Computational Mathematics
2025 Projects

Patrick E. Farrell

University of Oxford

Overview

General advice on projects

2025A: Convex hulls

2025B: Orbital elements

2025C: Pension planning

Summary

Previous projects: comments and guidance

2024A: Primality testing

2024C: Percolation

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 half 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 half 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^{st} project: 12 noon on Monday of week 2 TT25

The deadlines for these projects are

- 1^{st} project: 12 noon on Monday of week 2 TT25
- 2^{nd} project: 12 noon on Monday of week 5 TT25

The deadlines for these projects are

- 1^{st} project: 12 noon on Monday of week 2 TT25
- 2^{nd} project: 12 noon on Monday of week 5 TT25

Please submit online via Inspera before these deadlines.

The deadlines for these projects are

- 1^{st} project: 12 noon on Monday of week 2 TT25
- 2^{nd} project: 12 noon on Monday of week 5 TT25

Please submit online via Inspera before these deadlines.

The University imposes mark penalties for late submission.

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.

Each project has some mathematical questions not answered by coding. You can write your answer in words, or use basic latex: adding

```
# \begin{equation*}
# \sin{x} = 1.
# \end{equation*}
```

to your code produces published output that looks like

$$\sin x = 1.$$

The projects are similar to extended problem sheets.

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

Each project has some mathematical questions not answered by coding. You can write your answer in words, or use basic latex: adding

```
# \begin{equation*}
# \sin{x} = 1.
# \end{equation*}
```

to your code produces published output that looks like

$$\sin x = 1.$$

You can write latex inline using dollars: \$\sin{x}\$ produces $\sin x$.

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

Use `publish.py` 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.

Use `publish.py` 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.

I think everyone got `publish` working in the demonstration sessions, but in case it doesn't work, don't stress. Just submit the .py file.

Use `publish.py` 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.

I think everyone got `publish` working in the demonstration sessions, but in case it doesn't work, don't stress. Just submit the .py file.

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

What makes a good submission?

What makes a good submission?

Projects are marked for mathematical insight (40%), programming skill (40%), and clarity of presentation (20%).

What makes a good submission?

Projects are marked for mathematical insight (40%), programming skill (40%), and clarity of presentation (20%).

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 understanding of and insight into 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 a substantial 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 a substantial part of anyone else's program;
- using a substantial part of another 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 a substantial part of anyone else's program;
- using a substantial part of another 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 a substantial part of anyone else's program;
- using a substantial part of another 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 a substantial part of anyone else's program;
- using a substantial part of another 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 a substantial part of anyone else's program;
- using a substantial part of another 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 a substantial part of anyone else's program;
- using a substantial part of another 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.

Acceptable (and encouraged) practices include:

- using search engines like Google;

Acceptable (and encouraged) practices include:

- using search engines like Google;
- consulting documentation, e.g. for libraries used;

Acceptable (and encouraged) practices include:

- using search engines like Google;
- consulting documentation, e.g. for libraries used;
- consulting online forums, e.g. StackExchange, for details of how to use libraries or resolve errors.

Acceptable (and encouraged) practices include:

- using search engines like Google;
- consulting documentation, e.g. for libraries used;
- consulting online forums, e.g. StackExchange, for details of how to use libraries or resolve errors.

When you do these, include the URL of the resource you have employed in code comments, as a citation.

Section 3

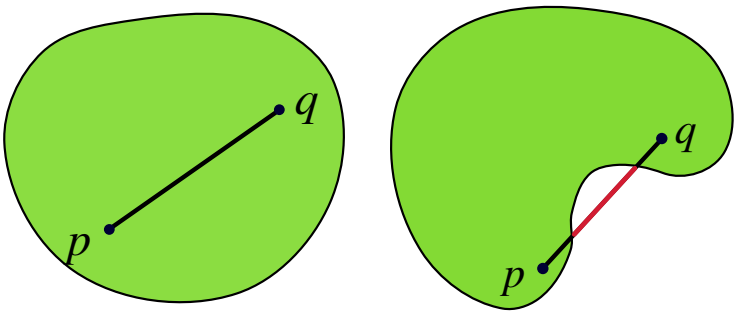
2025A: Convex hulls

Convexity is a fundamental notion in geometry.

Definition (Convex set)

Let $V = \mathbb{R}^d$, $d \in \mathbb{N}_+$. A set $C \subset V$ is *convex* if the line segment joining any two points in C is itself a subset of C:

$$\forall\, p, q \in C \,\forall\, t \in [0,1]\ tp + (1-t)q \in C.$$



Definition (Convex hull)

Given $S \subset V$, its convex hull $C = \operatorname{hull} S$ is the minimal convex set containing S.

Definition (Convex hull)

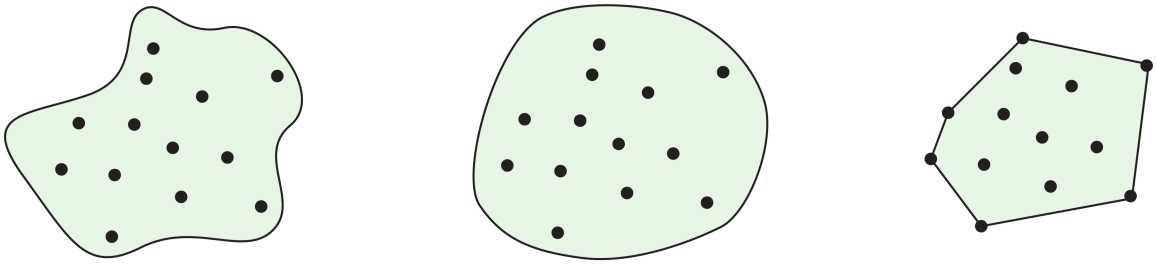
Given $S \subset V$, its convex hull $C = \operatorname{hull} S$ is the minimal convex set containing S.

Minimality means that if $S \subset C'$ and C' is convex, then $C \subseteq C'$.

Definition (Convex hull)

Given $S \subset V$, its convex hull $C = \operatorname{hull} S$ is the minimal convex set containing S.

Minimality means that if $S \subset C'$ and C' is convex, then $C \subseteq C'$.



(a) (b) (c)

$S =$ black points. (a) A nonconvex containing set. (b) A convex containing set. (c) The minimal convex containing set, the convex hull.

Given a point set S, how do we compute its convex hull?

Given a point set S, how do we compute its convex hull?

In this project we will study two algorithms for solving this problem in two dimensions:

- Graham scan;
- divide and conquer.

Given a point set S, how do we compute its convex hull?

In this project we will study two algorithms for solving this problem in two dimensions:

- Graham scan;
- divide and conquer.

The Graham scan is very simple and efficient, but only works in two dimensions. Divide and conquer extends to higher dimensions but is a little bit more involved.

Given a point set S, how do we compute its convex hull?

