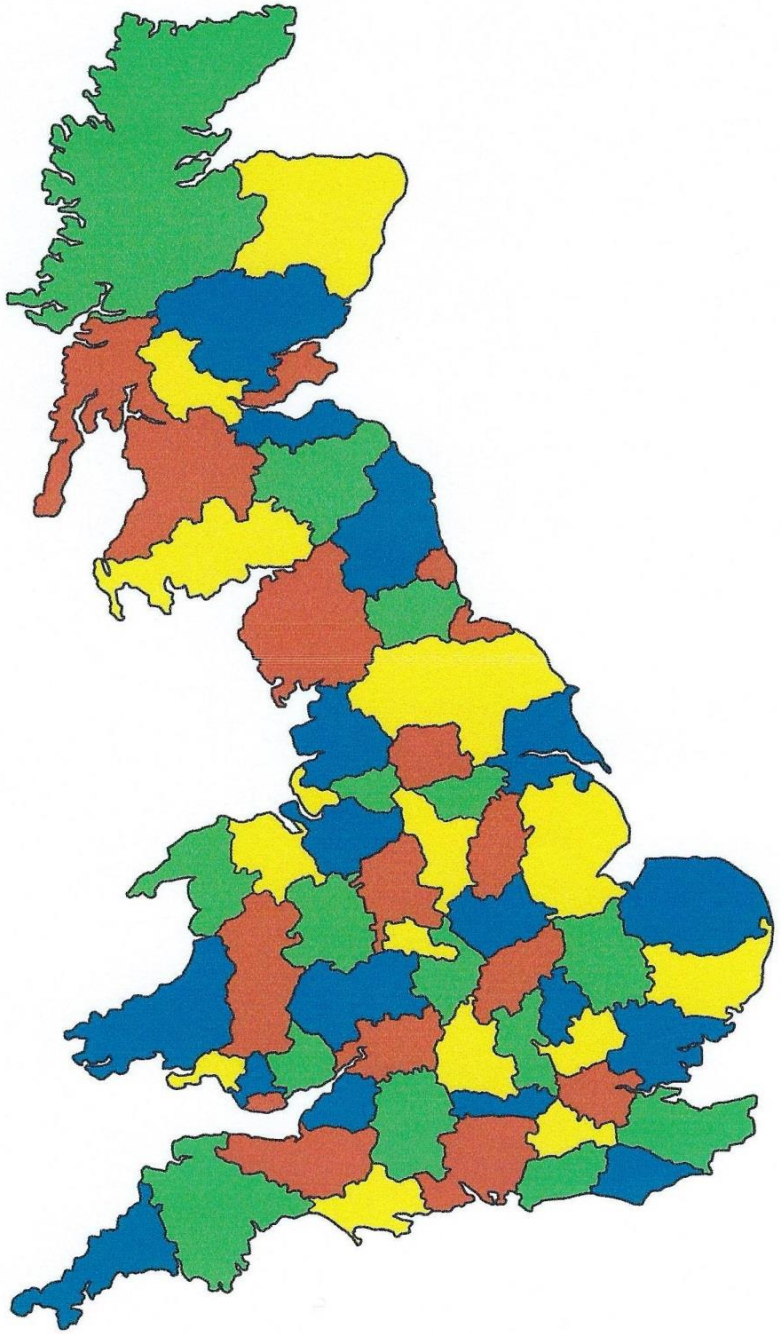
Computers are not just useful for finding counterexamples, though. They can help us find *proofs*.

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.

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.

Francis Guthrie, 1831–1899



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.

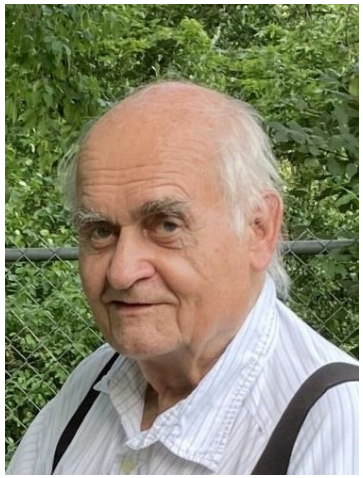
While the proof was wrong, the basic strategy was right.

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



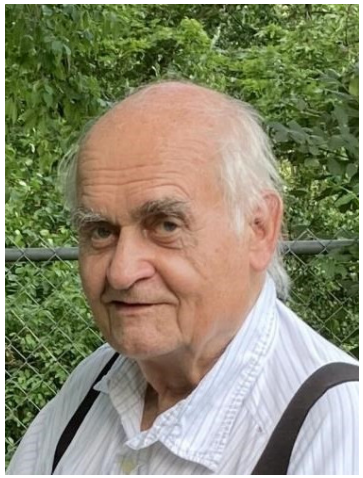
Wolfgang Haken, 1928–2022

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.

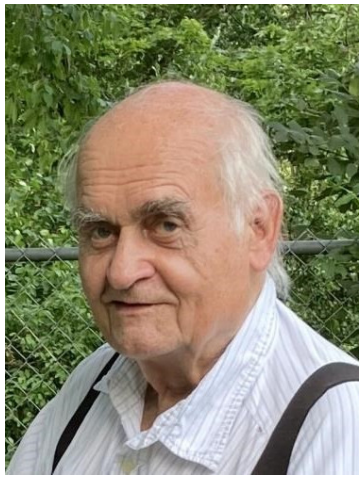
With a computer, they found an unavoidable set with 1834 cases, and programmed it to mechanically check that in each case the graph can be coloured with four colours.

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

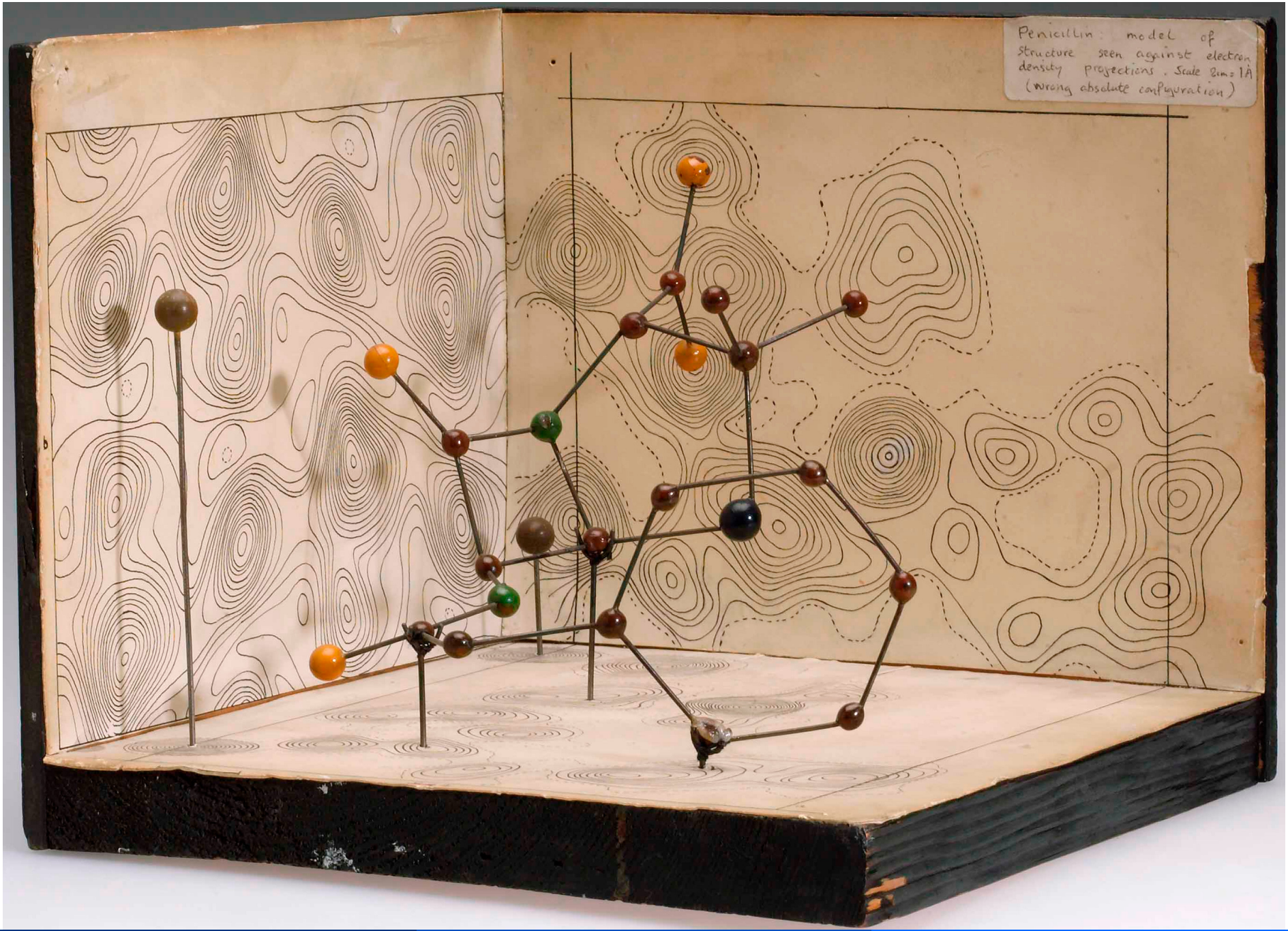
Dorothy Hodgkin was one of Oxford's pioneers of computational mathematics.

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.

Dorothy Hodgkin, 1910–1994

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.


Penicillin: model of
structure seen against electron
density projections. Scale 2in = 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

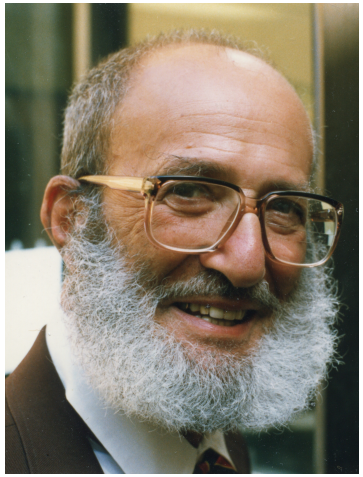
Computational mathematics is the subject that studies the use of computation to solve mathematical problems.

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

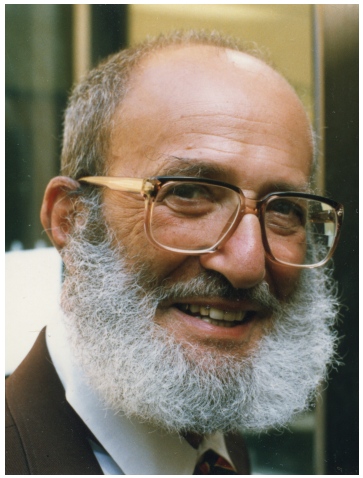
> *When you try to prove a theorem, you don't just list the hypotheses, and then start to reason. What you do is trial and error, experimentation, guesswork. You want to find out what the facts are.*



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.*

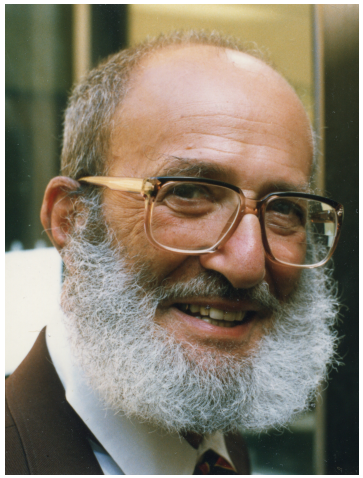


Paul Halmos, 1916–2006

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

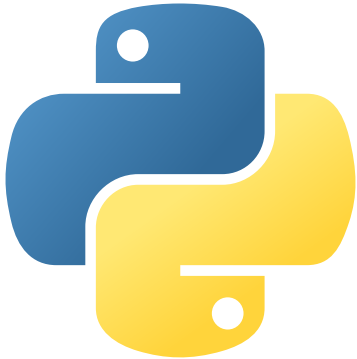
- to solve mathematical problems with computers;
- along the way to learn to program computers.

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!

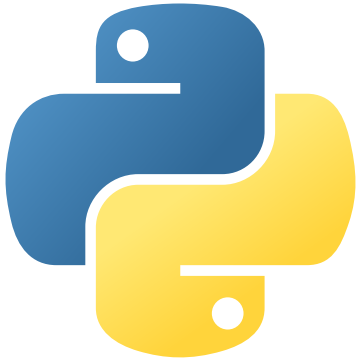
On a pragmatic point, a very large fraction of Oxford mathematics graduates will pursue careers where programming is useful, if not essential. These include

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

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

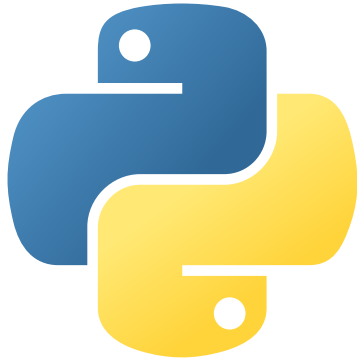


In this course, we will learn the Python programming language.



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.

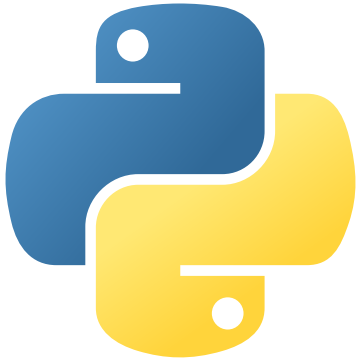


Python is one of the world's most popular programming languages. In particular, it is the leading language in scientific data analysis and machine learning.

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=4931`

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=4931`

| Weeks | Chapters to read | Optional chapters | Problem sheet to start |
|---|---|---|---|
| 1–2 MT | 1–3 | - | - |
| 3–4 MT | 4–5 | - | 1 |
| 5–6 MT | 7 | 8 | 2 |
| 7–8 MT | 10 | - | 3 |
| 1–2 HT | 12 | to come | 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=4931`

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

- $1^{\text{st}}$ project: 12 noon on Monday of week 6 HT24
- $2^{\text{nd}}$ project: 12 noon on Monday of week 9 HT24

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 term. 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

- $1^{\text{st}}$ project: 12 noon on Monday of week 6 HT24
- $2^{\text{nd}}$ project: 12 noon on Monday of week 9 HT24

These submissions must be your own unaided work.

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)

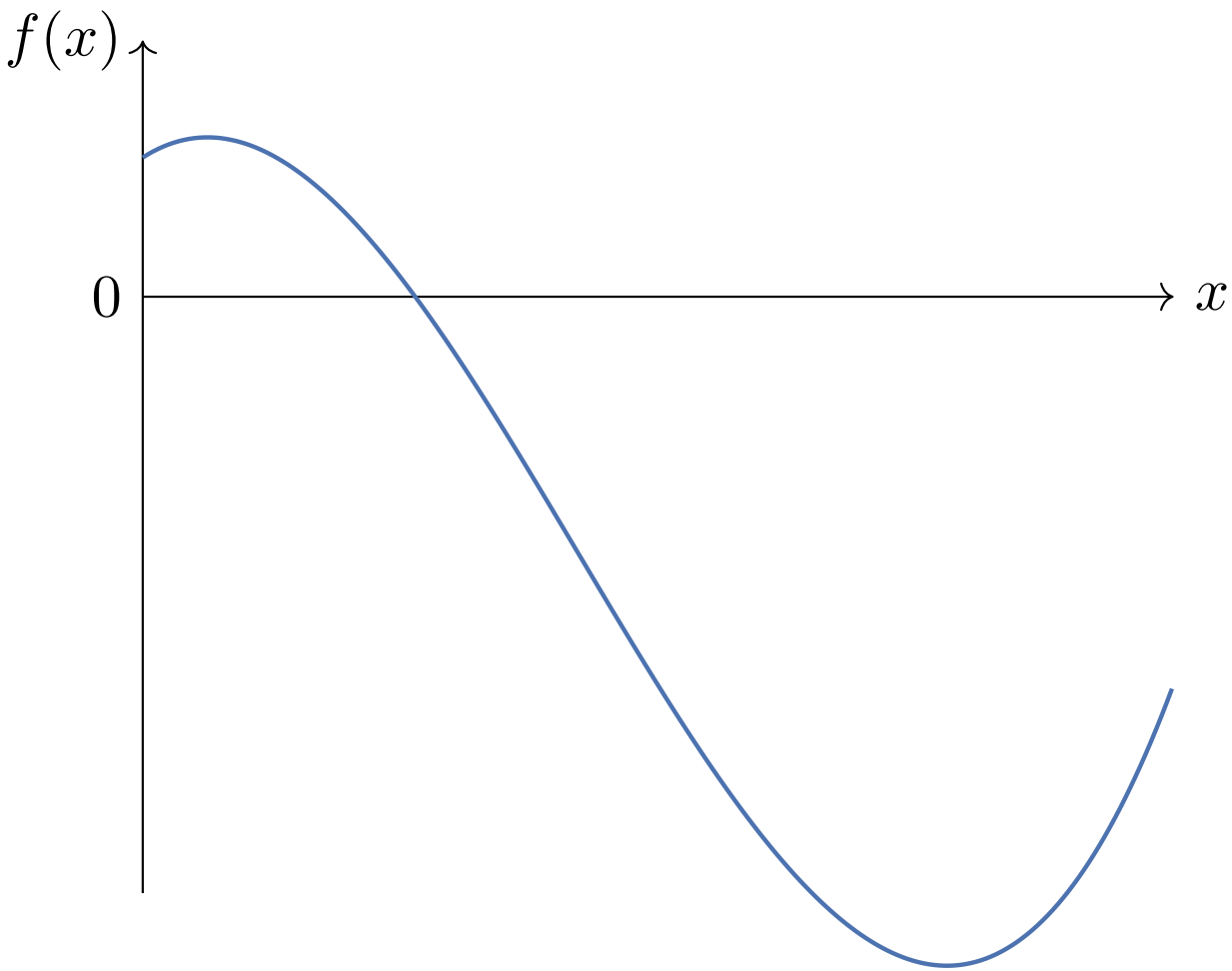
*If $f : [a, b] \to \mathbb{R}$ is continuous with $f(a)f(b) < 0$, then there exists $x^\star \in (a, b)$ with $f(x^\star) = 0$.*

