Discretisation

The physical systems we model are described by continuous mathematical functions, $f(x, t)$ and their derivatives in space and time.

To represent this continuous system on a computer we must discretize it—convert the continuous function into a discrete number of points in space at one or more discrete instances in time.

There are many different discretization methods used throughout the physical sciences, engineering, and applied mathematics fields, each with their own strengths and weaknesses.

Grid-based vs. Grid-less methods

Broadly speaking, we can divide these methods into:

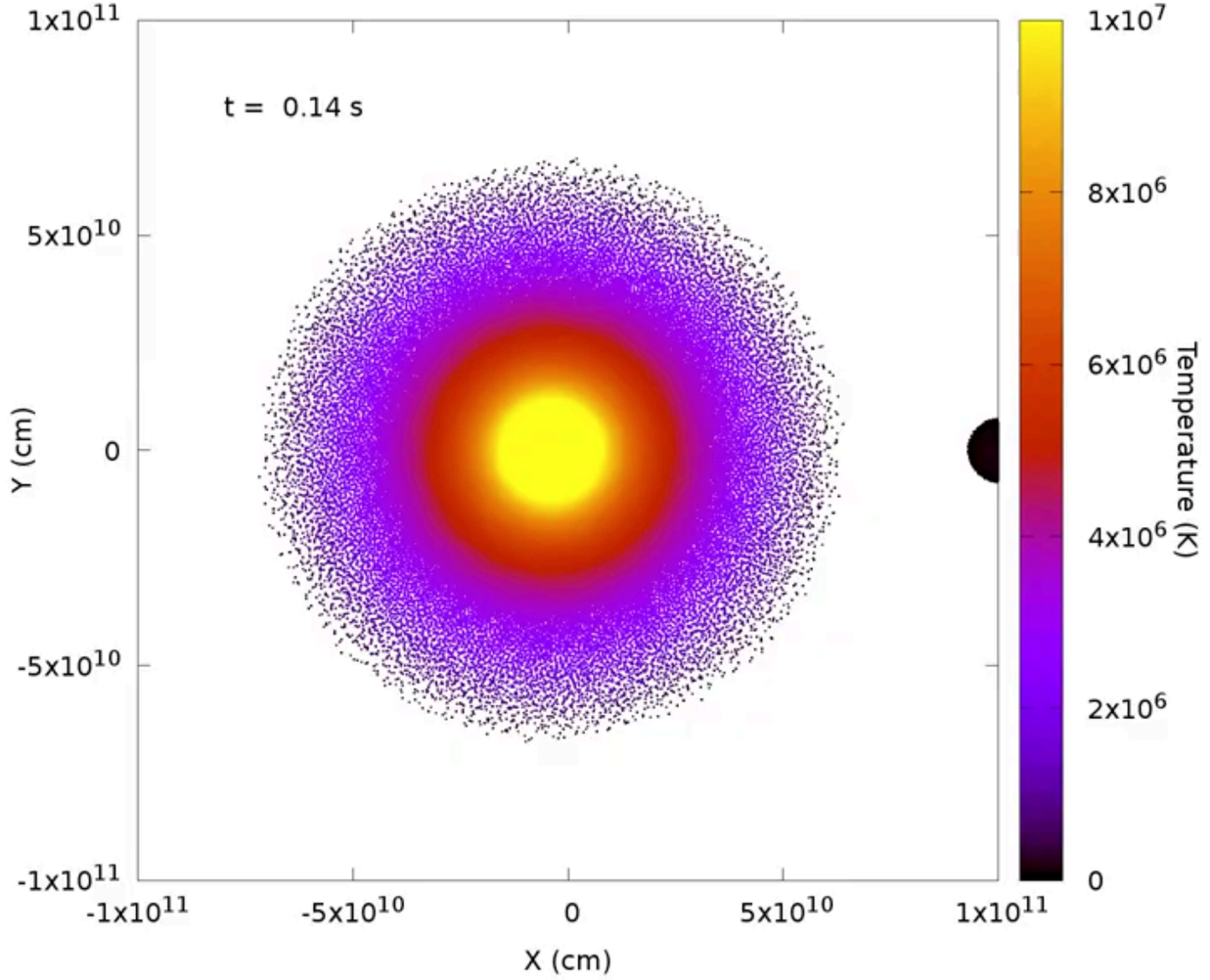
grid-based and grid-less methods.