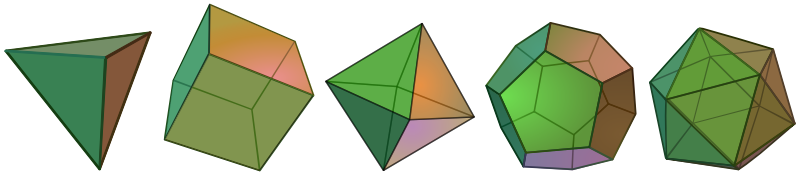
In this project we will study two algorithms for solving this problem in two dimensions:

- Graham scan;
- divide and conquer.

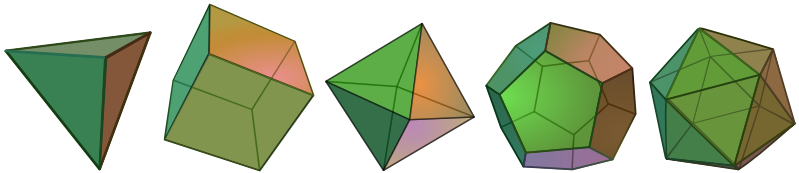
The Graham scan is very simple and efficient, but only works in two dimensions. Divide and conquer extends to higher dimensions but is a little bit more involved.

We will also investigate the complexity of these algorithms, in runtime and in storage.

Euclid's Elements culminates with a proof that there are only five regular convex three-dimensional polyhedra:



Euclid's Elements culminates with a proof that there are only five regular convex three-dimensional polyhedra:



Plato associates each of these to a constituent of the physical universe:

> *... four equilaterals form the sides of a regular solid, the tetrahedron or pyramid, which is the constituent particle of fire: eight such equilaterals are the sides of the octahedron, which is the particle of air; twenty equilaterals are the sides of the icosahedron, being the particle of water. ... six squares are the sides of a fourth regular solid called the cube, which is the particle proper to earth. A fifth regular solid still exists, namely the dodecahedron, which does not form the element of any substance; but God used it as a pattern for dividing the zodiac into its twelve signs.*

Section 4

2025B: Orbital elements

In 1609 Kepler realised that a single planet orbiting the sun follows an elliptical trajectory.

Johannes Kepler, 1571–1630

In 1609 Kepler realised that a single planet orbiting the sun follows an elliptical trajectory.

With multiple planets this is no longer true, but it is still a good approximation.

Johannes Kepler, 1571–1630

In 1609 Kepler realised that a single planet orbiting the sun follows an elliptical trajectory.

With multiple planets this is no longer true, but it is still a good approximation.

Geometers therefore paid great attention to ellipses in three dimensions.

Johannes Kepler, 1571–1630

In 1609 Kepler realised that a single planet orbiting the sun follows an elliptical trajectory.

With multiple planets this is no longer true, but it is still a good approximation.

Geometers therefore paid great attention to ellipses in three dimensions.

Ellipses in three dimensions are described by six parameters: the *orbital elements*.

Johannes Kepler, 1571–1630

In *A Synopsis of the Astronomy of Comets*, Halley realised that the comet he had observed in 1682 was the same as that seen by Kepler in 1607:

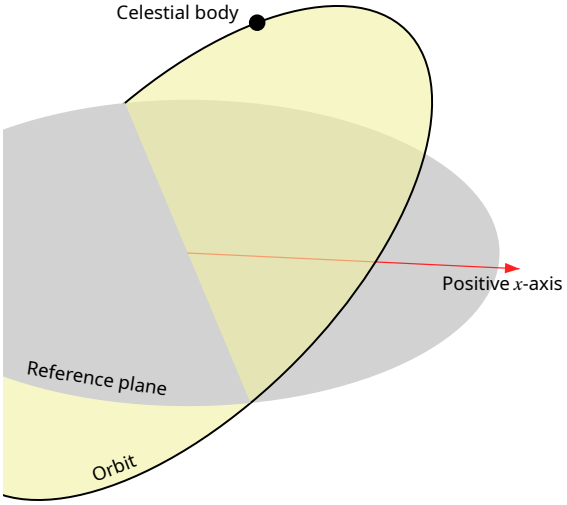
Edmund Halley FRS, 1656–1742

> *The principal Use therefore of this Table of the Elements of their Motions, and that which induced me to construct it, is, That whenever a new Comet shall appear, we may be able to know, by comparing together the Elements, whether it be any of those which has appear'd before, and consequently to determine its Period, and the Axis of its Orbit, and to foretell its Return. And, indeed, there are many Things which make me believe that the Comet which Apian observ'd in the Year 1531, was the same with that which Kepler and Longomontanus took Notice of and describ'd in the Year 1607, and which I my self have seen return, and observ'd in the Year 1682. All the Elements agree, and nothing seems to contradict this my Opinion, besides the Inequality of the Periodick Revolutions.*

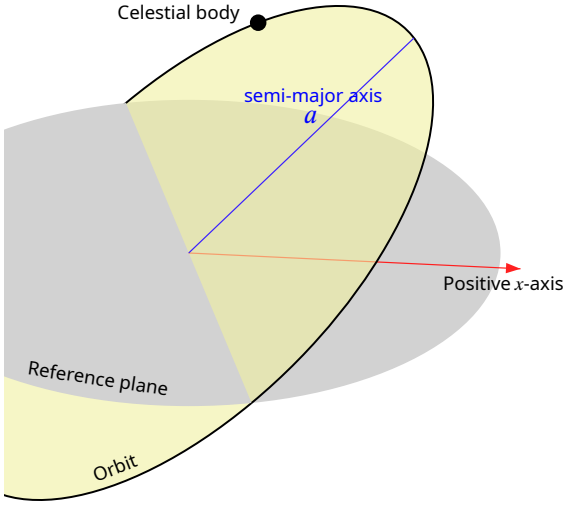
Halley's comet also appeared in 1066, just before the Norman invasion.

Halley's comet also appeared in 1066, just before the Norman invasion.

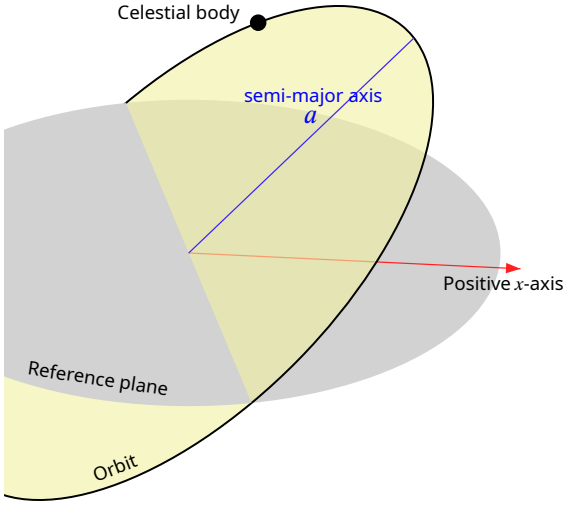
In this project we use the orbital elements of Halley's comet to 'foretell its Return'.