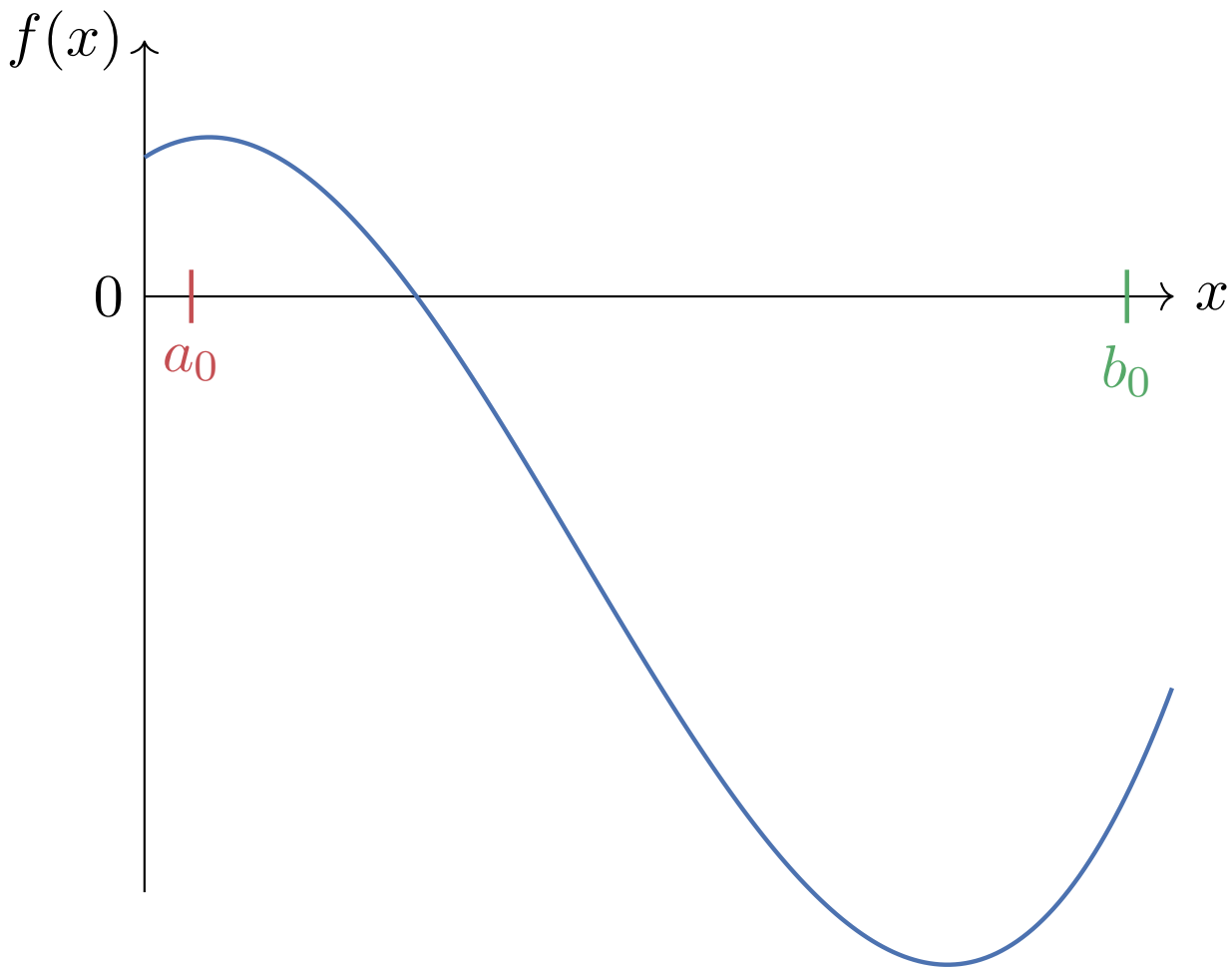
The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

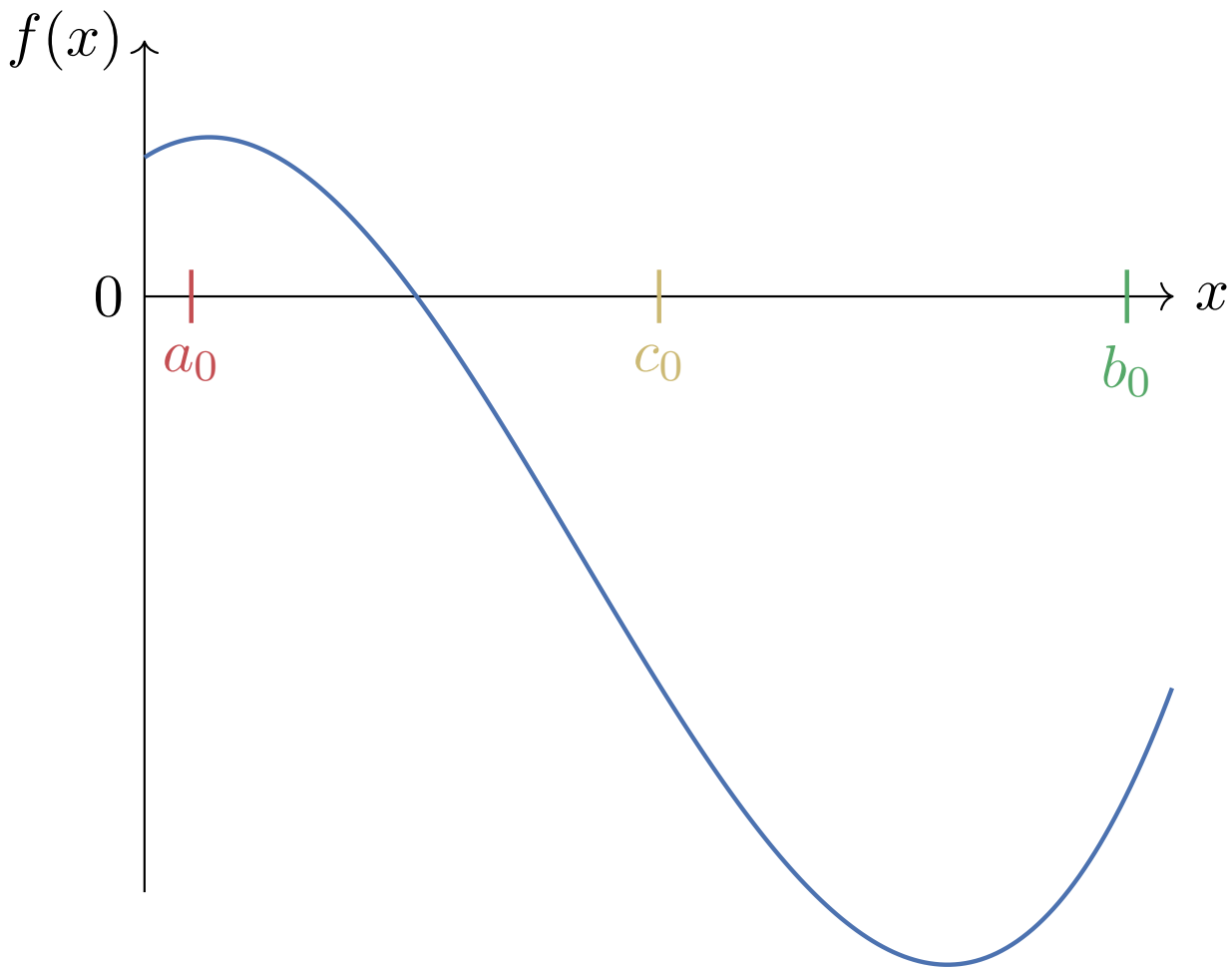
Bernhard Bolzano, 1781–1848

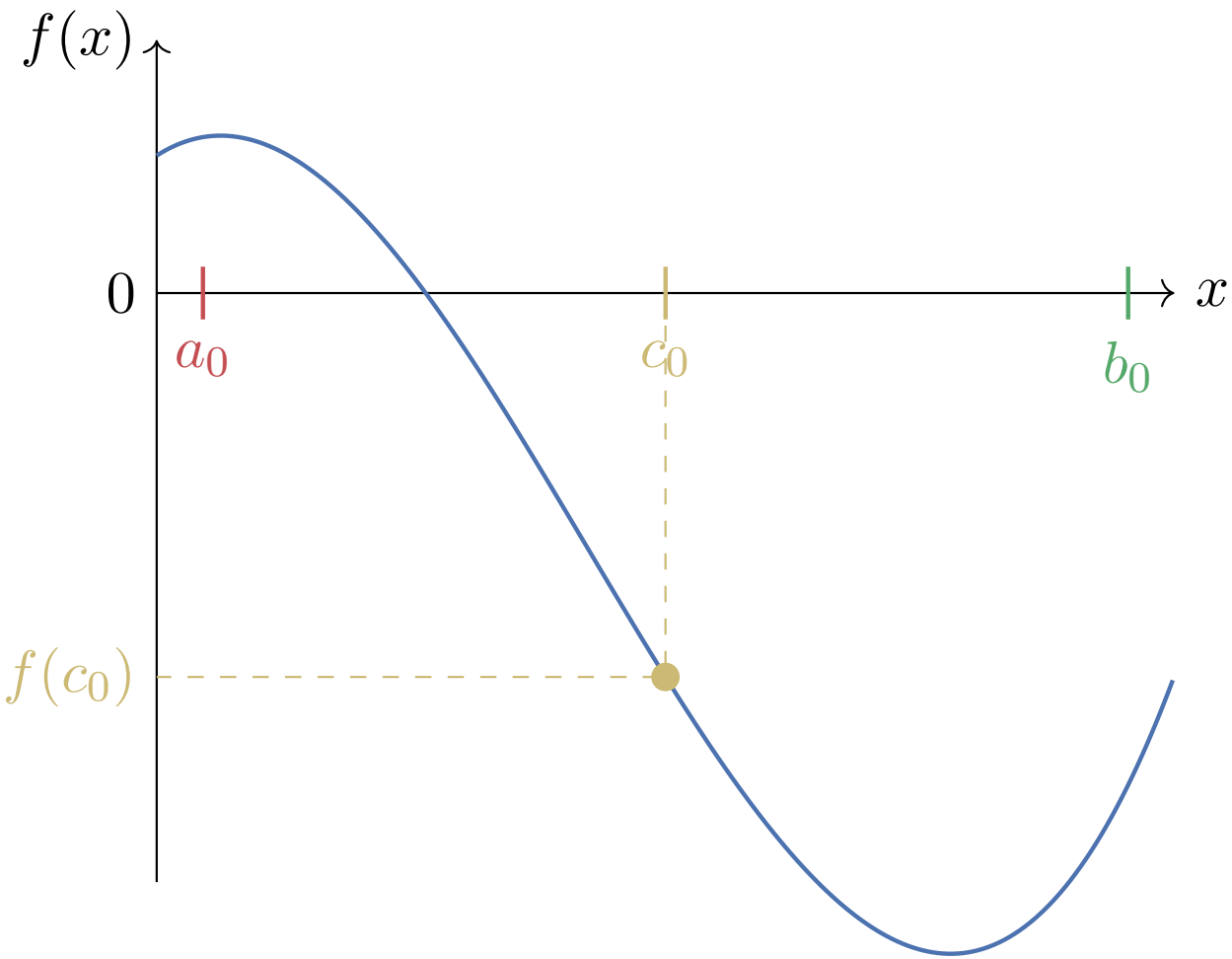
We evaluate $f$ at $c = (a + b)/2$. We then have three possibilities:

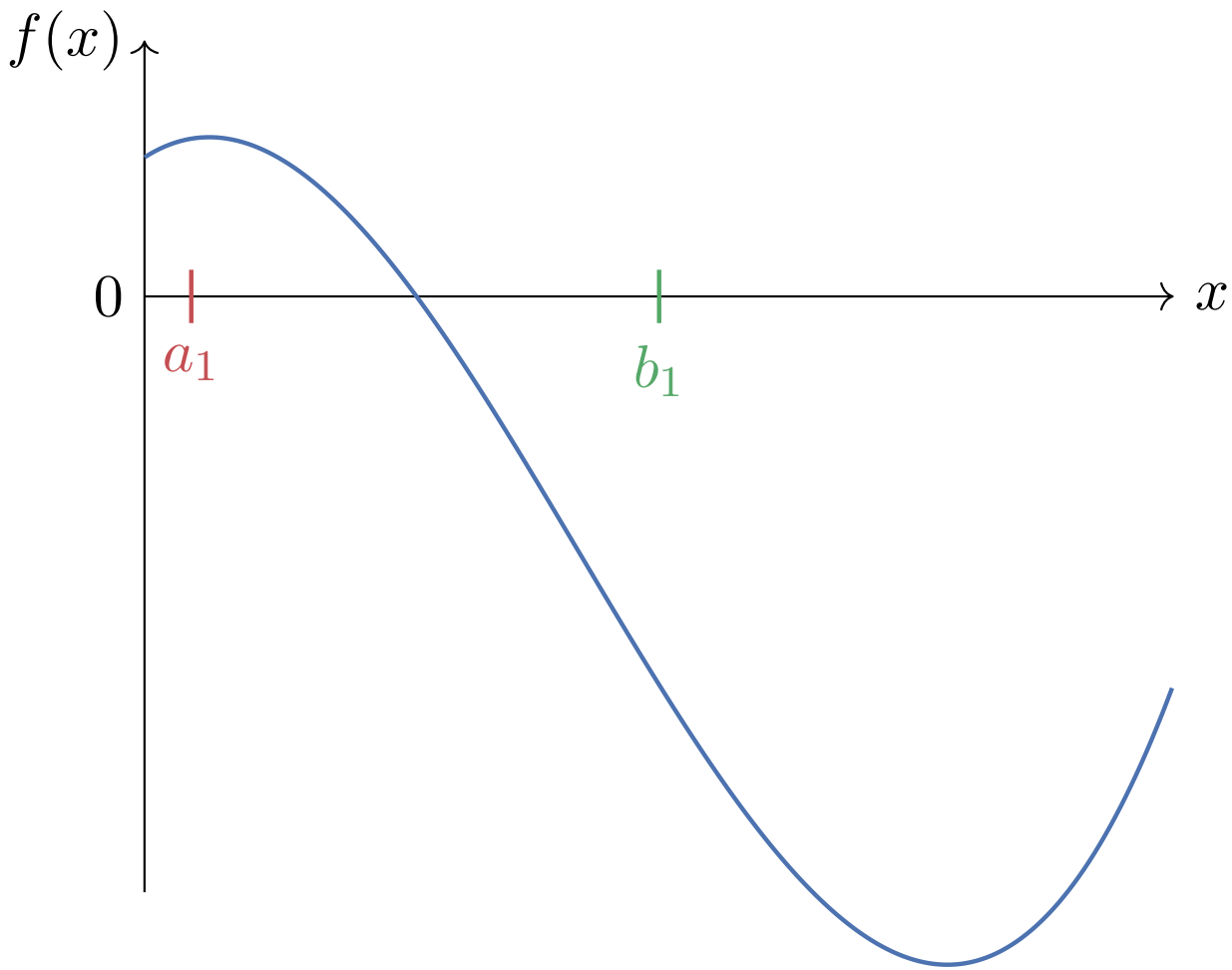
1. $f(c) = 0$, so we are done!
2. $f(c)$ has the same sign as $f(a)$, so there exists a root in $(c, b)$.
3. $f(c)$ has the same sign as $f(b)$, so there exists a root in $(a, c)$.