Grid-less methods include those which represent the function as a superposition of continuous basis functions (e.g. sines and cosines). This is the fundamental idea be- hind ***spectral methods*.**

A different class of methods are those that use discrete particles to represent the mass distribution and produce continuous functions by integrating over these particles with a suitable kernel—this is the basis of ***smoothed particle hydrodynamics*** **(SPH)**.

Introduction to CFD: discretisation

SPH simulations

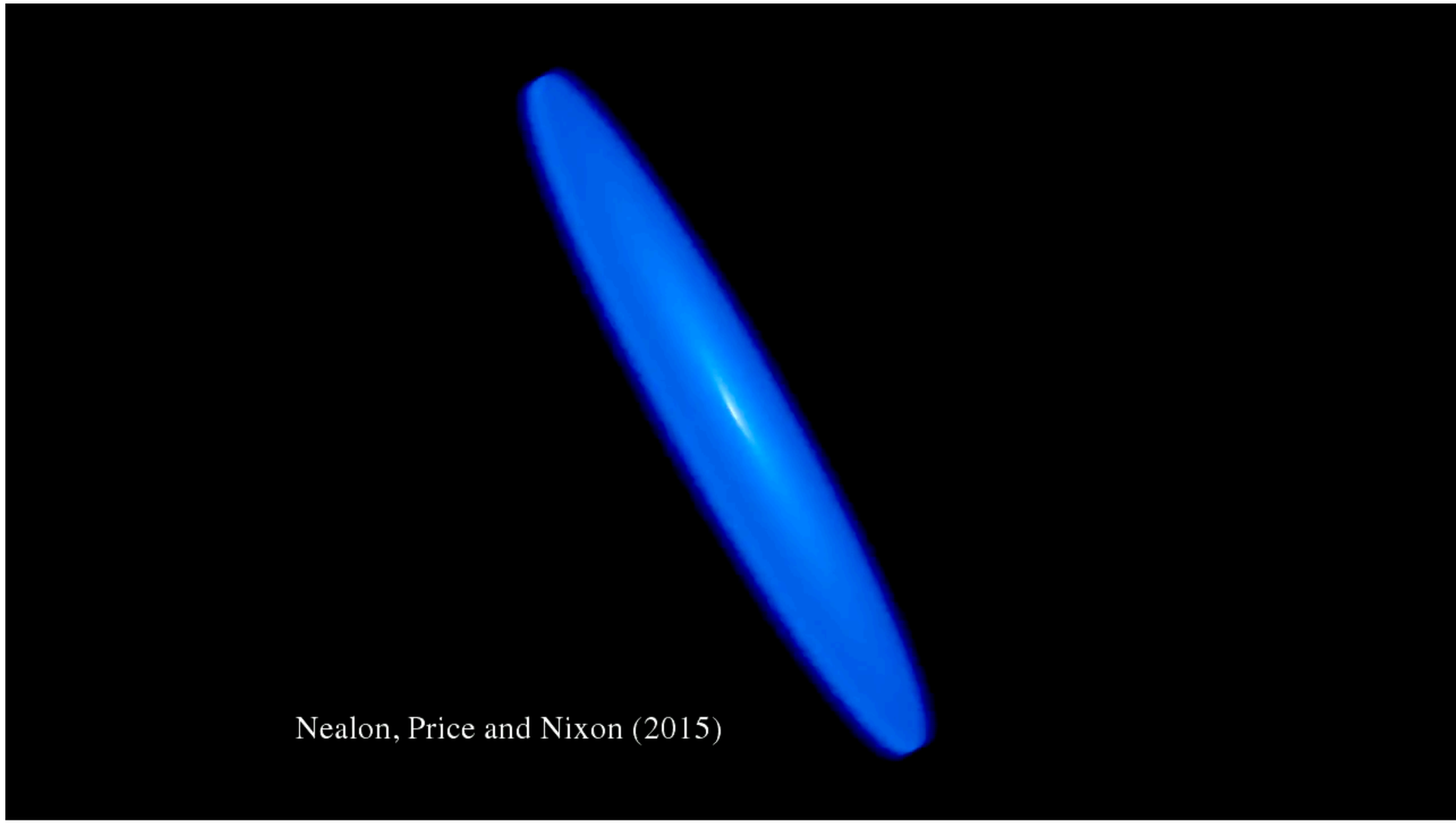


Merger of a brown dwarf (BD) star of 0.019 solar masses with a main sequence (MS) star of 1 solar mass with the smoothed particle hydrodynamics (SPH) code SPHYNX. The BD has 100,000 SPH particles, while the MS has 5.2 million. Color represents temperature. The simulation is in 3D, but only a thin slice centred in Z=0 is shown in this video.

Pre-print: https://arxiv.org/pdf/2210.17363.pdf

SPH simulations

SPH simulation of an accretion disc around a black hole, initially inclined at 120 degrees to the plane of the black hole spin (vector up the page). As the disc evolves the torque produced by the rotating black hole is stronger than the internal torque holding the disc together, causing rings of material to be torn off and precess effectively independently. This disc runs for almost 1600 orbits at the inner edge and the simulation uses 10 million particles.