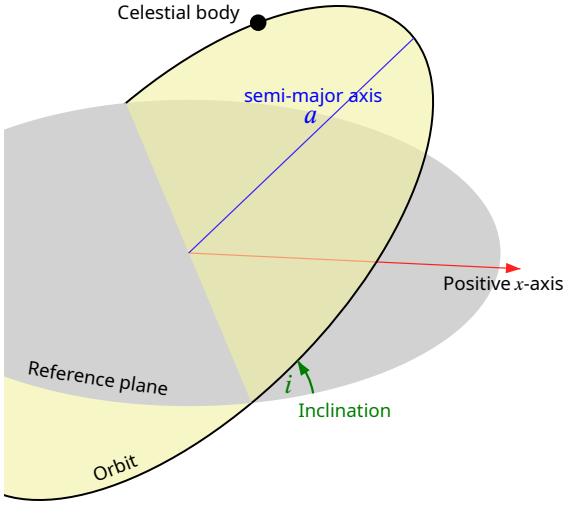
We describe ellipses in three dimensions with six parameters (a, e, i, Ω, ω, T).



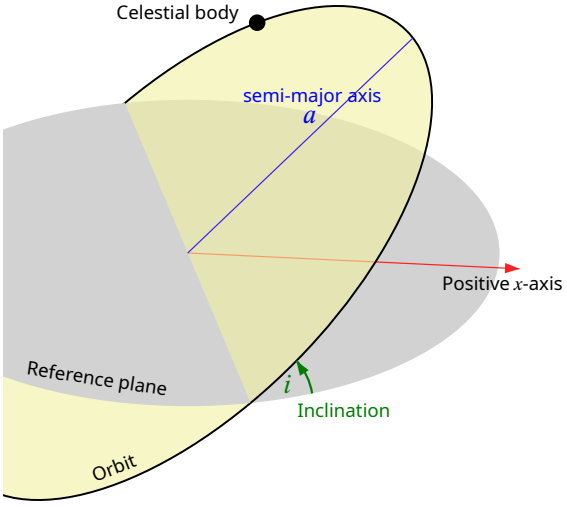
The *semi-major axis* a is half the longest diameter of the ellipse.



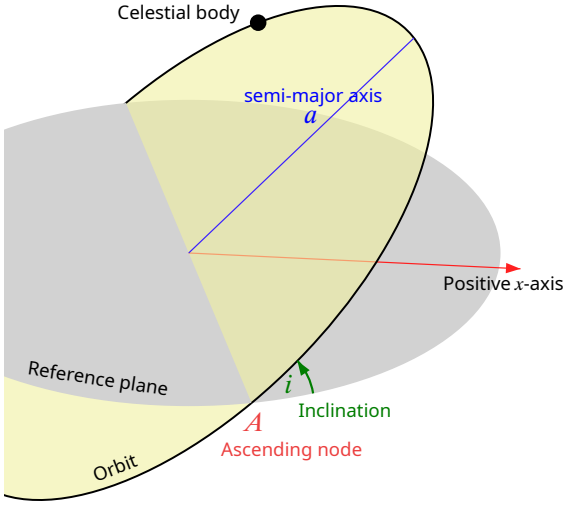
The *eccentricity* $e \in [0, 1)$ describes how circular or elliptical the orbit is.



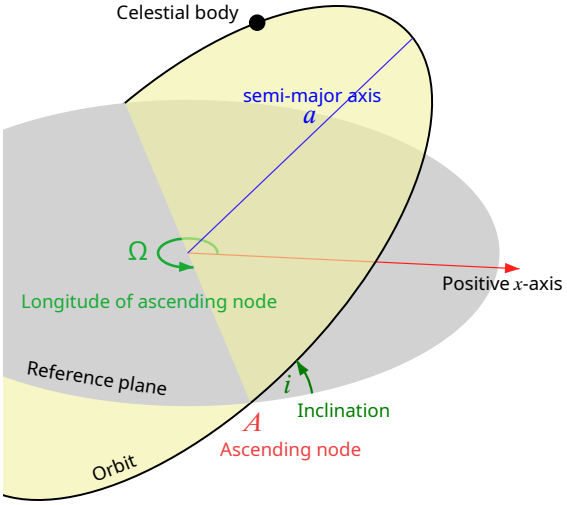
The *inclination* i measures the angle between the orbiting ellipse and the reference plane.



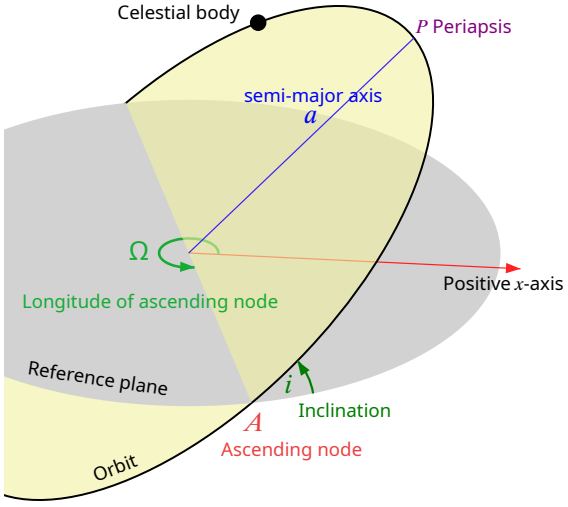
The *nodes* are the two points where the orbit intersects the reference plane.



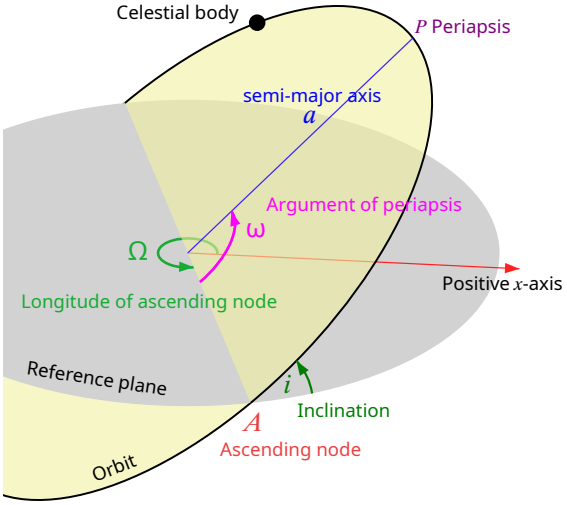
The *ascending node* is the node with $\dot{z} > 0$.



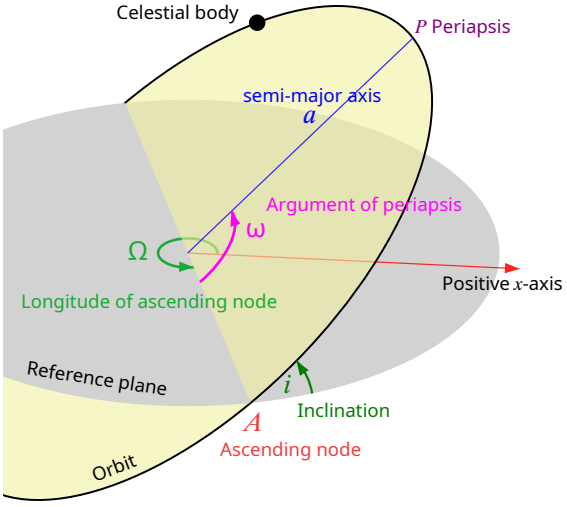
The *longitude of the ascending node* Ω measures the angle to the ascending node.