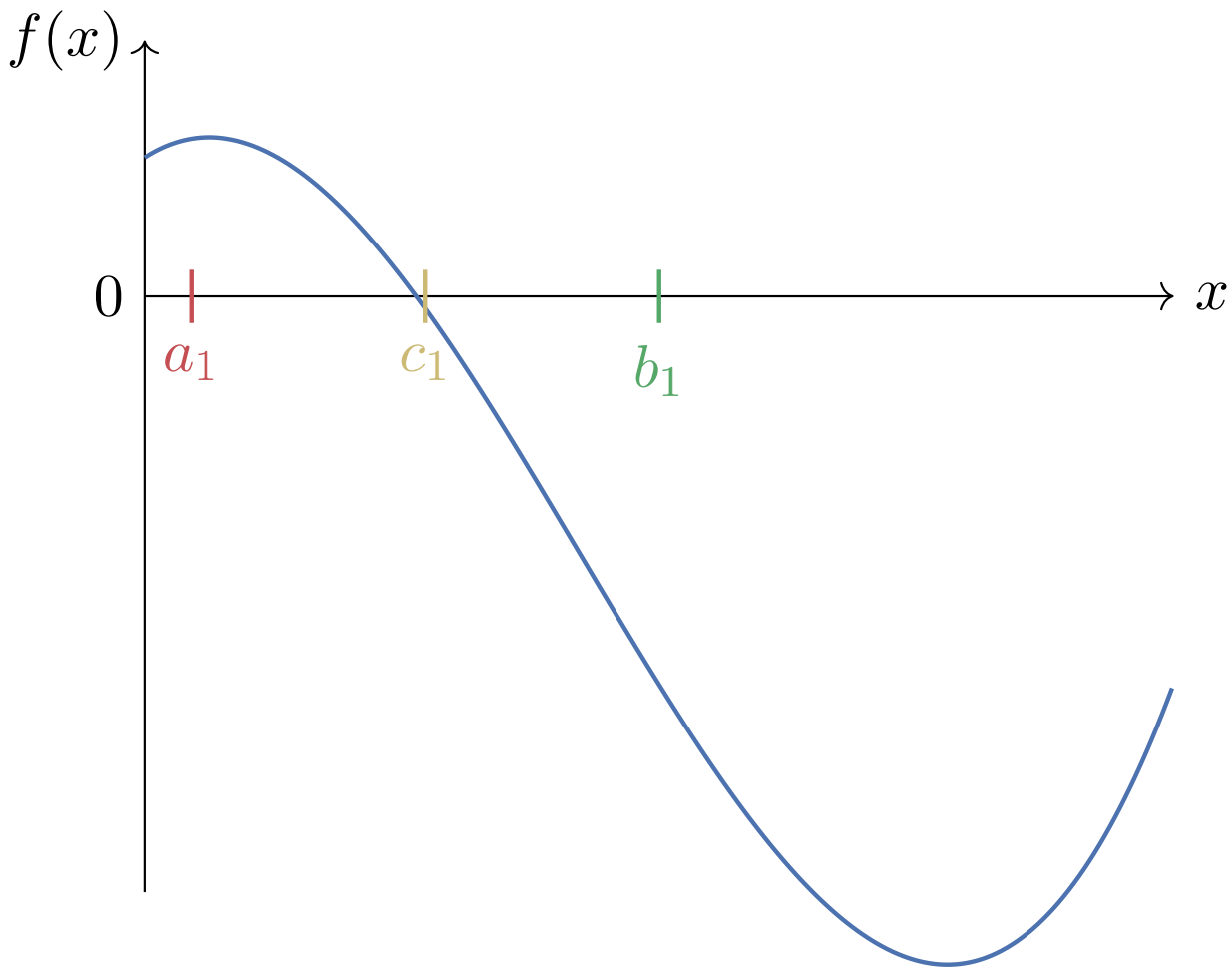

$f(x)$
0
$x$

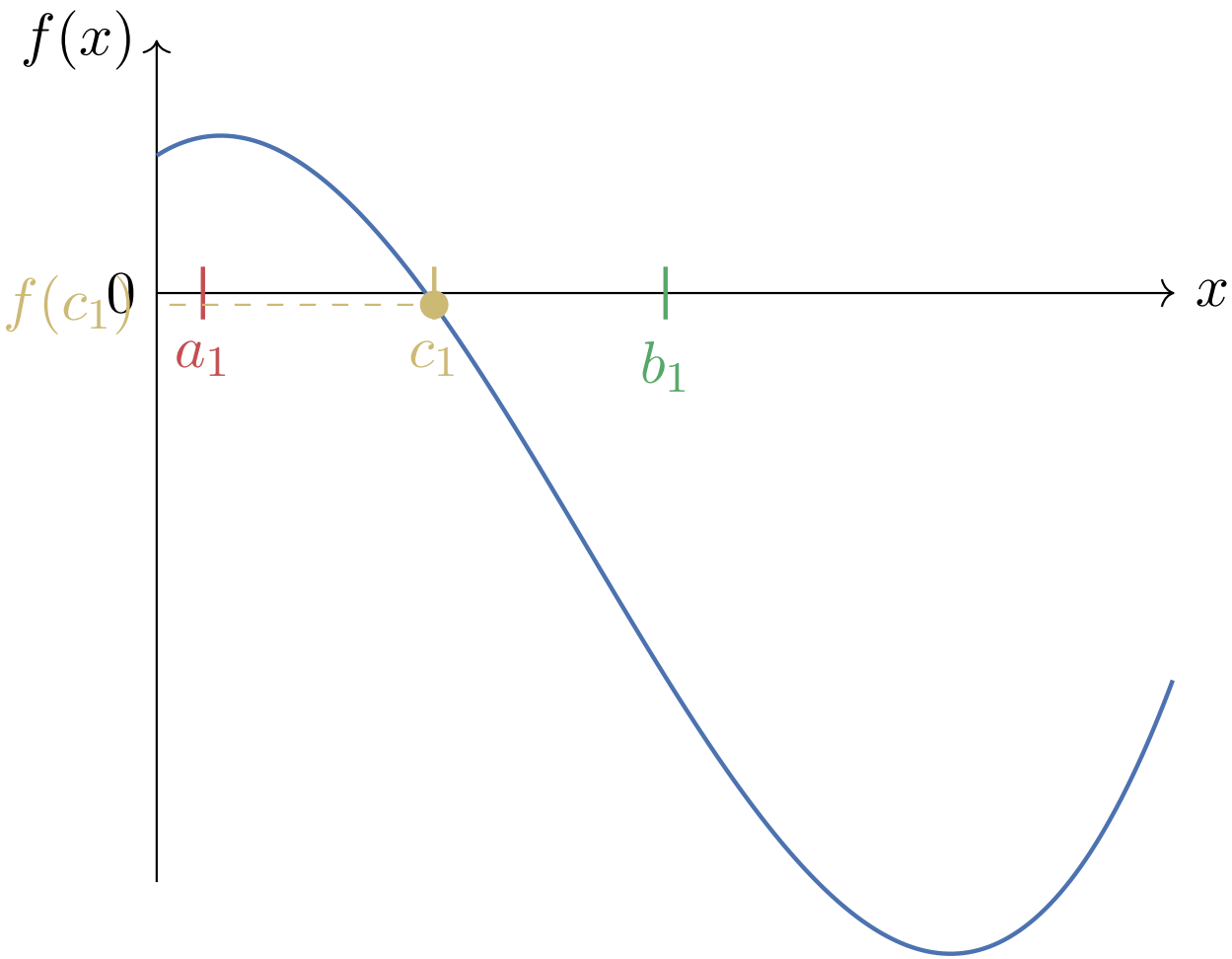

$f(x)$
0
$a_0$
$b_0$
$x$

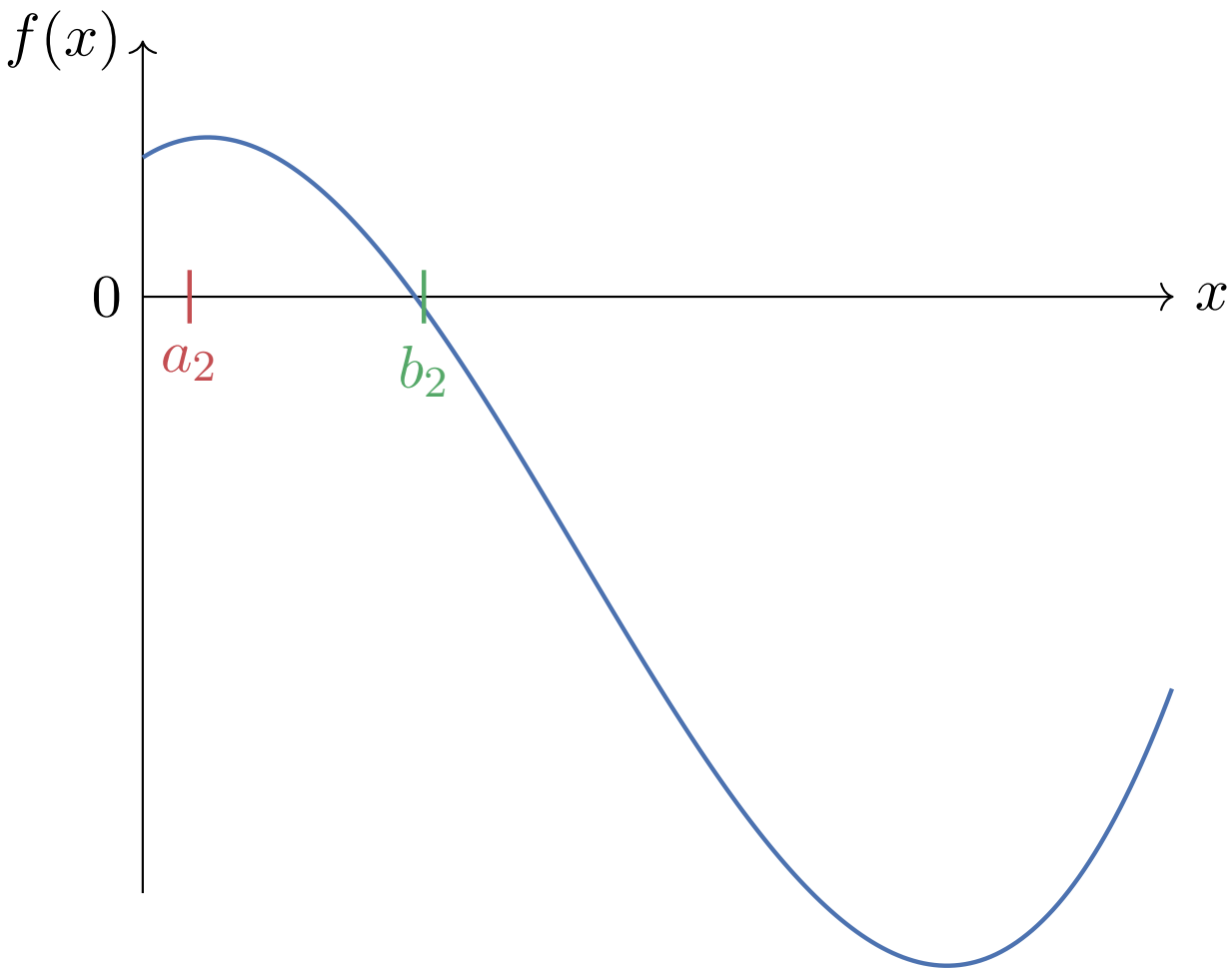

$f(x)$
0
$x$
$a_0$
$c_0$
$b_0$

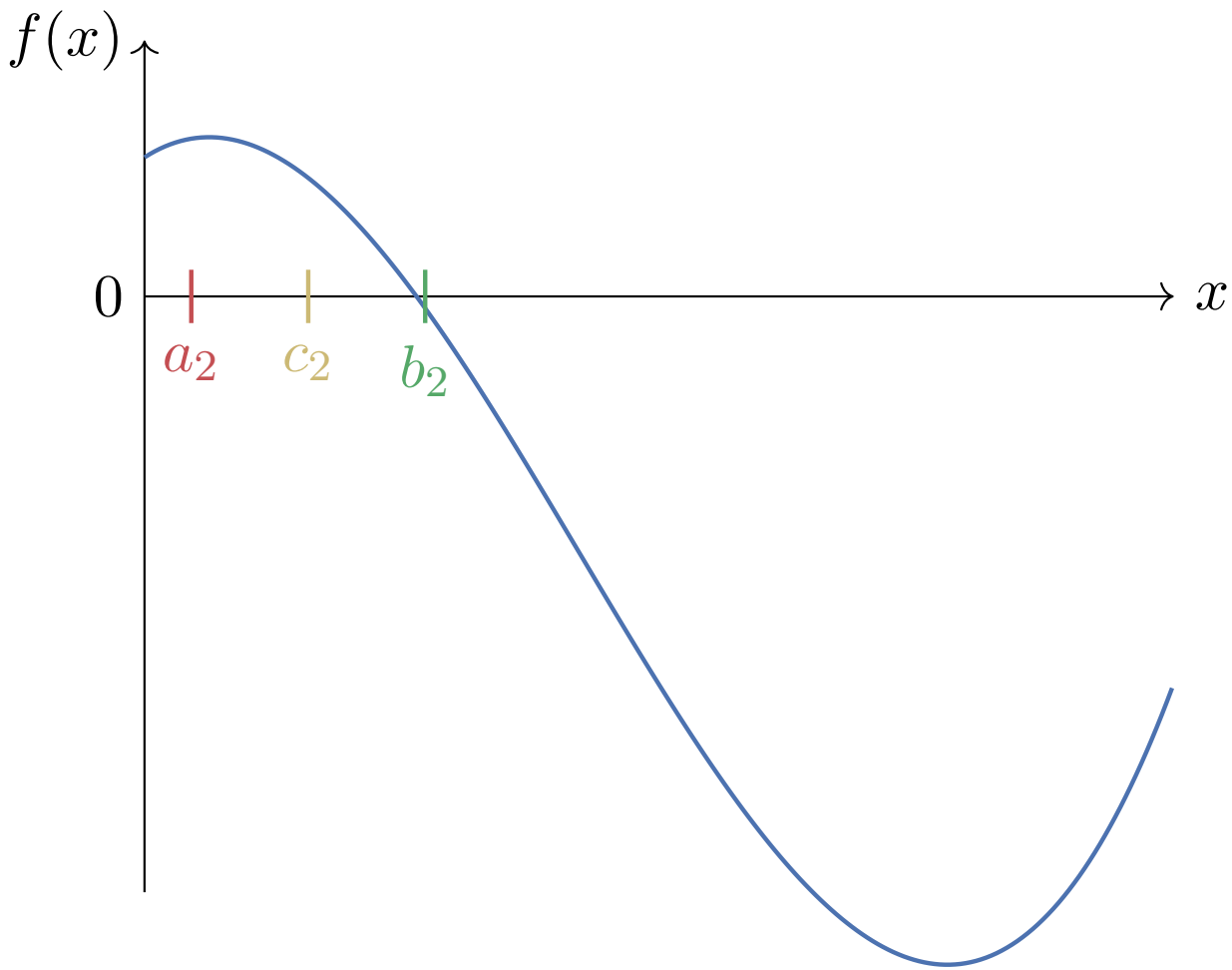

$f(x)$
$0$
$x$
$a_0$
$c_0$
$b_0$
$f(c_0)$

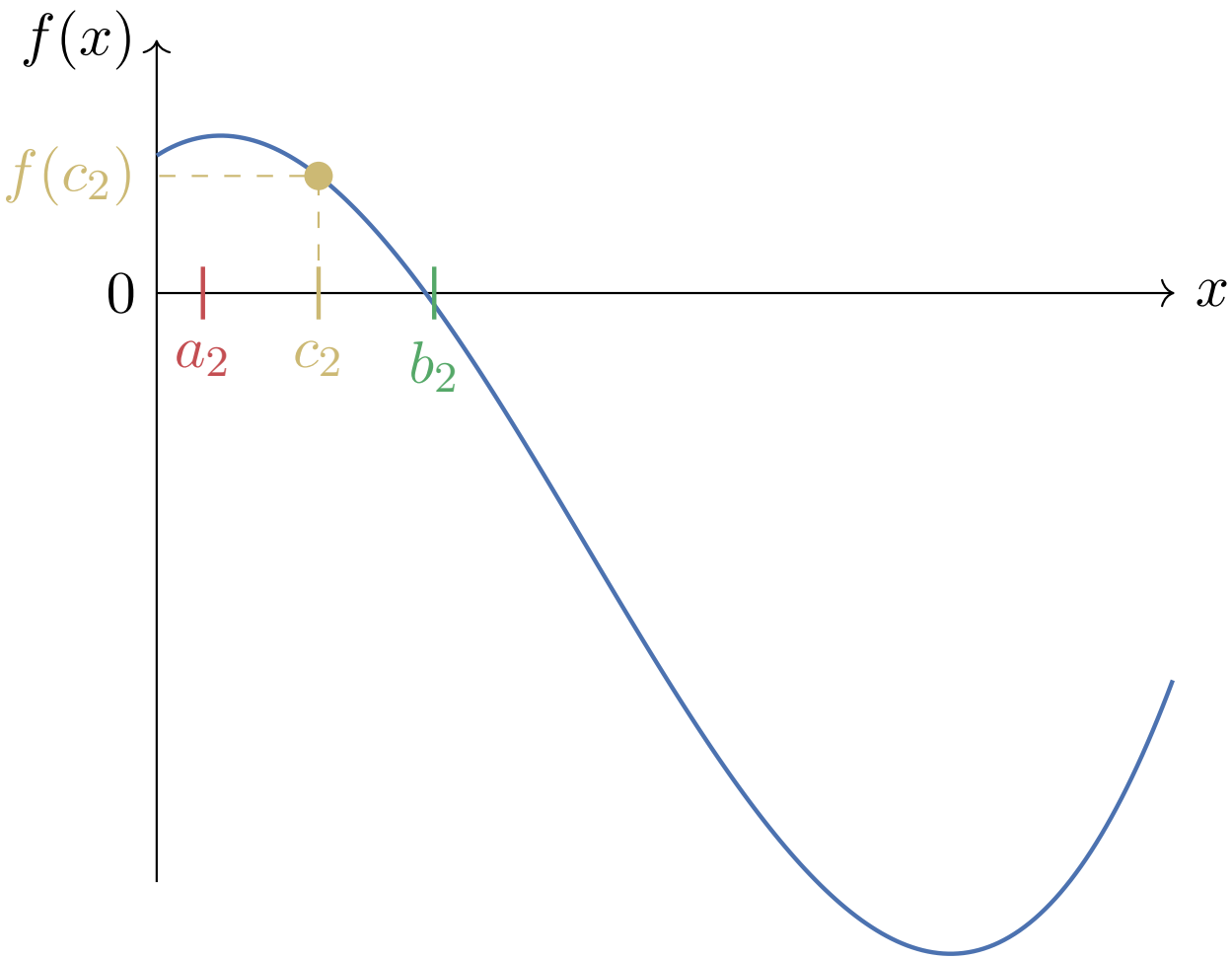

$f(x)$
0
$x$
$a_1$
$b_1$

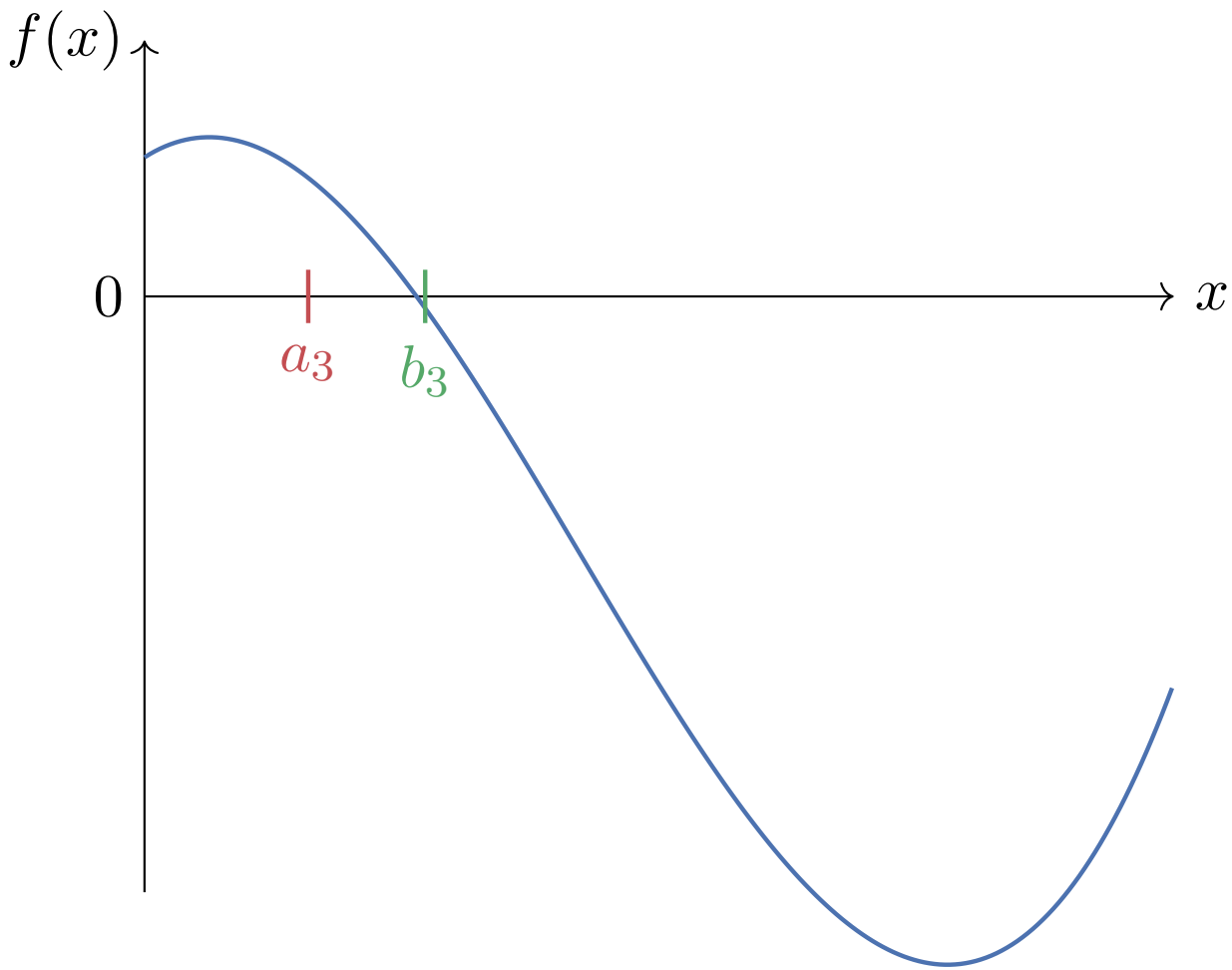

$f(x)$
0
$x$
$a_1$
$c_1$
$b_1$