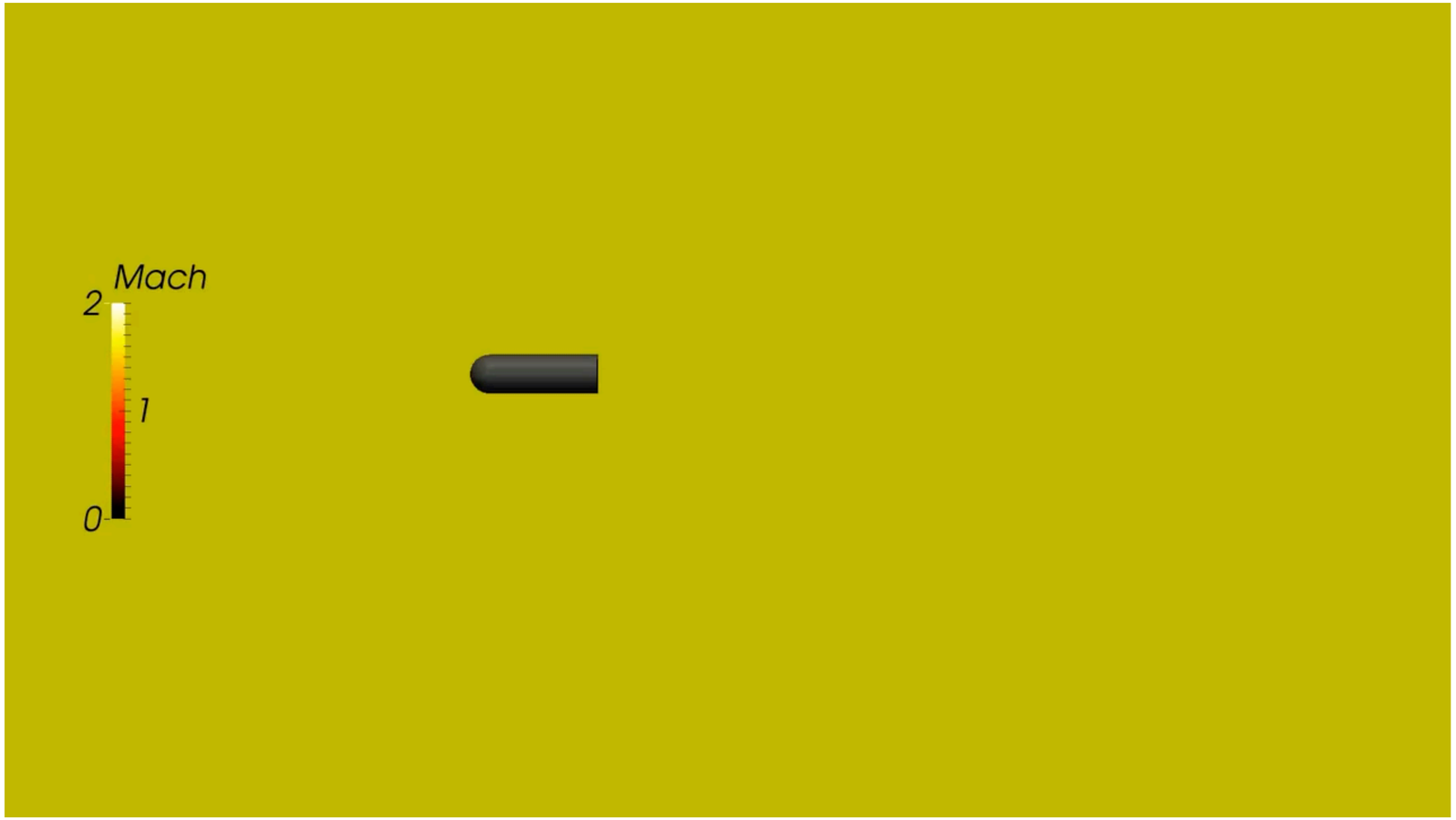
Grid-based methods

For grid-based methods, we talk about both the style of the grid (structured vs. unstructured) and the discretisation method, e.g.

- the finite-difference
- finite-volume, and
- finite-element methods.

Introduction to CFD: discretisation