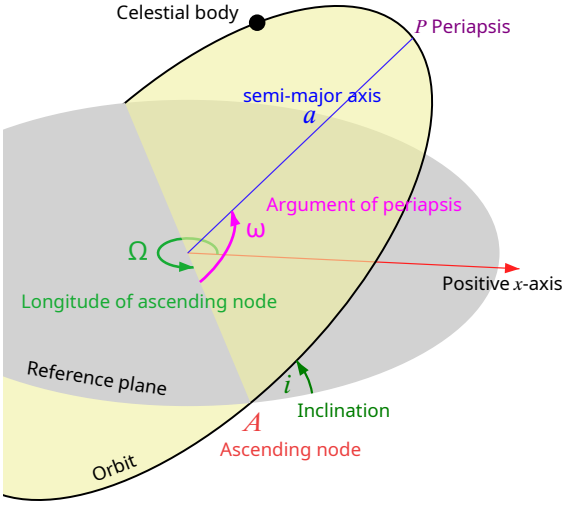
The *periapsis* is the point on the orbit closest to the sun.



The *argument of periapsis* measures the angle between the line of nodes and the periapsis.



The *period* T is the time taken by one orbit.



Finally, the *time of periapsis* τ is a time at which the body was at periapsis.

Once you have the orbital elements, you can compute the position of the body at any time.

Once you have the orbital elements, you can compute the position of the body at any time.

The key step is solving *Kepler's equation*

$$f(E) = E - e\sin E - M$$

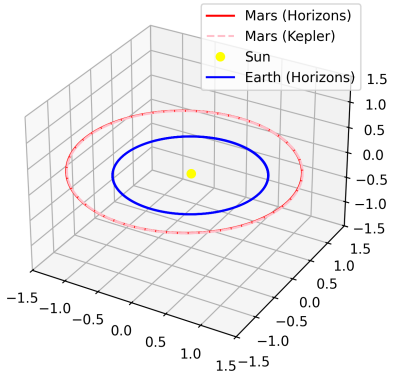
for given M. We analyse this with the Banach contraction mapping theorem.

Once you have the orbital elements, you can compute the position of the body at any time.

The key step is solving *Kepler's equation*

$$f(E) = E - e\sin E - M$$

for given M. We analyse this with the Banach contraction mapping theorem.



Section 5

2025C: Pension planning

Almost all pensions are now *defined contribution*: you and your employer invest a certain amount M each month, and you use whatever money you have at retirement age to live on.

Almost all pensions are now *defined contribution*: you and your employer invest a certain amount M each month, and you use whatever money you have at retirement age to live on.

Our investments are uncertain: we do not know in advance how much money we will have.

Almost all pensions are now *defined contribution*: you and your employer invest a certain amount M each month, and you use whatever money you have at retirement age to live on.

Our investments are uncertain: we do not know in advance how much money we will have.

How much do we need to invest in order to have a high probability of a comfortable retirement?

Here is a very simple model of a pension.

Here is a very simple model of a pension.

We decide to invest in a stock whose price per share is $S(t)$.

Here is a very simple model of a pension.

We decide to invest in a stock whose price per share is $S(t)$.

Each month we buy M worth of shares, so our holdings H increase by $M/S(t)$ when $t = 0, 1/12, 2/12, \ldots$.

Here is a very simple model of a pension.

We decide to invest in a stock whose price per share is $S(t)$.

Each month we buy M worth of shares, so our holdings H increase by $M/S(t)$ when $t = 0, 1/12, 2/12, \ldots$.

The final value of our pension is

$$V = H(T)S(T)$$

where T is the time at which we retire (e.g. $T = 40$ years).

Here is a very simple model of a pension.

We decide to invest in a stock whose price per share is $S(t)$.

Each month we buy M worth of shares, so our holdings H increase by $M/S(t)$ when $t = 0, 1/12, 2/12, \ldots$.

The final value of our pension is

$$V = H(T)S(T)$$

where T is the time at which we retire (e.g. $T = 40$ years).

Will V be enough to live on?

To answer questions about V, we need a model for $S(t)$.

To answer questions about V, we need a model for $S(t)$.

A reasonable first choice is *geometric Brownian motion*:

$$\mathrm{d}S = \mu S \mathrm{d}t + \sigma S \mathrm{d}W.$$

To answer questions about V, we need a model for $S(t)$.

A reasonable first choice is *geometric Brownian motion*:

$$\mathrm{d}S = \mu S \mathrm{d}t + \sigma S \mathrm{d}W.$$

This is a *stochastic differential equation* (SDE). It defines a stochastic process $S(t)$, a family of random variables indexed by time.

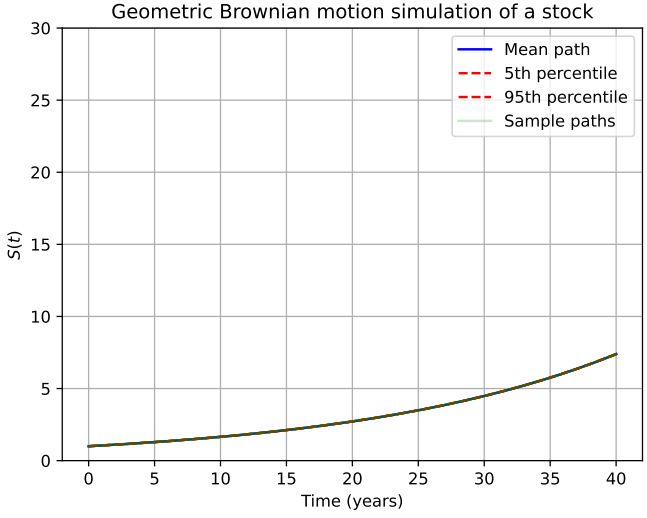
To answer questions about V, we need a model for $S(t)$.