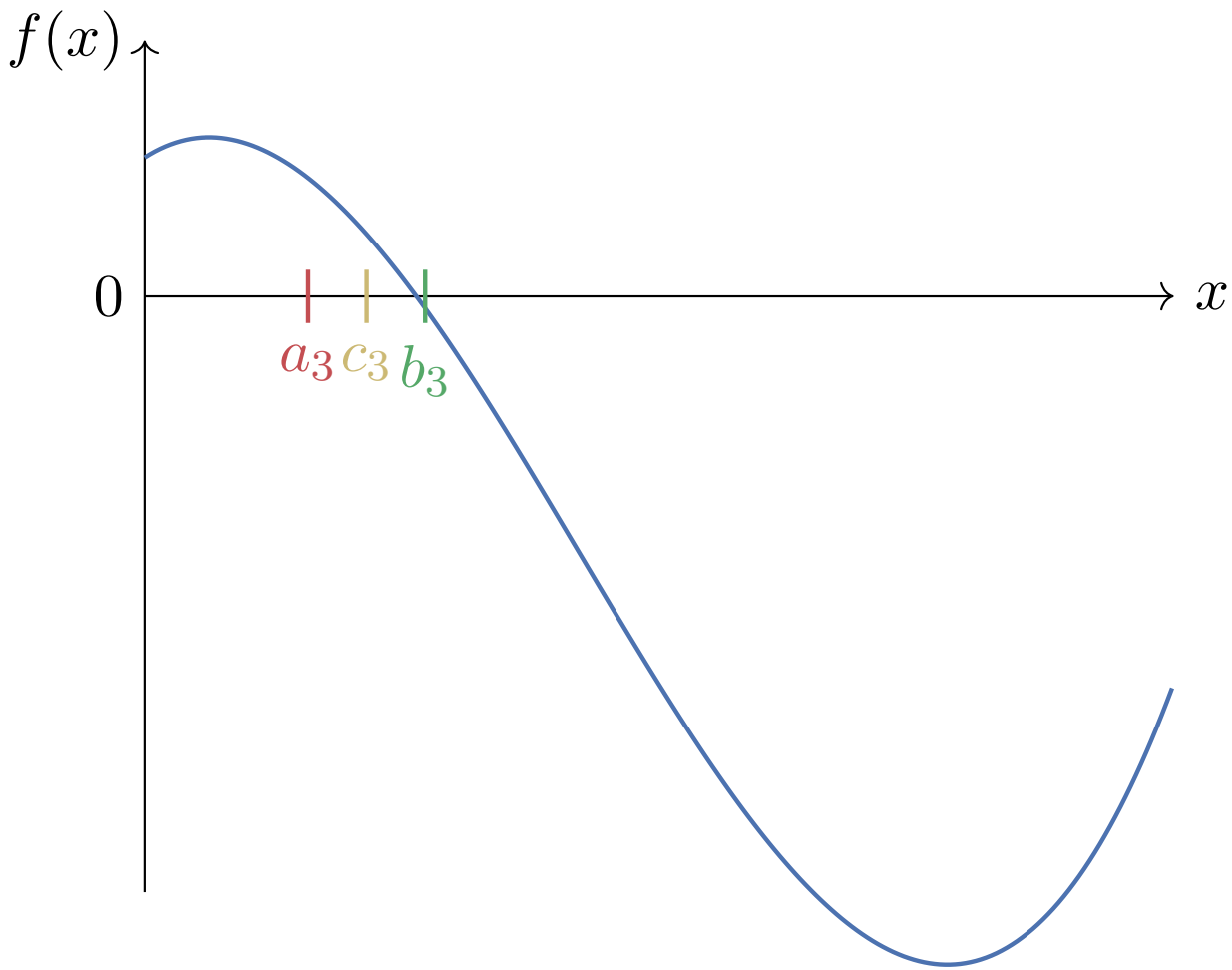

$f(x)$
$f(c_1)0$
$x$
$a_1$
$c_1$
$b_1$

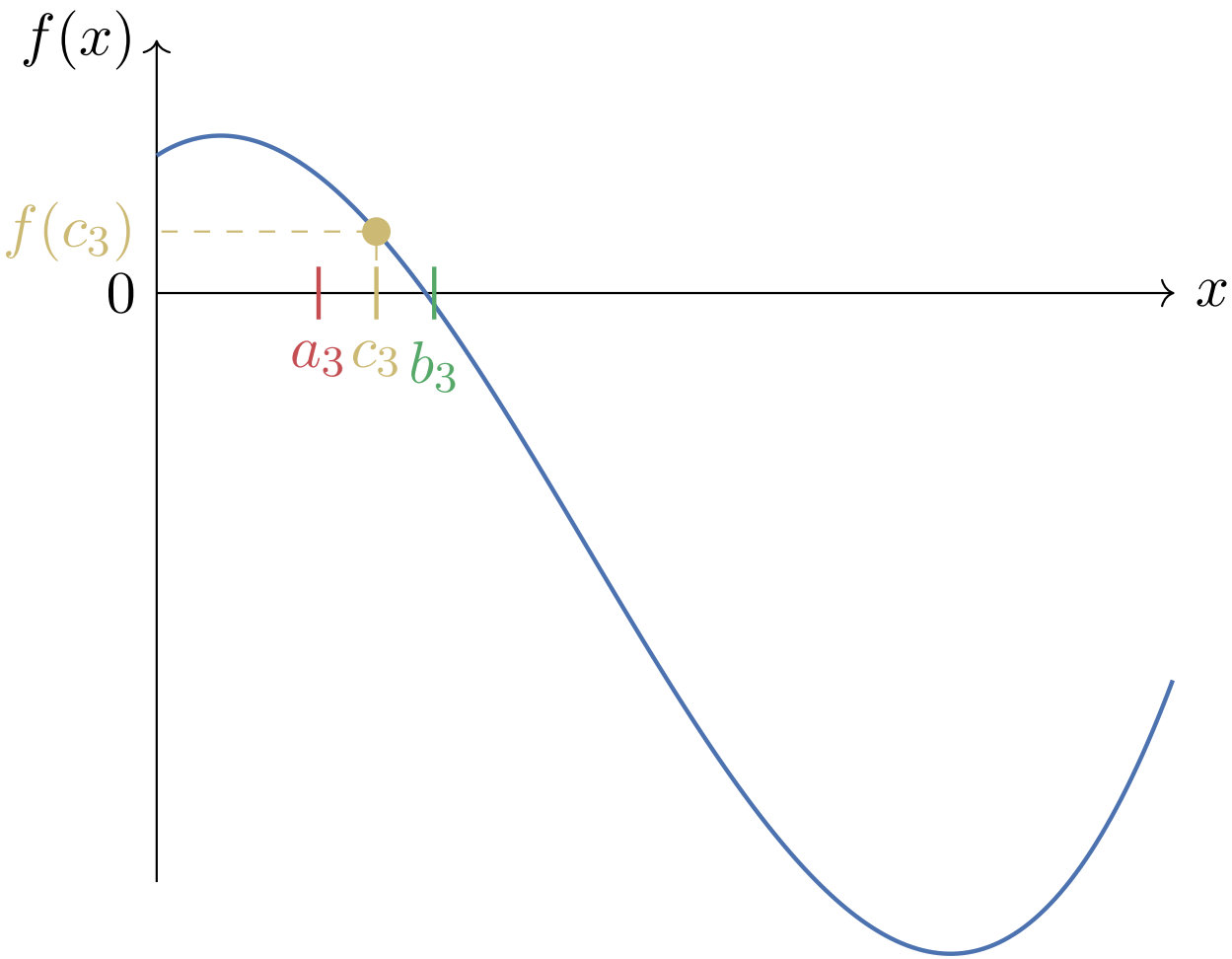

$f(x)$
0
$x$
$a_2$
$b_2$

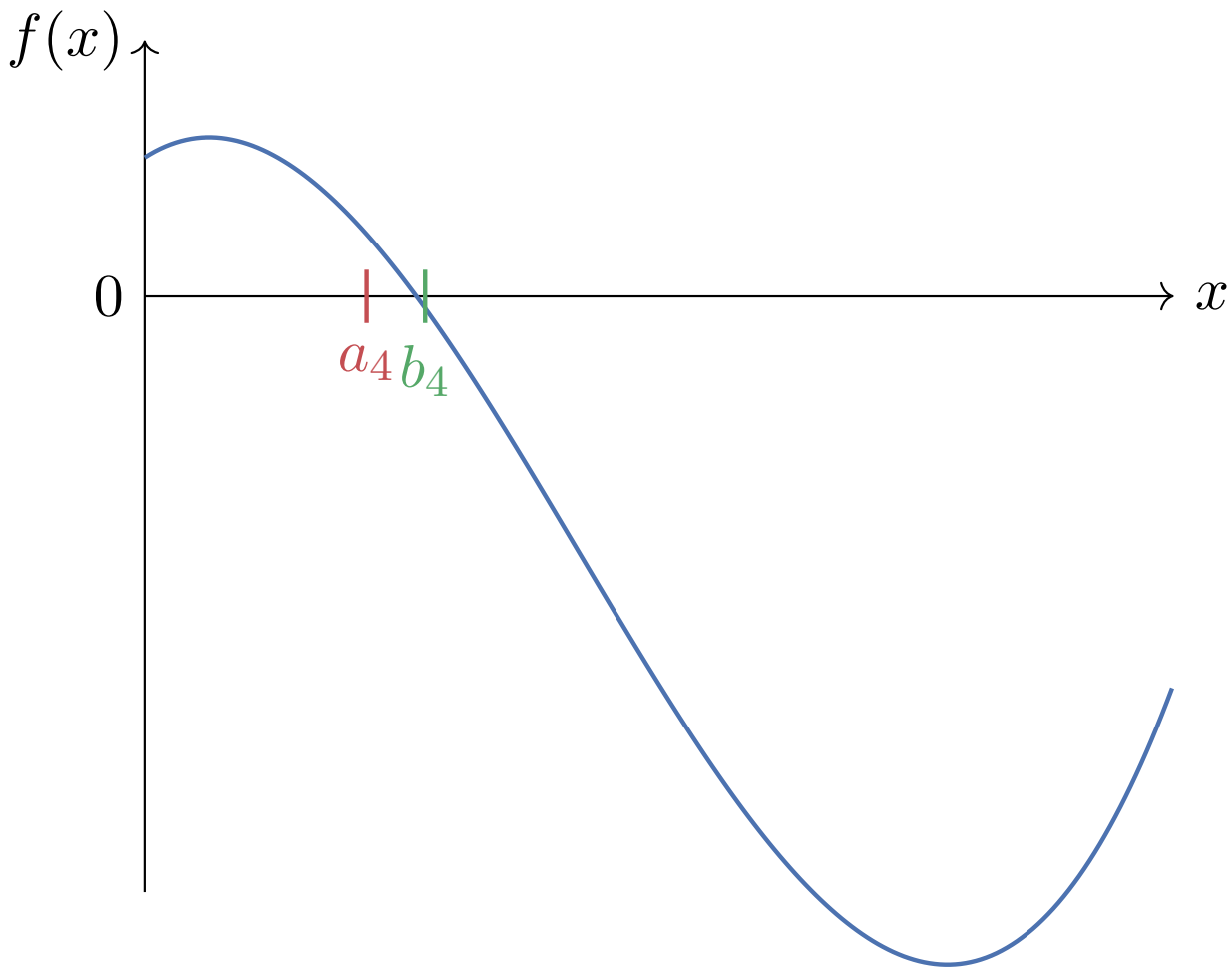

$f(x)$
0
$x$
$a_2$
$c_2$
$b_2$


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


$f(x)$
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 $\pi$?

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.

→ `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.