Grid-based simulations



This is a 2D CFD simulation of a supersonic object moving with a Mach number of 1.6 done with OpenFoam.

https://www.youtube.com/watch?v=D6iuVr9V6os

Grid-based simulations

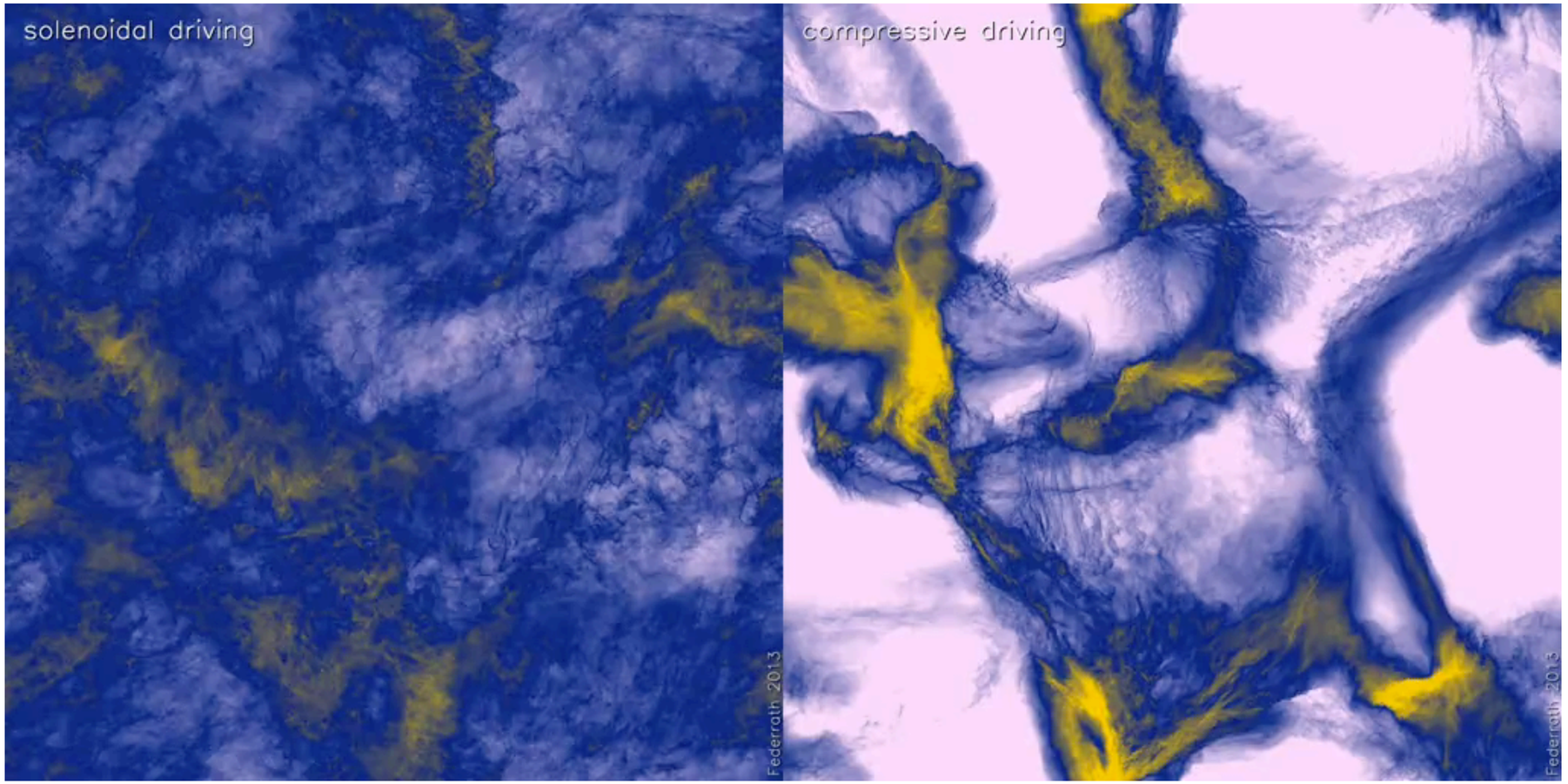
Divergence-free driving (Solenoidal)