A reasonable first choice is *geometric Brownian motion*:

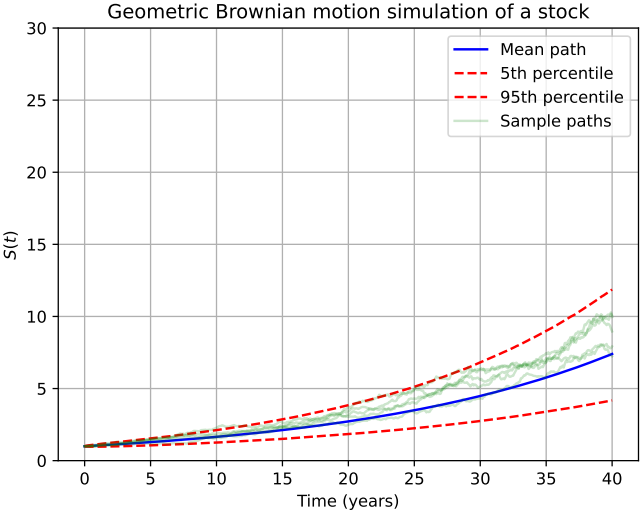
$$\mathrm{d}S = \mu S \mathrm{d}t + \sigma S \mathrm{d}W.$$

This is a *stochastic differential equation* (SDE). It defines a stochastic process $S(t)$, a family of random variables indexed by time.

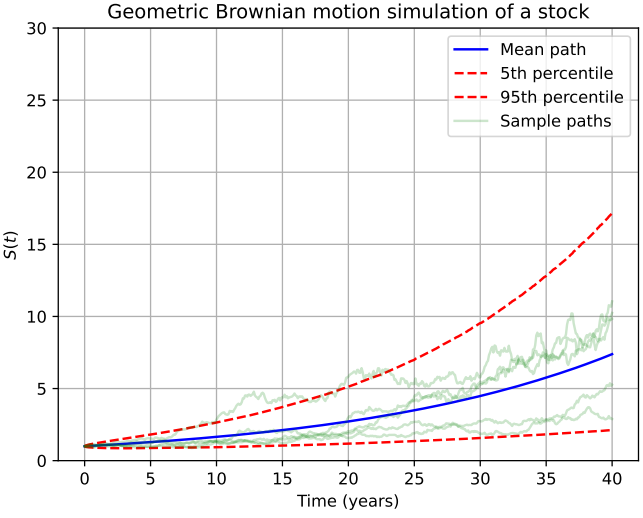
Here μ is the expected real growth rate (e.g. 5%), W is one-dimensional Brownian motion, and σ is the *volatility*, a measure of how random the trajectories are.


Geometric Brownian motion simulation of a stock
Mean path
5th percentile
95th percentile
Sample paths
S(t)
Time (years)
0
5
10
15
20
25
30
35
40

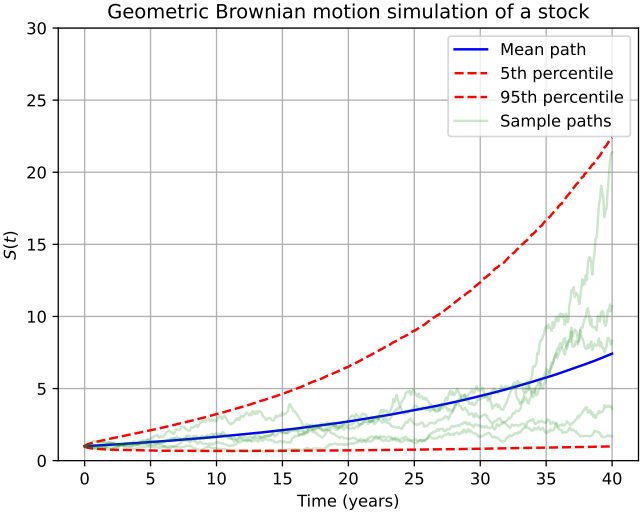
$\sigma = 0$


Geometric Brownian motion simulation of a stock
Mean path
5th percentile
95th percentile
Sample paths
S(t)
Time (years)
0
5
10
15
20
25
30
35
40

$\sigma = 0.05$


Geometric Brownian motion simulation of a stock
Mean path
5th percentile
95th percentile
Sample paths
S(t)
Time (years)
0
5
10
15
20
25
30
35
40

$\sigma = 0.10$


Geometric Brownian motion simulation of a stock
Mean path
5th percentile
95th percentile
Sample paths
S(t)
Time (years)
0
5
10
15
20
25
30
35
40

$\sigma = 0.15$

In this project, we compute samples from the SDE and use them to estimate quantities like

$$\mathbb{P}[V \geq £1,000,000]$$

as a function of the growth rate μ and monthly investment M.

Section 6

Summary

This year's projects are

- computing convex hulls;
- predicting orbits;
- pension planning.

This year's projects are

- computing convex hulls;
- predicting orbits;
- pension planning.

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

Section 7

Previous projects: comments and guidance

Section 8

2024A: 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

In this project we explored algorithms for primality testing.

Question 2024A.1. Modify your code for Exercise 7.6 (which implements an efficient variant of trial division) to return `(flag, ndivisions)`, where `flag = True` if the input is prime and `False` otherwise, and `ndivisions` is the count of the number of divisions performed. Print the output of the function applied to all $n \in 2:20$. How many divisions are performed to test the primality of 9999991111111?

Question 2024A.1. Modify your code for Exercise 7.6 (which implements an efficient variant of trial division) to return `(flag, ndivisions)`, where `flag = True` if the input is prime and `False` otherwise, and `ndivisions` is the count of the number of divisions performed. Print the output of the function applied to all $n \in 2:20$. How many divisions are performed to test the primality of 9999991111111?

$\longrightarrow$ code

Many answers didn't correctly count the number of divisions performed:

```
for m in range(2, math.isqrt(n)):
    if m == 2 or m == 3 or m % 6 == 1 or m % 6 == 5:
        ndivisions += 1
        if n % m == 0:
            return False
```

Many answers didn't correctly count the number of divisions performed:

```
for m in range(2, math.isqrt(n)):
    if m == 2 or m == 3 or m % 6 == 1 or m % 6 == 5:
        ndivisions += 1
        if n % m == 0:
            return False
```

Some answers constructed the list of all numbers to try:

```
numbers_to_try = list(range(5, rootn, 6)) + list(range(7, rootn, 6))
```

Many answers didn't correctly count the number of divisions performed:

```
for m in range(2, math.isqrt(n)):
    if m == 2 or m == 3 or m % 6 == 1 or m % 6 == 5:
        ndivisions += 1
        if n % m == 0:
            return False
```

Some answers constructed the list of all numbers to try:

```
numbers_to_try = list(range(5, rootn, 6)) + list(range(7, rootn, 6))
```

This is egregiously inefficient since it takes $O(\sqrt{n})$ storage whereas there's a straightforward $O(1)$ algorithm.

Many answers didn't correctly count the number of divisions performed:

```
for m in range(2, math.isqrt(n)):
    if m == 2 or m == 3 or m % 6 == 1 or m % 6 == 5:
        ndivisions += 1
        if n % m == 0:
            return False
```

Some answers constructed the list of all numbers to try:

```
numbers_to_try = list(range(5, rootn, 6)) + list(range(7, rootn, 6))
```

This is egregiously inefficient since it takes $O(\sqrt{n})$ storage whereas there's a straightforward $O(1)$ algorithm.

Some answers missed the point completely and tested with each odd number, or worse, with each number up to $\sqrt{n}$.