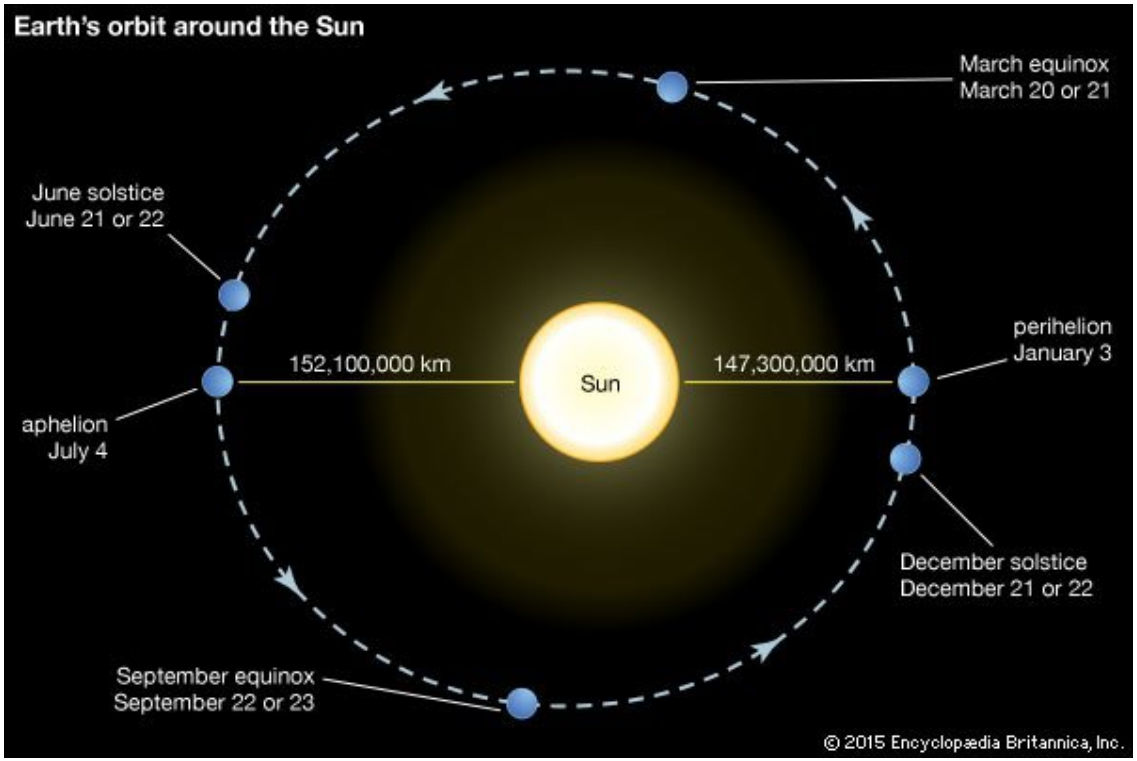
In 1843, describing Charles Babbage's Analytical Engine, Ada Lovelace wrote

> *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

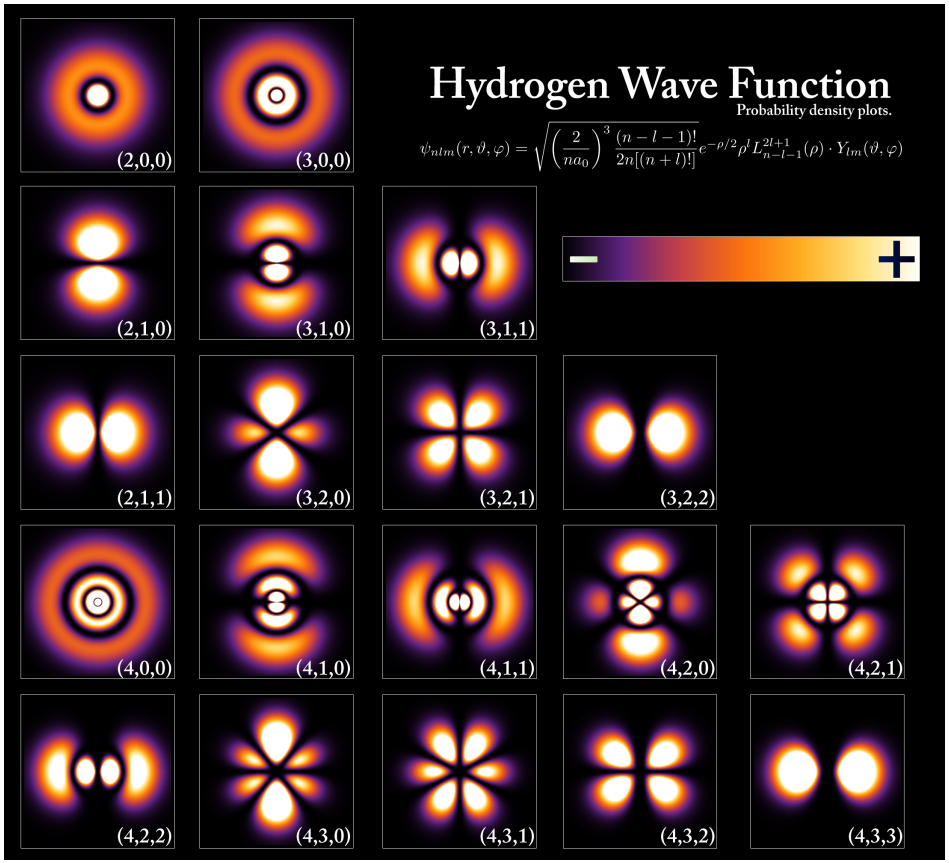
In the associated problem sheet, we use symbolic computing to

- 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.

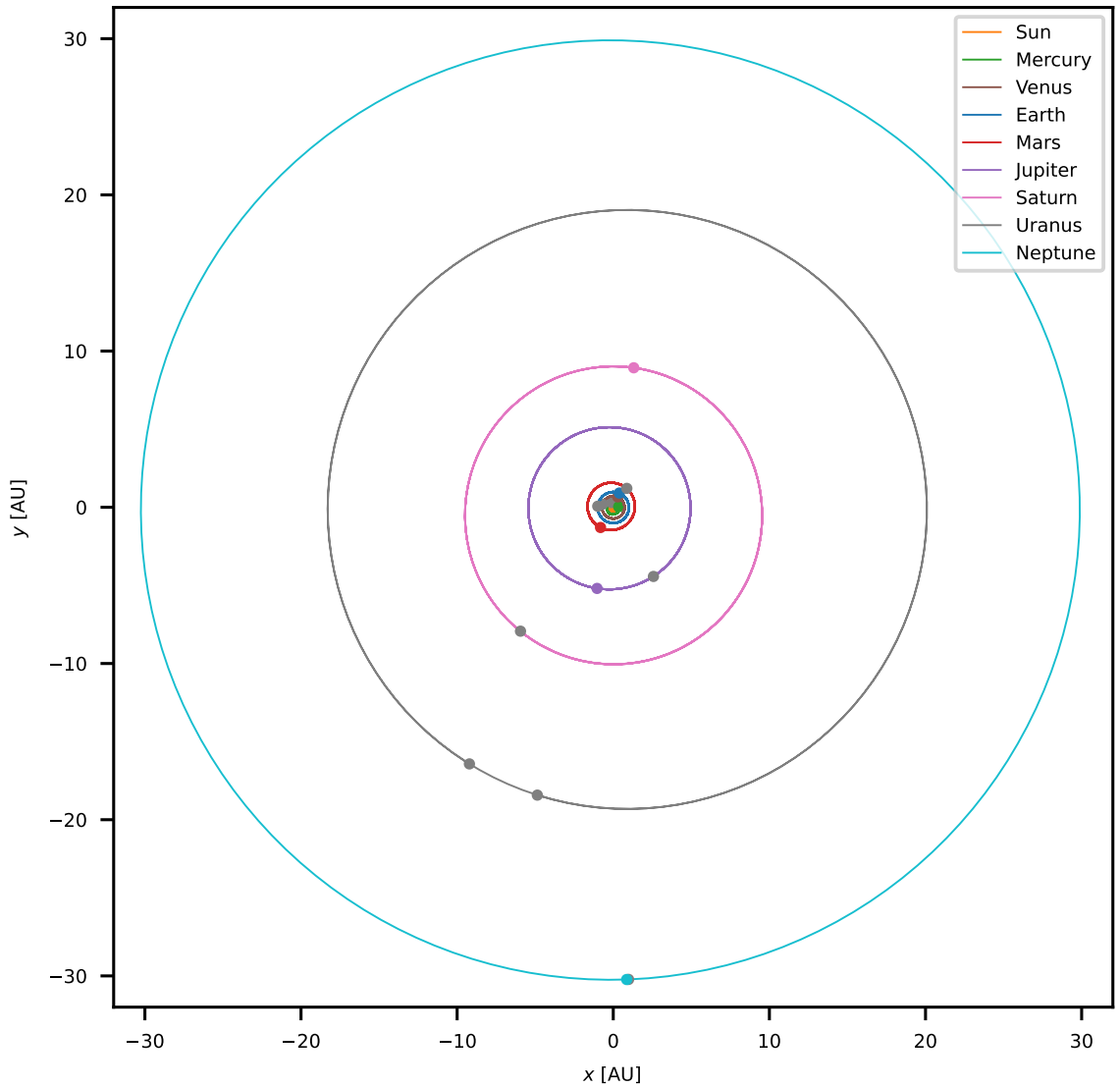


Week 1–2 HT introduces *numerical* computing, a powerful expansion of the conception of what it means to solve a mathematical problem.

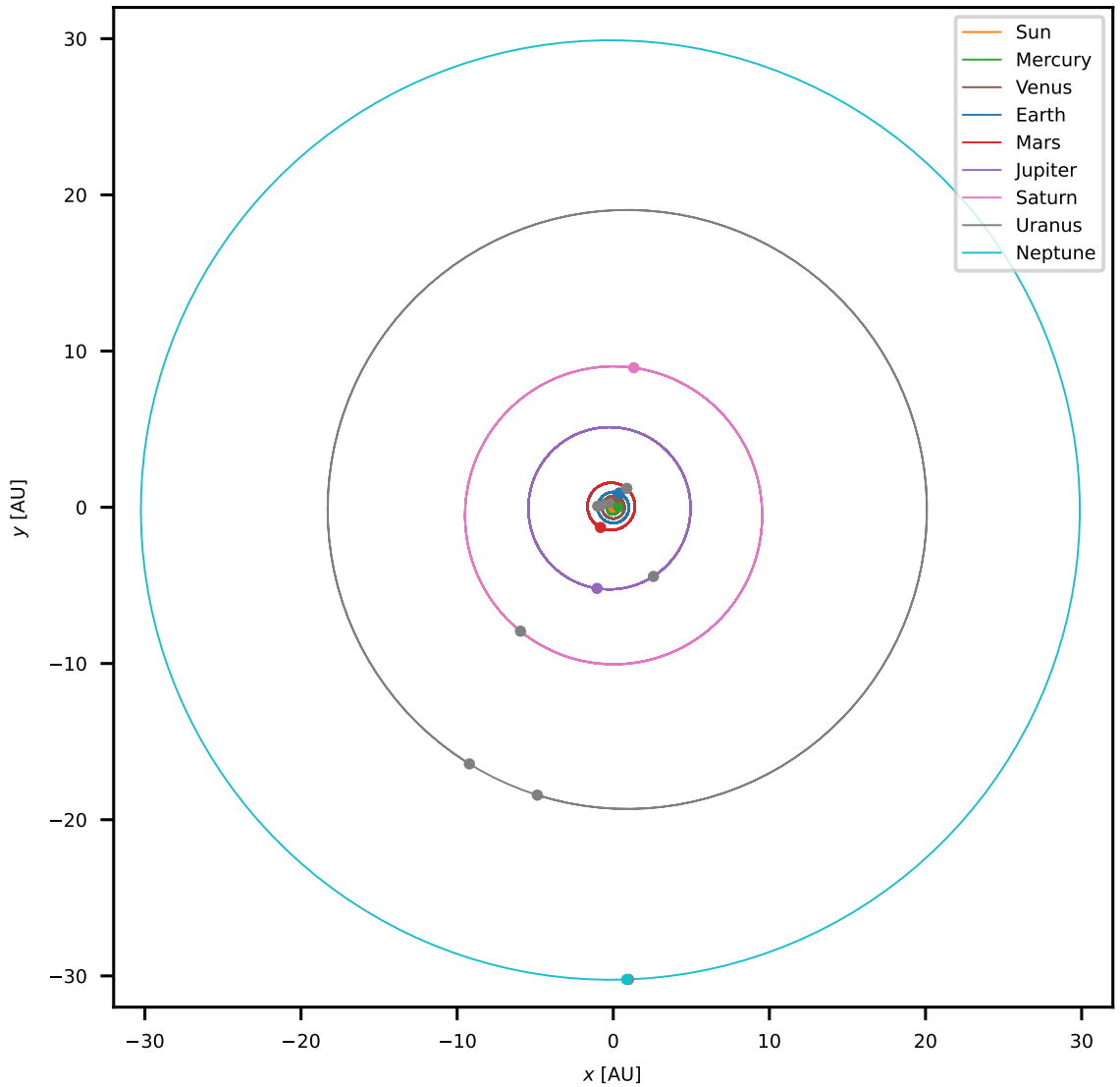
We will study

- 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

# Computational Mathematics
# Projects

Patrick E. Farrell

University of Oxford

Section 1

# Overview

Computational Mathematics is assessed by projects.

Computational Mathematics is assessed by projects.

You must complete two projects out of three offered.

Computational Mathematics is assessed by projects.

You must complete two projects out of three offered.

You can do the projects in any order.

Computational Mathematics is assessed by projects.

You must complete two projects out of three offered.

You can do the projects in any order.

Together, the two projects count for one third of a Prelims paper.

Computational Mathematics is assessed by projects.

You must complete two projects out of three offered.

You can do the projects in any order.

Together, the two projects count for one third of a Prelims paper.

Passing Computational Mathematics is necessary to pass Prelims.

The deadlines for these projects are

The deadlines for these projects are

- $1^{\text{st}}$ project: 12 noon on Monday of week 6 HT24 (19 Feb)

The deadlines for these projects are

- $1^{\text{st}}$ project: 12 noon on Monday of week 6 HT24 (19 Feb)
- $2^{\text{nd}}$ project: 12 noon on Monday of week 9 HT24 (11 Mar)

The deadlines for these projects are

- $1^{\text{st}}$ project: 12 noon on Monday of week 6 HT24 (19 Feb)
- $2^{\text{nd}}$ project: 12 noon on Monday of week 9 HT24 (11 Mar)

Please submit online via Inspera before these deadlines.

The deadlines for these projects are

- $1^{\text{st}}$ project: 12 noon on Monday of week 6 HT24 (19 Feb)
- $2^{\text{nd}}$ project: 12 noon on Monday of week 9 HT24 (11 Mar)

Please submit online via Inspera before these deadlines.

The University imposes mark penalties for late submission.

In weeks 1 and 2, the course continues the same as last term.
Demonstrator sessions are scheduled according to college, to support you in completing the final problem sheet.

In weeks 1 and 2, the course continues the same as last term. Demonstrator sessions are scheduled according to college, to support you in completing the final problem sheet.

After that, there are drop-in demonstrator sessions, open to students of any college:

- Weeks 3–8: Monday 3pm–4pm, and Thursday 3pm–4pm
- Week 5: Wednesday 3pm–5pm, and Friday 3pm–5pm.
- Week 8: Friday 3pm–5pm.

All drop-in sessions are in C1, except for W7 Thursday in C2, W8 Thursday in TCC.

Section 2

# General advice on projects

The projects are similar to extended problem sheets.

The projects are similar to extended problem sheets.

Write your code in Python, taking care to answer each question completely.

The projects are similar to extended problem sheets.

Write your code in Python, taking care to answer each question completely.

Use `publish` to generate a .html of the code and its output.

The projects are similar to extended problem sheets.

Write your code in Python, taking care to answer each question completely.

Use `publish` to generate a .html of the code and its output.

Submit both your code (.py) and the published output (.html), gathered into exactly one .zip or .tar.gz file.

The projects are similar to extended problem sheets.

Write your code in Python, taking care to answer each question completely.

Use `publish` to generate a .html of the code and its output.

Submit both your code (.py) and the published output (.html), gathered into exactly one .zip or .tar.gz file.

Examiners may wish to run your code to e.g. test if a function is implemented correctly.

New in HT: you can now do

(terminal) `python publish.py myscript.py`

to generate `myscript.html`.

New in HT: you can now do

(terminal)
```
python publish.py myscript.py
```

to generate `myscript.html`.

This is more efficient than using

(python)
```
from publish import publish; publish()
```

What makes a good submission?

What makes a good submission?

Projects are marked for mathematics, computing, and clarity of presentation.

What makes a good submission?

Projects are marked for mathematics, computing, and clarity of presentation.

Markers are looking for:

- clear and well-written code;
- computational evidence that each function is correct;
- clear and comprehensible plots (e.g. axis labels, titles, legends);
- mathematical discussion that indicates an understanding of the algorithms and observed results.

We strongly recommend you complete and upload them in good time.

We strongly recommend you complete and upload them in good time.

You may have unforeseen problems with

- getting the code to work correctly;
- hardware issues, e.g. your computer failing;
- network problems, when trying to upload.

We strongly recommend you complete and upload them in good time.

You may have unforeseen problems with

- getting the code to work correctly;
- hardware issues, e.g. your computer failing;
- network problems, when trying to upload.

Completing the projects in good time also gives you the opportunity to proofread and edit your work before submission.

We strongly recommend you complete and upload them in good time.

You may have unforeseen problems with

- getting the code to work correctly;
- hardware issues, e.g. your computer failing;
- network problems, when trying to upload.

Completing the projects in good time also gives you the opportunity to proofread and edit your work before submission.

Keep good backups of all your work.

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;
- using someone else's program as a model;

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;
- using someone else's program as a model;
- agreeing a detailed program plan with others;

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;
- using someone else's program as a model;
- agreeing a detailed program plan with others;
- using generative AI (ChatGPT etc) to produce any part of your code;

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;
- using someone else's program as a model;
- agreeing a detailed program plan with others;
- using generative AI (ChatGPT etc) to produce any part of your code;
- posting questions on the internet, such as on StackExchange;

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;
- using someone else's program as a model;
- agreeing a detailed program plan with others;
- using generative AI (ChatGPT etc) to produce any part of your code;
- posting questions on the internet, such as on StackExchange;
- sharing your work with other students (actively or passively, e.g. by uploading your work somewhere available to the public).

**All projects must be your own unaided work. You will be asked to make a declaration to this effect when you submit them.**

Examples of unacceptable conduct include:

- copying any part of anyone else's program;
- using someone else's program as a model;
- agreeing a detailed program plan with others;
- using generative AI (ChatGPT etc) to produce any part of your code;
- posting questions on the internet, such as on StackExchange;
- sharing your work with other students (actively or passively, e.g. by uploading your work somewhere available to the public).

The University's plagiarism policy applies in full. Potential penalties for plagiarism range from deduction of marks to expulsion.

Section 3

# Project A: primality testing

In 1801, in his magnum opus *Disquisitiones Arithmeticae*, Gauss wrote

> *The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length. ... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.*

Carl Friedrich Gauss, 1777–1855

In 1801, in his magnum opus *Disquisitiones Arithmeticae*, Gauss wrote

> *The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length. ... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.*

Carl Friedrich Gauss, 1777–1855

At Gauss' suggestion, in this project you will explore algorithms for primality testing.

The first step in the Rivest–Shamir–Adleman (RSA) cryptosystem, and many others, is to choose two large primes $p$ and $q$ (secret), and compute $n = pq$ (public).

Ron Rivest, 1947–

Adi Shamir, 1952–

Leonard Adleman, 1945–

The first step in the Rivest–Shamir–Adleman (RSA) cryptosystem, and many others, is to choose two large primes $p$ and $q$ (secret), and compute $n = pq$ (public).