Question 2024A.2. Compute the number of divisions performed for all numbers $n \in 2 : 10^5$. By means of a plot, verify that trial division takes about $\sqrt{n}/3$ divisions in the worst case to test a number n for primality.

The code for the first question already returns the number of divisions required:

```
N = list(range(2, 10**5+1))
divisions = [trial_division(n)[1] for n in N]
```

The code for the first question already returns the number of divisions required:

```
N = list(range(2, 10**5+1))
divisions = [trial_division(n)[1] for n in N]
```

In contrast, poor answers copied and pasted the answer to the first question and slightly tweaked the code to only return the number of divisions.

The code for the first question already returns the number of divisions required:

```
N = list(range(2, 10**5+1))
divisions = [trial_division(n)[1] for n in N]
```

In contrast, poor answers copied and pasted the answer to the first question and slightly tweaked the code to only return the number of divisions.

This is an opportunity for demonstrating insight. Is the approximation perfect? No, because we ignore the extra two divisions by $\{2, 3\}$.

The code for the first question already returns the number of divisions required:

```
N = list(range(2, 10**5+1))
divisions = [trial_division(n)[1] for n in N]
```

In contrast, poor answers copied and pasted the answer to the first question and slightly tweaked the code to only return the number of divisions.

This is an opportunity for demonstrating insight. Is the approximation perfect? No, because we ignore the extra two divisions by $\{2, 3\}$.

The very best answers (not necessary for full marks) commented that the number of inputs requiring the maximum number of divisions (i.e. prime numbers) appears not to decrease over time—this is because the Prime Number Theorem guarantees that

$$\frac{\pi(n)}{n} \sim \frac{1}{\log n}$$

which decays slowly.

Question 2024A.3. Write a function to implement the Fermat trial for given n and a. Write another function to apply the Fermat test with all a in a given list; if no list is supplied, use as default value all $a \in 2 : (n-2)$ in ascending order. This latter function should return a tuple `(flag, ntrials)` where `flag = False` if the Fermat test has shown n to not be prime and `True` otherwise[1], and where `ntrials` is the number of Fermat trials performed. Print the output of the function applied to the natural numbers $n \in 2 : 20$, using in each case all $a \in 2 : (n-2)$ in ascending order.

[Hint: the greatest common divisor can be computed using `math.gcd`.]

[Hint: in Python, the `pow` function takes an optional third argument. `pow(x, y, z)` calculates $x^y \mod z$.]

[1] In other words, a number with flag `True` might still be composite.

The question explicitly requests two functions. Some answers did not write two functions, or wrote two functions where one did not call the other.

The question explicitly requests two functions. Some answers did not write two functions, or wrote two functions where one did not call the other.

There was some confusion around how to write functions with default values for variables. Such answers generally used global variables to determine the bases to use, which is fragile, and they often worked incorrectly.

Question 2024A.4. Compute the first 5 odd numbers n where the Fermat test proves compositeness with just one trial, i.e. with $a = 2$.

Question 2024A.4. Compute the first 5 odd numbers n where the Fermat test proves compositeness with just one trial, i.e. with $a = 2$.

This was more or less answered correctly by everyone with a correct implementation of the Fermat test.

Question 2024A.5. For how many odd $n \in 3 : 10,000$ does the Fermat test prove compositeness with at most five trials (using $a \in 2 : \min(6, n-2)$)? What proportion of odd composite numbers in $3 : 10,000$ does this represent?

Question 2024A.5. For how many odd $n \in 3 : 10,000$ does the Fermat test prove compositeness with at most five trials (using $a \in 2 : \min(6, n-2)$)? What proportion of odd composite numbers in $3 : 10,000$ does this represent?

Generally answered well. Some students divided by the number of odd numbers, not odd composite numbers.

Question 2024A.5. For how many odd $n \in 3 : 10,000$ does the Fermat test prove compositeness with at most five trials (using $a \in 2 : \min(6, n-2)$)? What proportion of odd composite numbers in $3 : 10,000$ does this represent?

Generally answered well. Some students divided by the number of odd numbers, not odd composite numbers.

Some didn't get the loop right and their count was off by one or two.

Question 2024A.6. A *Carmichael number*, also called an absolute Fermat pseudoprime, is a composite number which passes the Fermat trial for any $a \in 2 : (n-1)$ with $\gcd(a, n) = 1$. Compute the Carmichael numbers up to $10,000$.

[Hint: the first Carmichael number is 561*.]*

Question 2024A.6. A *Carmichael number*, also called an absolute Fermat pseudoprime, is a composite number which passes the Fermat trial for any $a \in 2 : (n-1)$ with $\gcd(a, n) = 1$. Compute the Carmichael numbers up to $10,000$.

[Hint: the first Carmichael number is 561*.]*

Generally answered well. The very best answers (not needed for full marks) investigated whether the number of Carmichael numbers followed the $n^{\frac{2}{7}}$ law prediced by Alford, Granville & Pomerance.

Question 2024A.7. Write a function to implement the Miller–Rabin trial for given n and a.

Write another function to apply the Miller–Rabin test with all a in a given list; if no list is supplied, use as default value the single trial $a = 2$. This latter function should return a tuple `(flag, ntrials)` where `flag = False` if the Miller–Rabin test has shown n to not be prime and `True` otherwise[2], and where `ntrials` is the number of Miller–Rabin trials performed. Print the output of the function applied to the natural numbers $n \in 5 : 20$, using in each case only $a = 2$.

[2] In other words, a number with flag `True` might still be composite.

The main difficulty here was computing s and d such that

$$n - 1 = 2^s d$$

The main difficulty here was computing s and d such that

$$n - 1 = 2^s d$$

```
# Write $n - 1 = 2^s d$.
s = 0
d = n - 1
while d % 2 == 0:
    s += 1
    d = d / 2


assert d == int(d)
d = int(d)
assert n - 1 == 2**s * d
assert d % 2 == 1
```

Question 2024A.8. Using only the single trial with base $a = 2$, what is the minimal odd composite number n for which the test does not conclude that n is composite?

Question 2024A.9. Using only the trials $a \in \{2, 3\}$, what is the minimal odd composite number n for which the test does not conclude that n is composite?

Question 2024A.8. Using only the single trial with base $a = 2$, what is the minimal odd composite number n for which the test does not conclude that n is composite?

Question 2024A.9. Using only the trials $a \in \{2, 3\}$, what is the minimal odd composite number n for which the test does not conclude that n is composite?

These questions are very similar, so write one function that takes in the bases to use.

Question 2024A.8. Using only the single trial with base $a = 2$, what is the minimal odd composite number n for which the test does not conclude that n is composite?

Question 2024A.9. Using only the trials $a \in \{2, 3\}$, what is the minimal odd composite number n for which the test does not conclude that n is composite?

These questions are very similar, so write one function that takes in the bases to use.

This also makes it easier to test the code on the examples that are given in the question—the question tells us the minimal inconclusive odd composite number for bases $\{2, 3, 5\}$.