To choose $p$ and $q$, the standard method is to keep choosing random integers until a primality test passes.

Ron Rivest, 1947–

Adi Shamir, 1952–

Leonard Adleman, 1945–

In 1202, in the *Liber Abaci*, Leonardo of Pisa (Fibonacci) gave the first algorithm for determining whether a number $n \in \mathbb{N}$ is prime, trial division.

> *If it is even, then he recognises its composition. However if odd, then it will be composite or prime. ... Always he goes on dividing in order by prime numbers until he will find a prime number by which he can divide, and thence he will come to the square root; if he will be able to divide by none of them, then one will judge the number to be prime.*

Leonardo of Pisa, c. 1170–1250

In 1202, in the *Liber Abaci*, Leonardo of Pisa (Fibonacci) gave the first algorithm for determining whether a number $n \in \mathbb{N}$ is prime, trial division.

> *If it is even, then he recognises its composition. However if odd, then it will be composite or prime. ... Always he goes on dividing in order by prime numbers until he will find a prime number by which he can divide, and thence he will come to the square root; if he will be able to divide by none of them, then one will judge the number to be prime.*

Leonardo of Pisa, c. 1170–1250

Can we do better than trial division?

Yes, we can do better than trial division.

Pierre de Fermat, 1607–1665

Gary Miller, ?–

Michael Rabin, 1931–

Yes, we can do better than trial division.

In this project you will implement and investigate the Fermat and primality Miller–Rabin tests for primality.

Pierre de Fermat, 1607–1665

Gary Miller, ?–

Michael Rabin, 1931–

Yes, we can do better than trial division.

In this project you will implement and investigate the Fermat and primality Miller–Rabin tests for primality.

The current state of the art in primality testing is to combine the Miller–Rabin test with another test we won't implement, called the Lucas probable prime test.

Pierre de Fermat, 1607–1665

Gary Miller, ?–

Michael Rabin, 1931–

Yes, we can do better than trial division.

In this project you will implement and investigate the Fermat and primality Miller–Rabin tests for primality.

The current state of the art in primality testing is to combine the Miller–Rabin test with another test we won't implement, called the Lucas probable prime test.

This combination is known as the Baillie–Pomerance–Selfridge–Wagstaff test, and is the algorithm used in most practical mathematical software.

Pierre de Fermat, 1607–1665

Gary Miller, ?–

Michael Rabin, 1931–

Yes, we can do better than trial division.

In this project you will implement and investigate the Fermat and primality Miller–Rabin tests for primality.

The current state of the art in primality testing is to combine the Miller–Rabin test with another test we won't implement, called the Lucas probable prime test.

This combination is known as the Baillie–Pomerance–Selfridge–Wagstaff test, and is the algorithm used in most practical mathematical software.

No composite number is known that falsely passes this test. The construction of such a composite number, or a proof that no such number exists, would solve a major open question in computational number theory.

Pierre de Fermat, 1607–1665

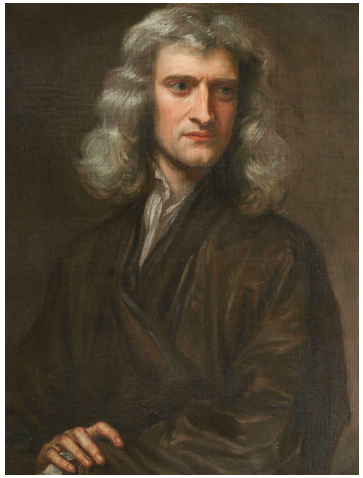
Gary Miller, ?–

Michael Rabin, 1931–

Section 4

# Project B: the Kepler problem

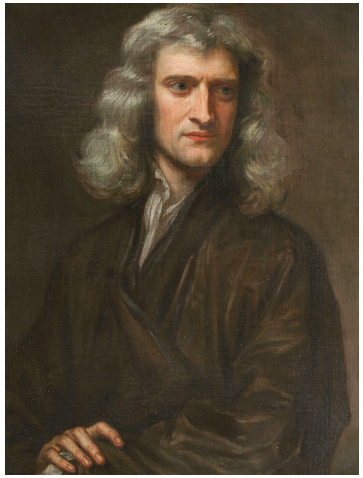
The glorious triumph of Newton's twin discoveries of calculus and Newtonian mechanics was that it allowed us to make *physical predictions* by *solving differential equations*.



Isaac Newton, 1643–1727

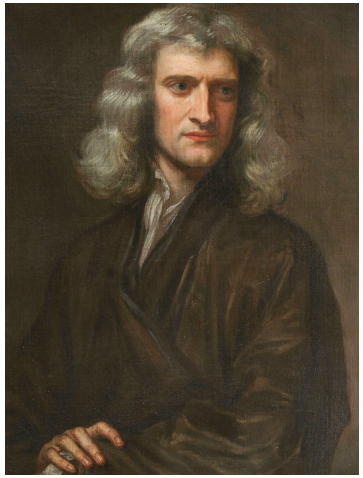
The glorious triumph of Newton's twin discoveries of calculus and Newtonian mechanics was that it allowed us to make *physical predictions* by *solving differential equations*.

Newton's second law provides an initial value problem for a second-order differential equation that describes the motion of a given system for all future time.



Isaac Newton, 1643–1727

The glorious triumph of Newton's twin discoveries of calculus and Newtonian mechanics was that it allowed us to make *physical predictions* by *solving differential equations*.



Isaac Newton, 1643–1727

Newton's second law provides an initial value problem for a second-order differential equation that describes the motion of a given system for all future time.

This tradition continues to this day; *all* of our physical theories are encoded in differential equations:

| Theory | Differential equation |
|---|---|
| Classical mechanics | Hamilton equations |
| Electromagnetism | Maxwell equations |
| Fluid mechanics | Navier–Stokes equations |
| Quantum mechanics | Schrödinger equation |
| General relativity | Einstein field equations |

However, *almost all differential equations* cannot be solved by hand!

Leonhard Euler, 1707–1783

However, *almost all differential equations* cannot be solved by hand!

If you want to make physical predictions, you must either extract partial information from the equations (e.g. conservation laws, *a priori* estimates, asymptotic statements) ...

Leonhard Euler, 1707–1783

However, *almost all differential equations* cannot be solved by hand!

If you want to make physical predictions, you must either extract partial information from the equations (e.g. conservation laws, *a priori* estimates, asymptotic statements) ...

... or solve the equations numerically. The numerical solution of differential equations is one of the secret technologies underpinning industrial civilisation; it influences almost everything, and almost no member of the public has heard of it.

Leonhard Euler, 1707–1783

However, *almost all differential equations* cannot be solved by hand!

If you want to make physical predictions, you must either extract partial information from the equations (e.g. conservation laws, *a priori* estimates, asymptotic statements) ...

... or solve the equations numerically. The numerical solution of differential equations is one of the secret technologies underpinning industrial civilisation; it influences almost everything, and almost no member of the public has heard of it.

Euler proposed the first useful scheme for the numerical solution of initial value problems in 1768, the forward Euler method, in *Institutiones Calculi Integralis*.

Leonhard Euler, 1707–1783

In this project you will investigate algorithms for the numerical solution of differential equations, in the context of the Kepler problem.

Johannes Kepler, 1571–1630

In this project you will investigate algorithms for the numerical solution of differential equations, in the context of the Kepler problem.

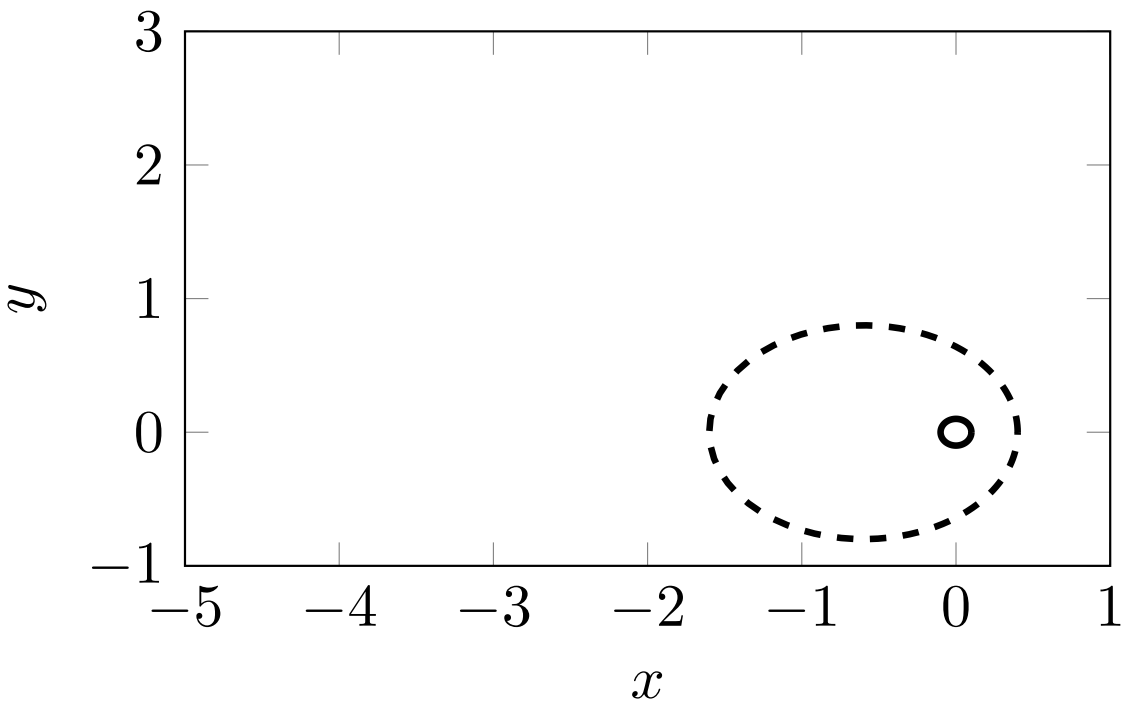
The Kepler problem describes the orbit of a single planet around its star. Here is an orbit and its forward Euler approximations for different timesteps.

Johannes Kepler, 1571–1630

In this project you will investigate algorithms for the numerical solution of differential equations, in the context of the Kepler problem.

The Kepler problem describes the orbit of a single planet around its star. Here is an orbit and its forward Euler approximations for different timesteps.

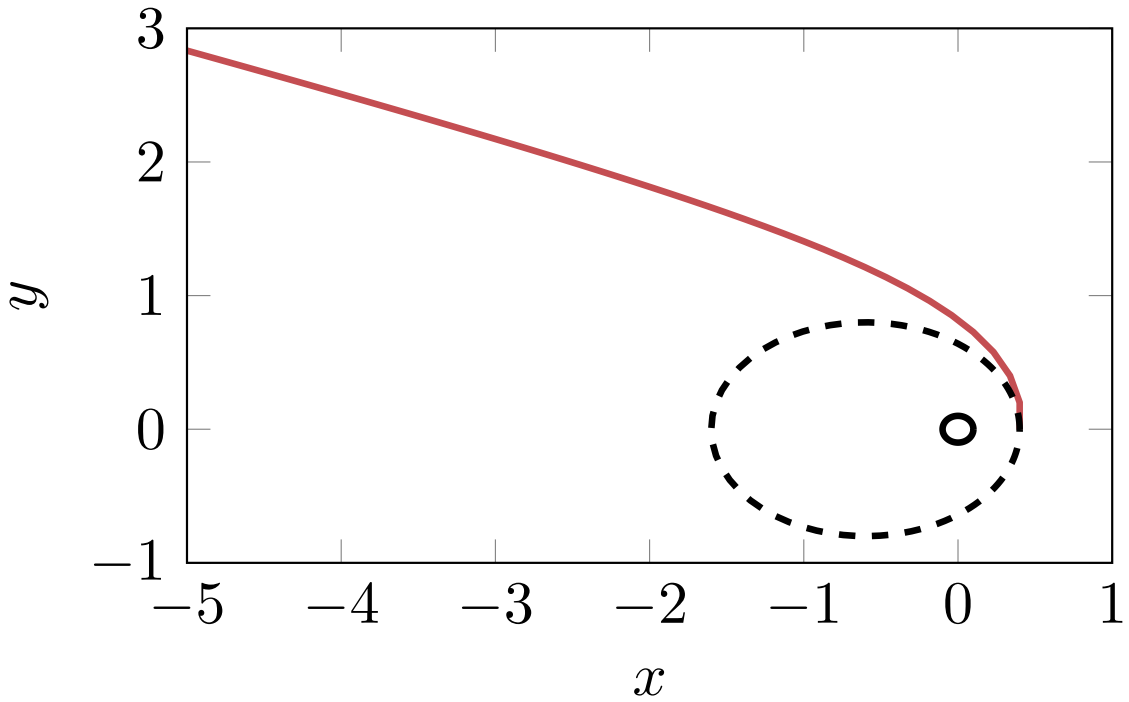
Johannes Kepler, 1571–1630



In this project you will investigate algorithms for the numerical solution of differential equations, in the context of the Kepler problem.

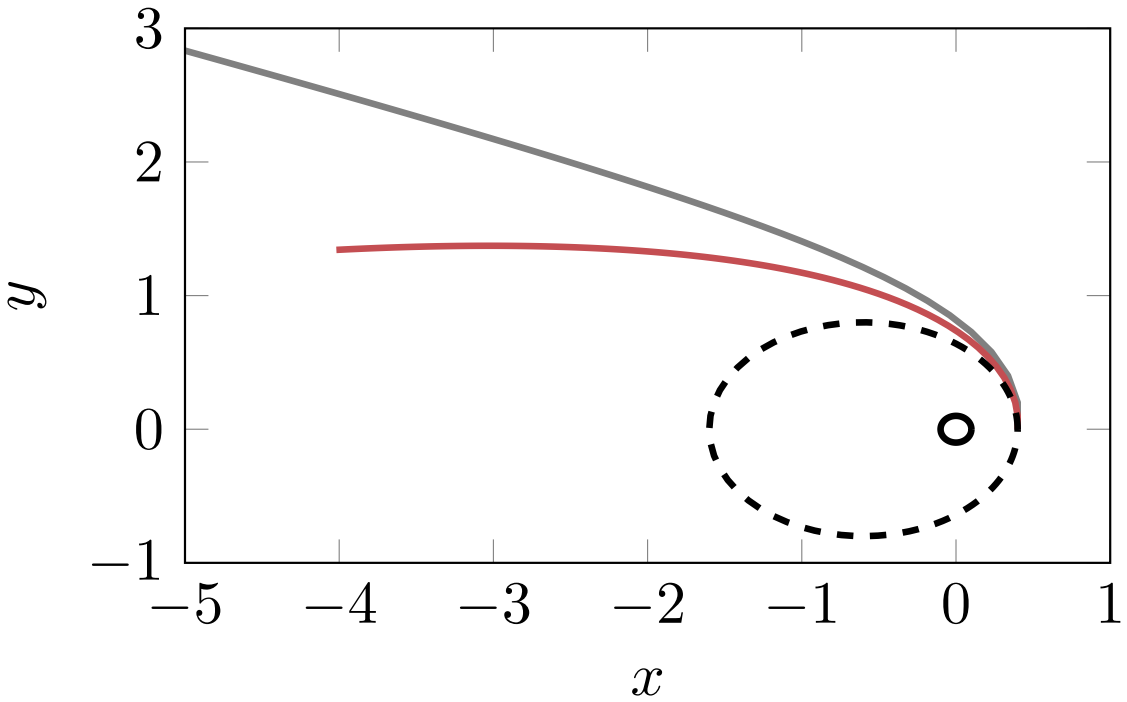
The Kepler problem describes the orbit of a single planet around its star. Here is an orbit and its forward Euler approximations for different timesteps.

Johannes Kepler, 1571–1630



In this project you will investigate algorithms for the numerical solution of differential equations, in the context of the Kepler problem.

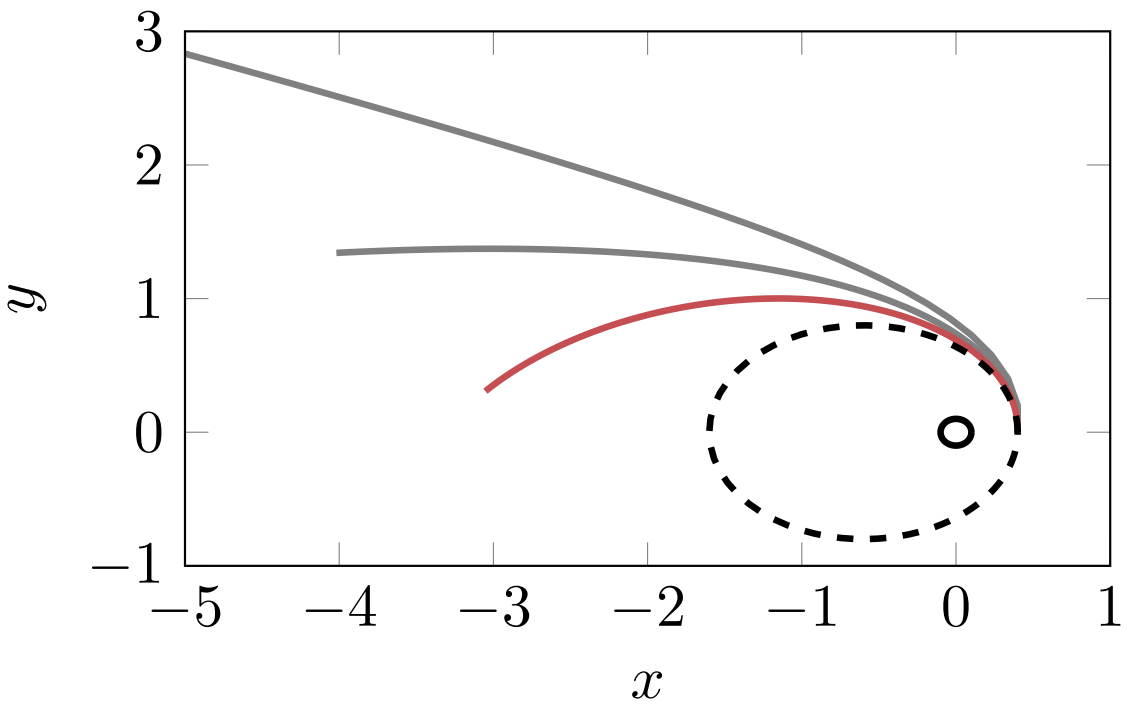
The Kepler problem describes the orbit of a single planet around its star. Here is an orbit and its forward Euler approximations for different timesteps.