Question 2024A.10. Using trials $a \in \{2, 3, 5, 7, 11, 13, 17\}$[3], how much faster or slower is the Miller–Rabin test than trial division to verify the primality of $n = 9999991111111$?

[3] With these bases, the Miller–Rabin test is guaranteed to be correct for this n.

Question 2024A.10. Using trials $a \in \{2, 3, 5, 7, 11, 13, 17\}$[3], how much faster or slower is the Miller–Rabin test than trial division to verify the primality of $n = 9999991111111$?

On Windows, it turns out that the operating system timer isn't fine-grained enough to capture the time taken by a Miller–Rabin trial. (I didn't know this.) Many answers therefore reported a time of 0.0 s.

[3] With these bases, the Miller–Rabin test is guaranteed to be correct for this n.

Question 2024A.10. Using trials $a \in \{2, 3, 5, 7, 11, 13, 17\}$[3], how much faster or slower is the Miller–Rabin test than trial division to verify the primality of $n = 9999991111111$?

On Windows, it turns out that the operating system timer isn't fine-grained enough to capture the time taken by a Miller–Rabin trial. (I didn't know this.) Many answers therefore reported a time of 0.0 s.

This obviously doesn't make sense. The right thing to do here is to instead time 100 or 1000 trials, and then divide by the number of trials taken, to get a sensible estimate.

[3] With these bases, the Miller–Rabin test is guaranteed to be correct for this n.

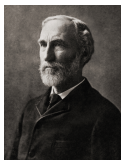
Section 9

2024C: 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



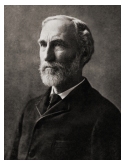
Josiah Willard Gibbs, 1839–1903

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

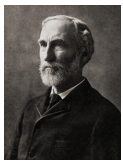
Statistical mechanics, founded by Maxwell, Boltzmann, and Gibbs in the 1800s, applies statistical and probabilistic methods to large assemblies of microscopic entities.

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

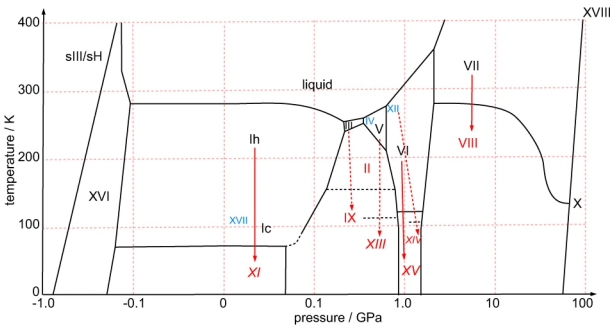
One of the major goals of statistical mechanics is to understand and predict *phase transitions*. A phase transition is an abrupt, discontinuous change in the properties of a system.

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).

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.

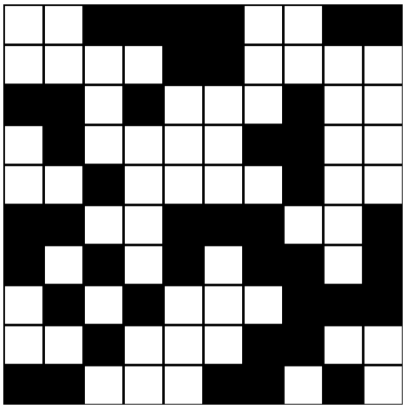
John Hammersley, 1920–2004

Hugo Duminil-Copin, 1985–

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

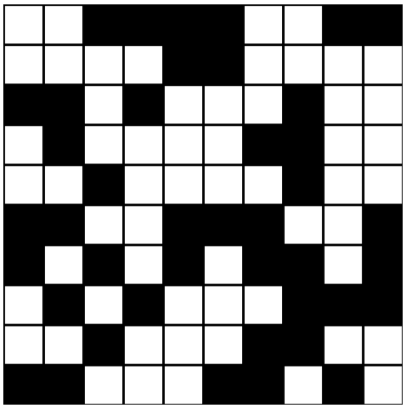
We consider the simplest unsolved case, *Bernoulli site percolation*.

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*.

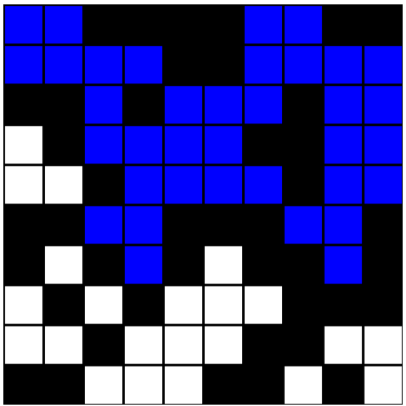
Each site on an $n \times n$ grid is *open* or *closed*, open with probability p.



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*.

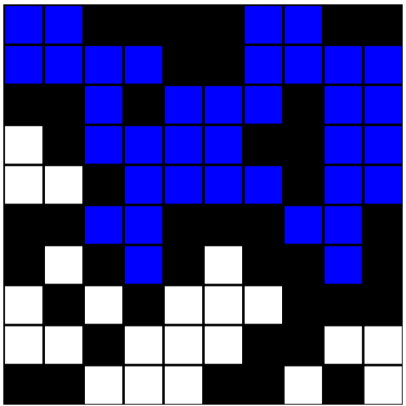
Each site on an $n \times n$ grid is *open* or *closed*, open with probability p.



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.

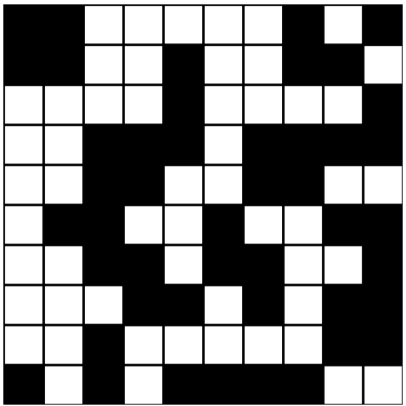


A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

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.

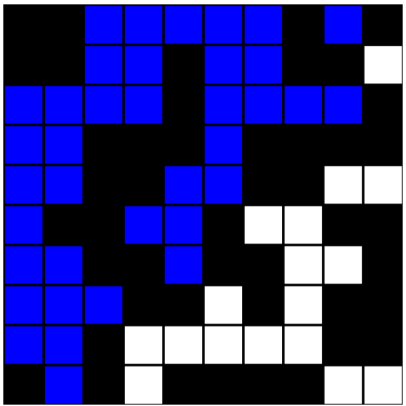


A site is *full* if it is open and can be connected via a chain of open sites to an open site in the top row (moving left, right, up, down).

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.

> *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

Question 2024C.1. Write a function `make_grid(n, p)` to make an $n \times n$ numpy array of Boolean values, with each site `True` with probability p and `False` otherwise.

[Hint: this should take one line of numpy code.]

Draw a few samples to ensure that the empirical probability of a site being open is approximately p.