Johannes Kepler, 1571–1630

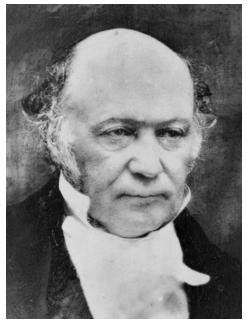
In this project you will investigate algorithms for the numerical solution of differential equations, in the context of the Kepler problem.

The Kepler problem describes the orbit of a single planet around its star. Here is an orbit and its forward Euler approximations for different timesteps.



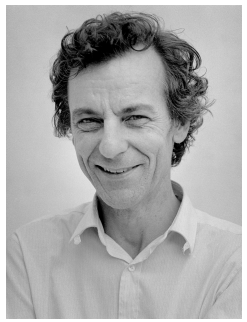

Johannes Kepler, 1571–1630

The reason for this surprising failure is that *the equations of motion have a deep geometric structure* that the forward Euler method neglects.



William Rowan Hamilton, 1805–1865

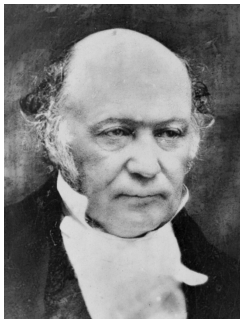
Emmy Noether, 1882–1935



Loup Verlet, 1931–2019

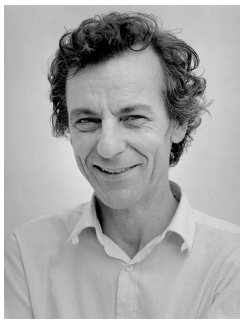
The reason for this surprising failure is that *the equations of motion have a deep geometric structure* that the forward Euler method neglects.

This geometric structure is exposed in the Hamiltonian formulation of classical mechanics, and has a fundamental connection to symmetries, via Noether's theorem.



William Rowan Hamilton, 1805–1865

Emmy Noether, 1882–1935

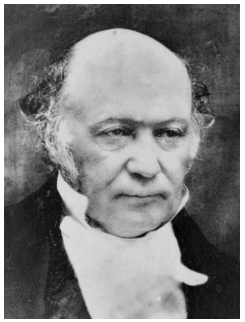


Loup Verlet, 1931–2019

The reason for this surprising failure is that *the equations of motion have a deep geometric structure* that the forward Euler method neglects.

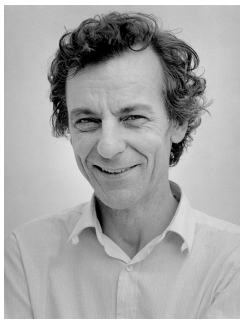
This geometric structure is exposed in the Hamiltonian formulation of classical mechanics, and has a fundamental connection to symmetries, via Noether's theorem.

In this project you will study numerical methods that honour and reflect this geometric structure.



William Rowan Hamilton, 1805–1865

Emmy Noether, 1882–1935



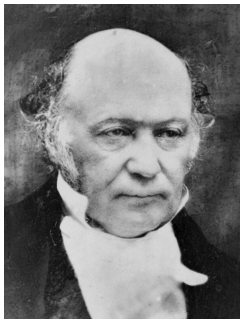
Loup Verlet, 1931–2019

The reason for this surprising failure is that *the equations of motion have a deep geometric structure* that the forward Euler method neglects.

This geometric structure is exposed in the Hamiltonian formulation of classical mechanics, and has a fundamental connection to symmetries, via Noether's theorem.

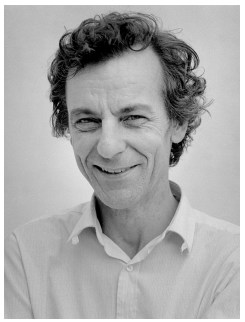
In this project you will study numerical methods that honour and reflect this geometric structure.

Such methods are essential for accurate long-term simulations of the solar system or understanding the molecular dynamics of complex materials.



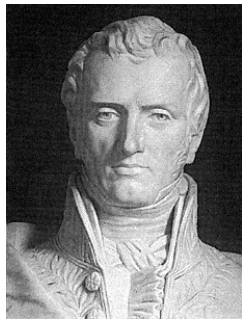
William Rowan Hamilton, 1805–1865

Emmy Noether, 1882–1935



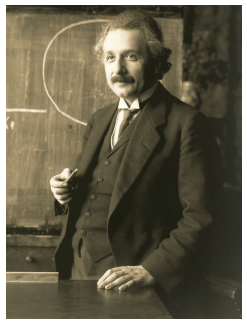
Loup Verlet, 1931–2019

The topic of identifying and preserving hidden algebraic, topological, or geometric structures when solving differential equations is a current research frontier.



Claude–Louis Navier, 1785–1836

George Gabriel Stokes, 1819–1903

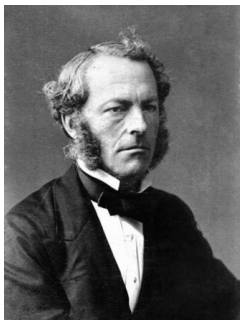


Albert Einstein, 1879–1955

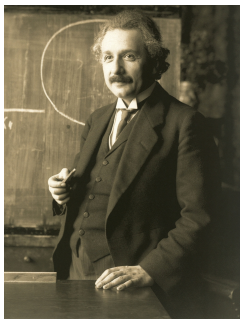
The topic of identifying and preserving hidden algebraic, topological, or geometric structures when solving differential equations is a current research frontier.

For example, there remain major open questions about the numerical solution of the Navier–Stokes equations of fluid mechanics, or the Einstein field equations of general relativity.

Claude–Louis Navier, 1785–1836



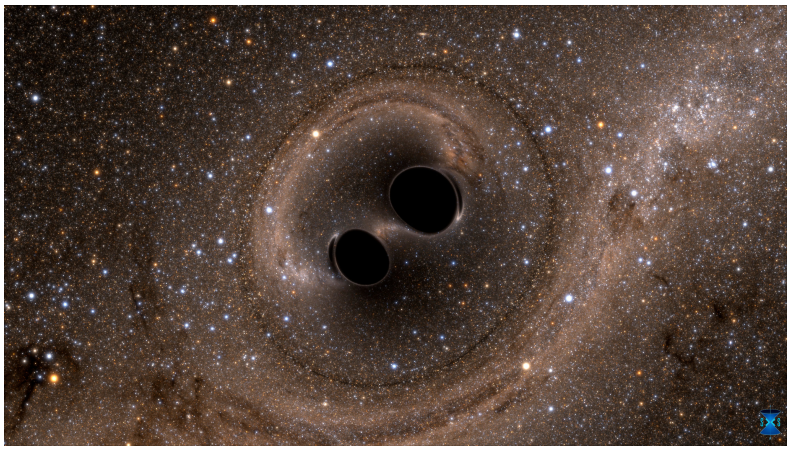
George Gabriel Stokes, 1819–1903



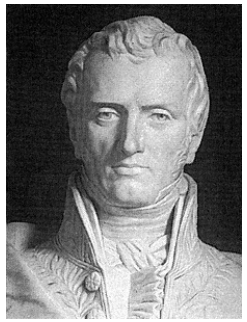
Albert Einstein, 1879–1955

The topic of identifying and preserving hidden algebraic, topological, or geometric structures when solving differential equations is a current research frontier.

For example, there remain major open questions about the numerical solution of the Navier–Stokes equations of fluid mechanics, or the Einstein field equations of general relativity.

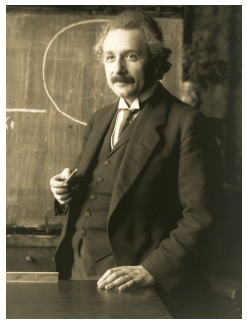


Research in this area allows us to simulate phenomena such as the merger of two black holes.



Claude–Louis Navier, 1785–1836

George Gabriel Stokes, 1819–1903

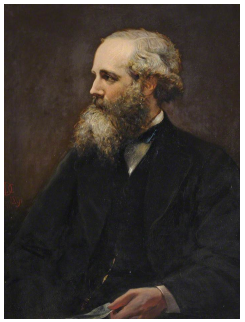


Albert Einstein, 1879–1955

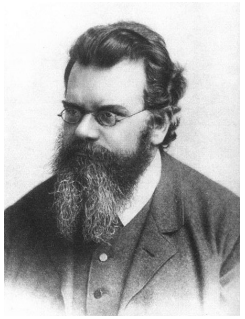
Section 5

# Project C: percolation

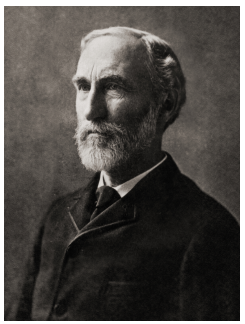
Statistical mechanics, founded by Maxwell, Boltzmann, and Gibbs in the 1800s, applies statistical and probabilistic methods to large assemblies of microscopic entities.



James Clerk Maxwell, 1831–1879



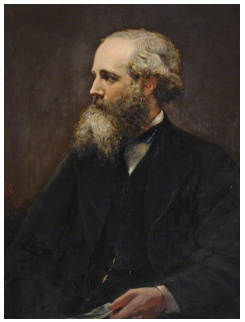
Ludwig Boltzmann, 1844–1906



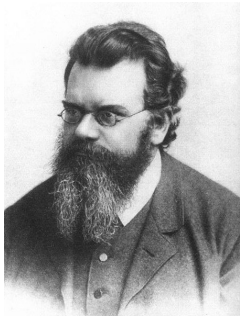
Josiah Willard Gibbs, 1839–1903

Statistical mechanics, founded by Maxwell, Boltzmann, and Gibbs in the 1800s, applies statistical and probabilistic methods to large assemblies of microscopic entities.

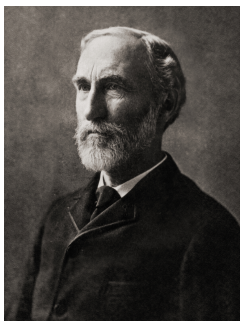
For example, the Newtonian approach to understanding a gas would be to track the velocity and position of each of the trillions of trillions of molecules in a typical cubic metre.



James Clerk Maxwell, 1831–1879



Ludwig Boltzmann, 1844–1906

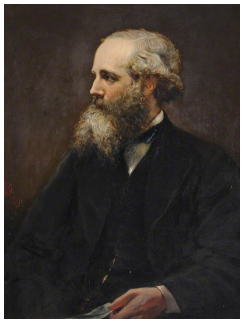


Josiah Willard Gibbs, 1839–1903

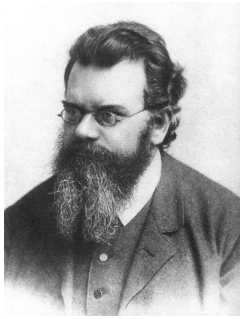
Statistical mechanics, founded by Maxwell, Boltzmann, and Gibbs in the 1800s, applies statistical and probabilistic methods to large assemblies of microscopic entities.

For example, the Newtonian approach to understanding a gas would be to track the velocity and position of each of the trillions of trillions of molecules in a typical cubic metre.

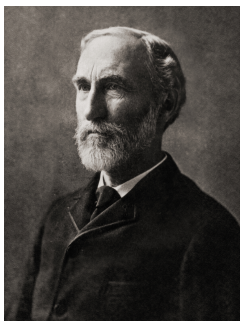
Maxwell's great insight was that this description was excessive. To understand the macroscopic properties of the gas like its pressure or temperature, you could instead merely store a *probability distribution* recording statistics about the molecules.



James Clerk Maxwell, 1831–1879



Ludwig Boltzmann, 1844–1906



Josiah Willard Gibbs, 1839–1903

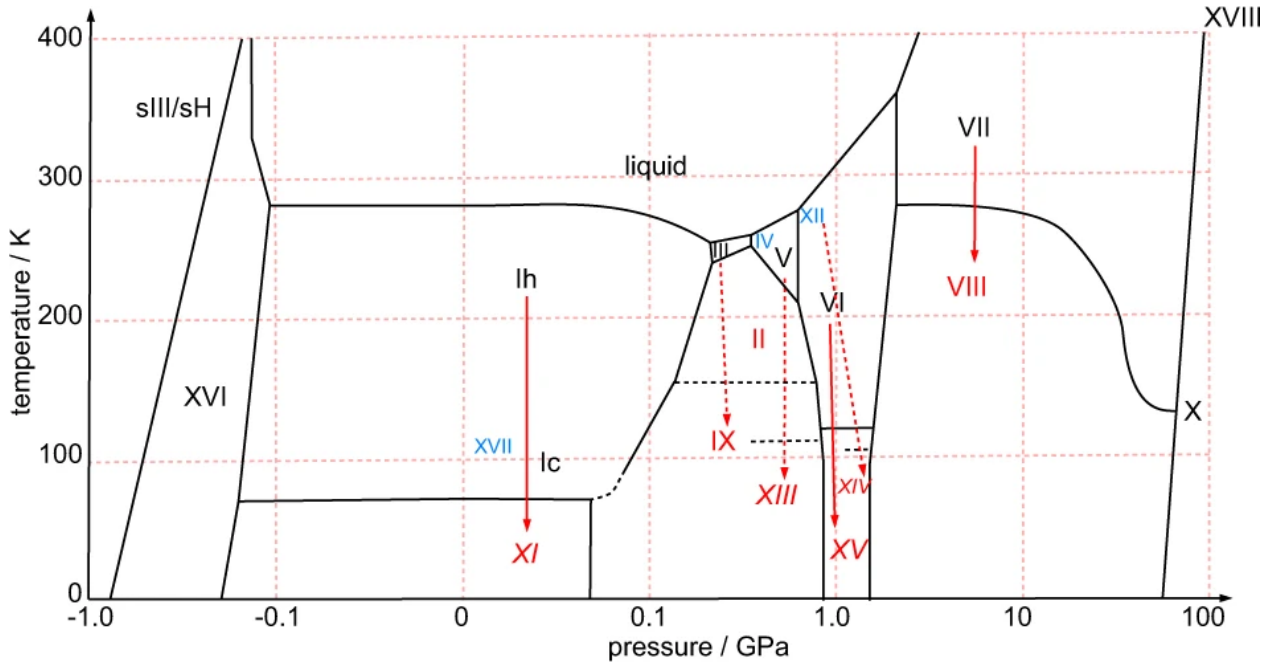
One of the major goals of statistical mechanics is to understand and predict *phase transitions*. A phase transition is an abrupt, discontinuous change in the properties of a system.

One of the major goals of statistical mechanics is to understand and predict *phase transitions*. A phase transition is an abrupt, discontinuous change in the properties of a system.

For example, if you cool a gas, at a critical temperature it will (usually) turn into a liquid, with its density and volume changing discontinuously.

One of the major goals of statistical mechanics is to understand and predict *phase transitions*. A phase transition is an abrupt, discontinuous change in the properties of a system.

For example, if you cool a gas, at a critical temperature it will (usually) turn into a liquid, with its density and volume changing discontinuously.




Phase diagram of ice, from Hansen (2021).

Several phases of $H_2O$.

One route to understanding phase transitions is to study simpler mathematical systems that exhibit them.

John Hammersley, 1920–2004

Hugo Duminil-Copin, 1985–

One route to understanding phase transitions is to study simpler mathematical systems that exhibit them.

A prominent class of such systems is studied in *percolation theory*. Percolation theory describes the properties of a graph as nodes or edges are added.

John Hammersley, 1920–2004

Hugo Duminil-Copin, 1985–

One route to understanding phase transitions is to study simpler mathematical systems that exhibit them.

A prominent class of such systems is studied in *percolation theory*. Percolation theory describes the properties of a graph as nodes or edges are added.

Percolation theory was founded in a seminal 1957 article by Broadbent & Hammersley. Hammersley was later a professor at Trinity College, Oxford.

John Hammersley, 1920–2004

Hugo Duminil-Copin, 1985–

One route to understanding phase transitions is to study simpler mathematical systems that exhibit them.

A prominent class of such systems is studied in *percolation theory*. Percolation theory describes the properties of a graph as nodes or edges are added.

Percolation theory was founded in a seminal 1957 article by Broadbent & Hammersley. Hammersley was later a professor at Trinity College, Oxford.

Percolation theory is active to this day. Hugo Duminil-Copin won a Fields Medal in 2022 for his work in this area.

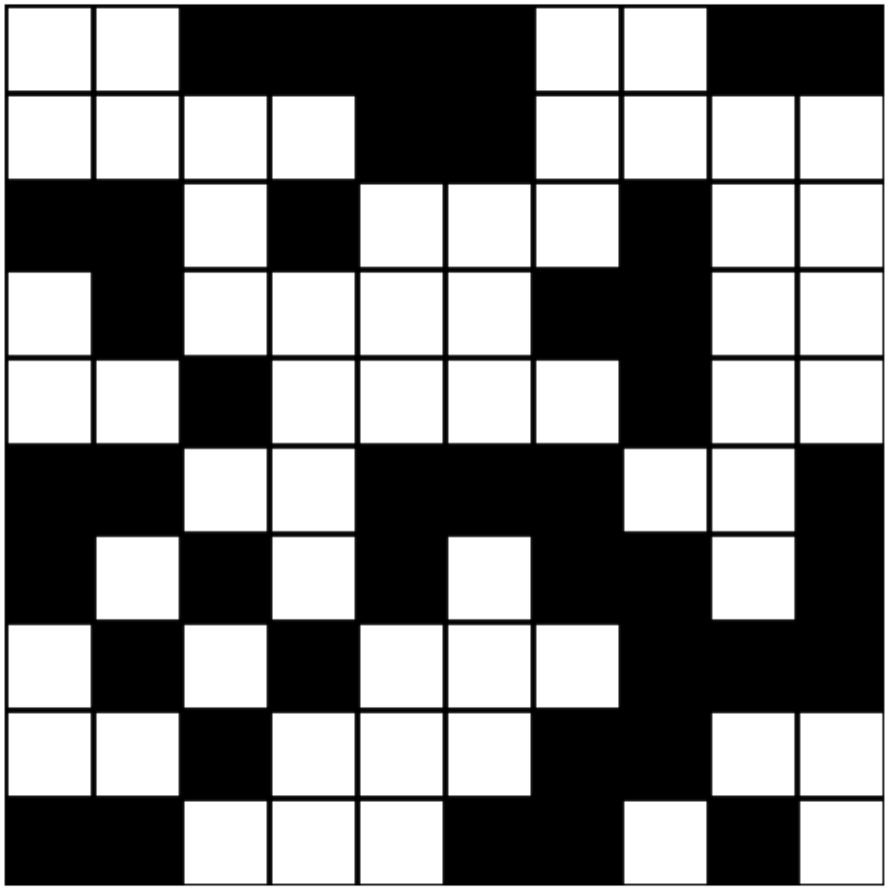
John Hammersley, 1920–2004

Hugo Duminil-Copin, 1985–

We consider the simplest unsolved case, *Bernoulli site percolation*.

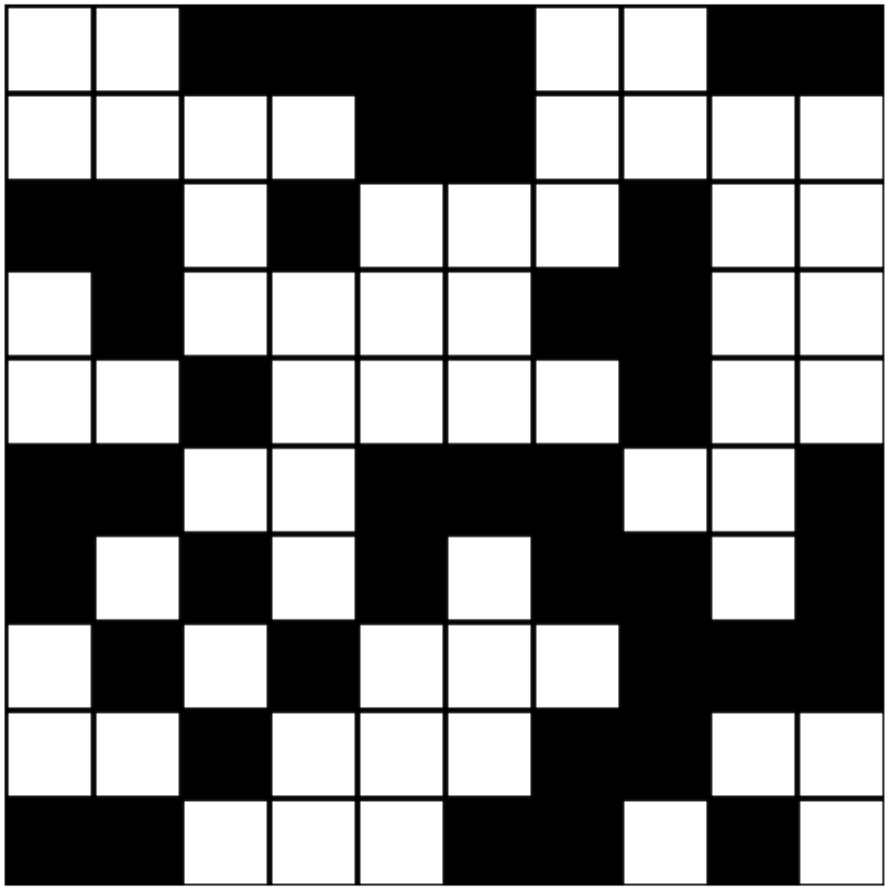
We consider the simplest unsolved case, *Bernoulli site percolation*.

Each site on an $n \times n$ grid is *open* or *closed*, open with probability $p$.



We consider the simplest unsolved case, *Bernoulli site percolation*.

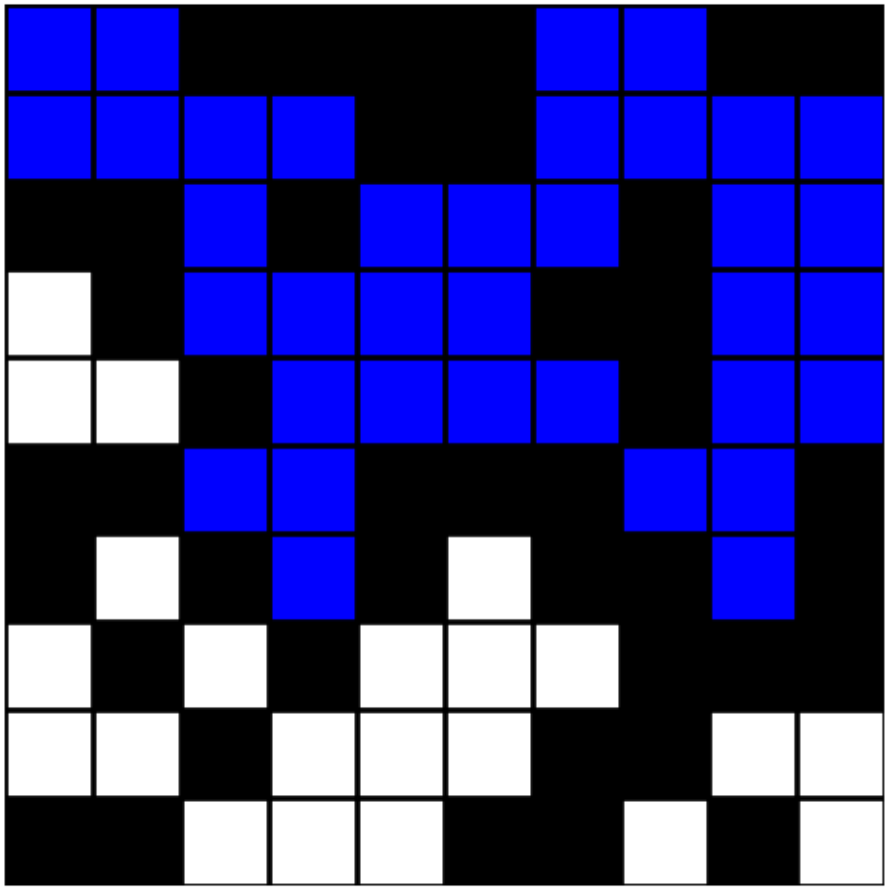
Each site on an $n \times n$ grid is *open* or *closed*, open with probability $p$.



A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

We consider the simplest unsolved case, *Bernoulli site percolation*.

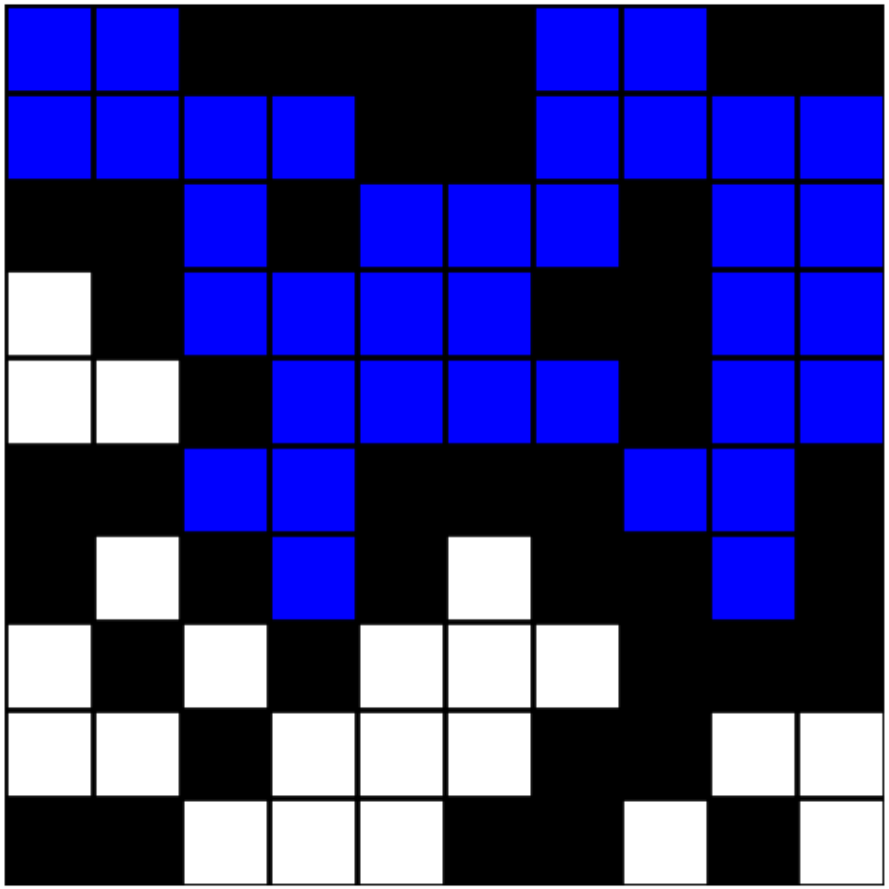
Each site on an $n \times n$ grid is *open* or *closed*, open with probability $p$.



A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

We consider the simplest unsolved case, *Bernoulli site percolation*.

Each site on an $n \times n$ grid is *open* or *closed*, open with probability $p$.

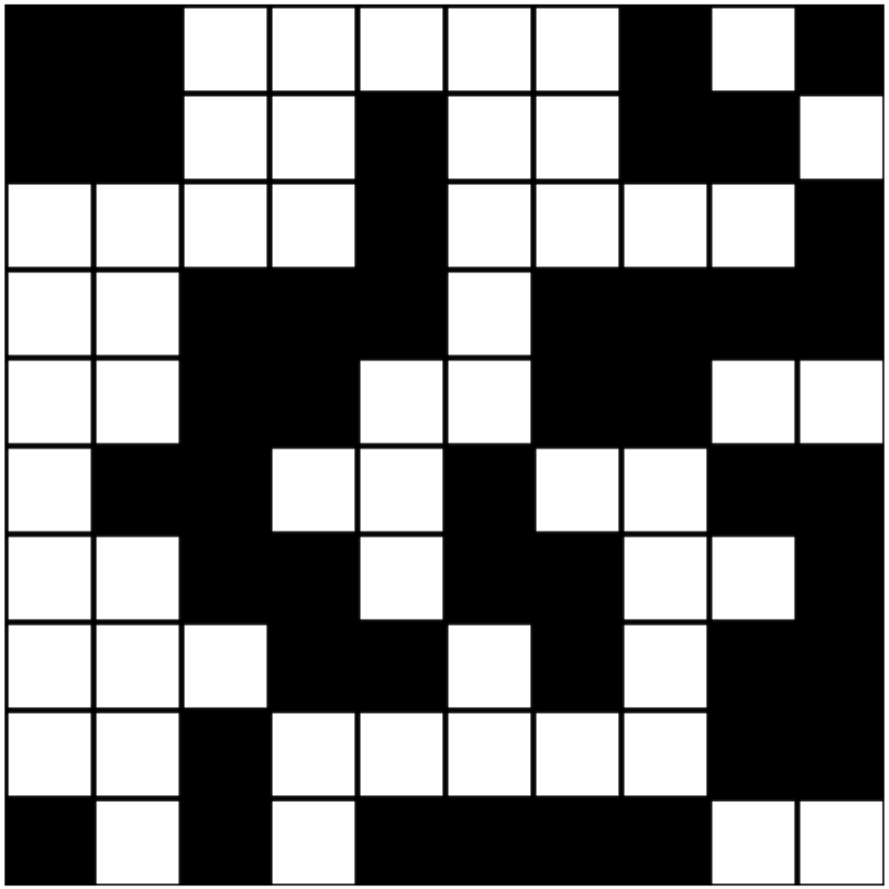


A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

The system *percolates* if there is a full site on the bottom row.

We consider the simplest unsolved case, *Bernoulli site percolation*.

Each site on an $n \times n$ grid is *open* or *closed*, open with probability $p$.

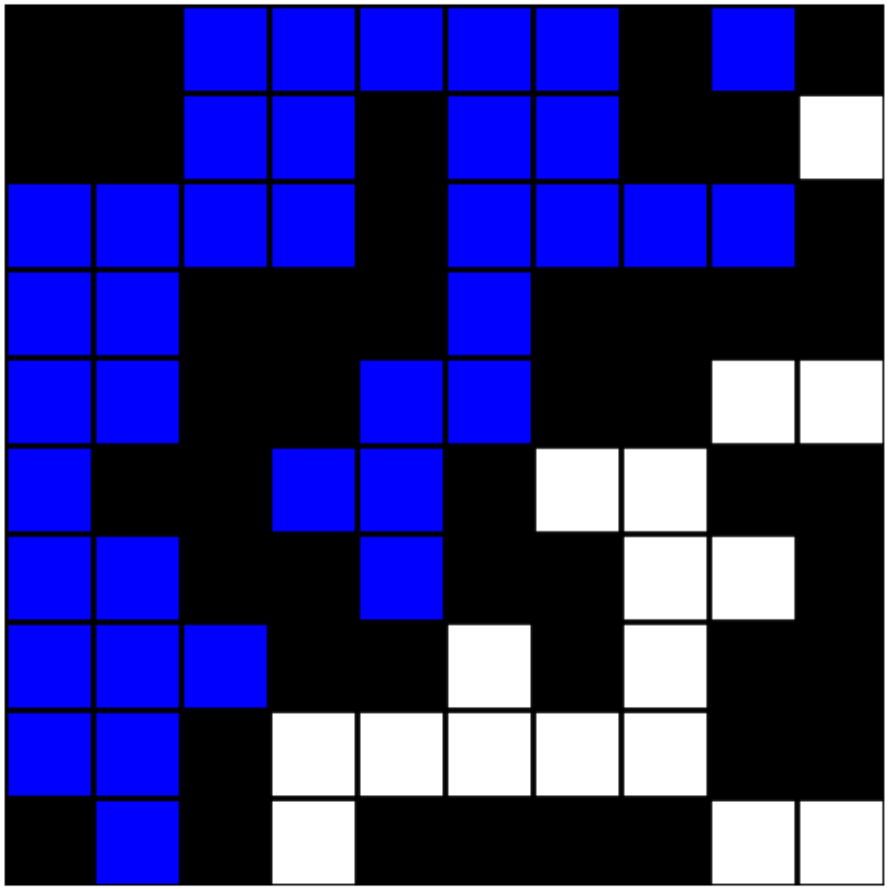


A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

The system *percolates* if there is a full site on the bottom row.

We consider the simplest unsolved case, *Bernoulli site percolation*.

Each site on an $n \times n$ grid is *open* or *closed*, open with probability $p$.



A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

The system *percolates* if there is a full site on the bottom row.

Each site is open with *vacancy probability* $p$.

Each site is open with *vacancy probability* $p$.

For a given $p$, what is the *probability of percolation* $C(p)$?

Each site is open with *vacancy probability* $p$.

For a given $p$, what is the *probability of percolation* $C(p)$?

This exhibits a phase transition!

Each site is open with *vacancy probability* $p$.

For a given $p$, what is the *probability of percolation* $C(p)$?

This exhibits a phase transition!

$C(p)$ jumps very rapidly from $0$ to $1$ around a critical $p = p_c$.

Each site is open with *vacancy probability* $p$.

For a given $p$, what is the *probability of percolation* $C(p)$?

This exhibits a phase transition!

$C(p)$ jumps very rapidly from $0$ to $1$ around a critical $p = p_c$.

Despite a great deal of effort, no analytical formula is known for the critical probability $p_c$.

To estimate $C(p)$, we use *Monte Carlo methods*.

Stanisław Ulam, 1909–1984

John von Neumann, 1903–1957

To estimate $C(p)$, we use *Monte Carlo methods*.

Essentially, for fixed $p$ we will draw many sample grids, and count the fraction that percolate.

Stanisław Ulam, 1909–1984

John von Neumann, 1903–1957

To estimate $C(p)$, we use *Monte Carlo methods*.

Essentially, for fixed $p$ we will draw many sample grids, and count the fraction that percolate.

Stanisław Ulam, 1909–1984

John von Neumann, 1903–1957

To estimate $C(p)$, we use *Monte Carlo methods*.

Essentially, for fixed $p$ we will draw many sample grids, and count the fraction that percolate.

> *The first thoughts and attempts I made to practice were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays.*

Stanisław Ulam, 1909–1984

John von Neumann, 1903–1957

Section 6

# Summary

This year's projects are

- primality testing;
- the Kepler problem;
- percolation;

This year's projects are

- primality testing;
- the Kepler problem;
- percolation;

I hope that you find the projects interesting, and that you have fun!