Question 2024C.1. Write a function `make_grid(n, p)` to make an $n \times n$ numpy array of Boolean values, with each site `True` with probability p and `False` otherwise.

[Hint: this should take one line of numpy code.]

Draw a few samples to ensure that the empirical probability of a site being open is approximately p.

This can be answered in one line using `np.random.binomial(1, p, size=(n, n))`, but many answers didn't use this.

Question 2024C.1. Write a function `make_grid(n, p)` to make an $n \times n$ numpy array of Boolean values, with each site `True` with probability p and `False` otherwise.

[Hint: this should take one line of numpy code.]

Draw a few samples to ensure that the empirical probability of a site being open is approximately p.

This can be answered in one line using `np.random.binomial(1, p, size=(n, n))`, but many answers didn't use this.

Weaker answers forgot to compute the empirical probabilities.

Question 2024C.1. Write a function `make_grid(n, p)` to make an $n \times n$ numpy array of Boolean values, with each site `True` with probability p and `False` otherwise.

[Hint: this should take one line of numpy code.]

Draw a few samples to ensure that the empirical probability of a site being open is approximately p.

This can be answered in one line using `np.random.binomial(1, p, size=(n, n))`, but many answers didn't use this.

Weaker answers forgot to compute the empirical probabilities.

Weaker answers computed the mean with manual loops; better answers used `np.mean` or `np.sum`.

Question 2024C.2. Write a function `visualise_grid` to visualise a grid produced by `make_grid` with matplotlib. The function should take in a Boolean array. The output should look similar to [Figure]: plot closed sites in black; plot open sites in white; colour the borders of each square in black.

[Hint: you will need to consult the matplotlib documentation and other online resources to do this; the relevant matplotlib methods were not discussed in the handbook.]

Use your function to visualise a few sample grids.

Question 2024C.2. Write a function `visualise_grid` to visualise a grid produced by `make_grid` with matplotlib. The function should take in a Boolean array. The output should look similar to [Figure]: plot closed sites in black; plot open sites in white; colour the borders of each square in black.

[Hint: you will need to consult the matplotlib documentation and other online resources to do this; the relevant matplotlib methods were not discussed in the handbook.]

Use your function to visualise a few sample grids.

Some answers flipped black and white. A few answers flipped up with down. You should print out the grid that you render and check by eye that they conform.

Question 2024C.3. Write a function `visualise_fill` to visualise the fill status of a given grid. The function should take as input two Boolean arrays, the grid and the fill status. The output should look similar to [Figures]. Plot closed sites in black, open unfilled sites in white, and open filled sites in blue. Colour the borders of each square in black.

Question 2024C.3. Write a function `visualise_fill` to visualise the fill status of a given grid. The function should take as input two Boolean arrays, the grid and the fill status. The output should look similar to [Figures]. Plot closed sites in black, open unfilled sites in white, and open filled sites in blue. Colour the borders of each square in black.

My comments here are similar.

Question 2024C.4. Write a function `compute_fill` that takes in a grid produced by `make_grid` and calculates whether each site is full or not.

[Hint: think carefully about what should happen when a site is visited. For example, if it is already full, it should terminate without further action.]

[Hint: it may be useful during your development to visualise the fill state of the grid as you visit each site in the top row.]

Question 2024C.4. Write a function `compute_fill` that takes in a grid produced by `make_grid` and calculates whether each site is full or not.

[Hint: think carefully about what should happen when a site is visited. For example, if it is already full, it should terminate without further action.]

[Hint: it may be useful during your development to visualise the fill state of the grid as you visit each site in the top row.]

Implementations ranged from very elegant to very convoluted. Some implementations swept through the grid row-by-row, not allowing propagation of fill upwards.

Question 2024C.5. Write a function `percolates` that returns `True` if the given grid percolates, and `False` otherwise.

[Hint: the core logic can be written with one line of numpy.]

Draw 10 samples of a 20×20 grid with vacancy probability $p = 0.6$. For each, visualise its fill status, titling each figure with whether that grid percolates or not.

Question 2024C.5. Write a function `percolates` that returns `True` if the given grid percolates, and `False` otherwise.

[Hint: the core logic can be written with one line of numpy.]

Draw 10 samples of a 20×20 grid with vacancy probability $p = 0.6$. For each, visualise its fill status, titling each figure with whether that grid percolates or not.

Some implementations were less efficient than necessary: they didn't terminate once a filled site was found on the bottom row.

Question 2024C.5. Write a function `percolates` that returns `True` if the given grid percolates, and `False` otherwise.

[Hint: the core logic can be written with one line of numpy.]

Draw 10 samples of a 20×20 grid with vacancy probability $p = 0.6$. For each, visualise its fill status, titling each figure with whether that grid percolates or not.

Some implementations were less efficient than necessary: they didn't terminate once a filled site was found on the bottom row.

A few answers were egregiously inefficient: they called `compute_fill` over and over on each column.

Question 2024C.6. Take a suitable grid $P \subset [0,1]$ of p values. (You may wish to increase the resolution for $p \in [0.4, 0.7]$.) For each $p \in P$, draw N samples of a 20×20 grid with vacancy probability p. For each sample, calculate whether the grid percolates or not; the fraction of grids that percolates is our estimate for $C(p)$. Plot $C(p)$ as a function of p.

[Hint: you will need to choose suitable N and P so that the interpolation error and statistical error due to sampling are acceptable. The curve should appear smooth; if it is not, try increasing N and/or refining P.]

Question 2024C.6. Take a suitable grid $P \subset [0,1]$ of p values. (You may wish to increase the resolution for $p \in [0.4, 0.7]$.) For each $p \in P$, draw N samples of a 20×20 grid with vacancy probability p. For each sample, calculate whether the grid percolates or not; the fraction of grids that percolates is our estimate for $C(p)$. Plot $C(p)$ as a function of p.

[Hint: you will need to choose suitable N and P so that the interpolation error and statistical error due to sampling are acceptable. The curve should appear smooth; if it is not, try increasing N and/or refining P.]

The main problem here was insufficient samples; trial and error shows you needed around $N = 20,000$ for a reasonably smooth $C(p)$ curve.

Question 2024C.6. Take a suitable grid $P \subset [0,1]$ of p values. (You may wish to increase the resolution for $p \in [0.4, 0.7]$.) For each $p \in P$, draw N samples of a 20×20 grid with vacancy probability p. For each sample, calculate whether the grid percolates or not; the fraction of grids that percolates is our estimate for $C(p)$. Plot $C(p)$ as a function of p.

[Hint: you will need to choose suitable N and P so that the interpolation error and statistical error due to sampling are acceptable. The curve should appear smooth; if it is not, try increasing N and/or refining P.]

The main problem here was insufficient samples; trial and error shows you needed around $N = 20,000$ for a reasonably smooth $C(p)$ curve.

Here efficiency of the code matters (for you more than for me): with a reasonable implementation, it took my code about 7 minutes to answer this question. If your implementation is very inefficient, then that could become hours, or days.

Conclusion

Best of luck with the projects!