Curl-free driving (Compressive)

Shearing - Gas collisions - Outflows/Winds - Shocks - SN explosions - Gravity

Rotation

Compression



Projected gas density (Mach 17 turbulence) by Federrath 2013

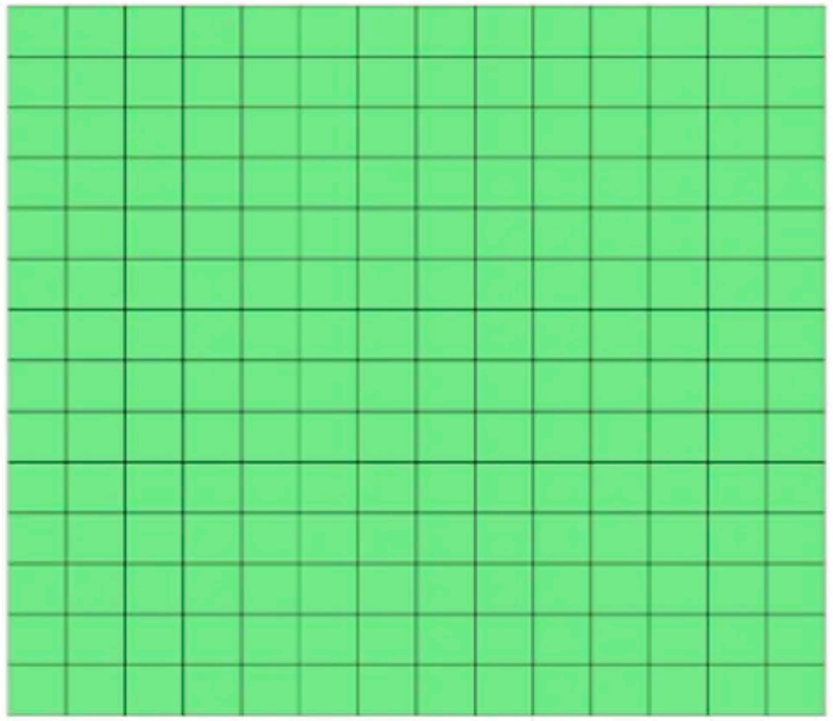
Structured vs. Unstructured Grids

Structured grids are logically Cartesian. This means that you can reference the location of any cell in the computational domain via an integer index in each spatial dimension. From a programming standpoint, the grid structure can be represented exactly by a multi-dimensional array.

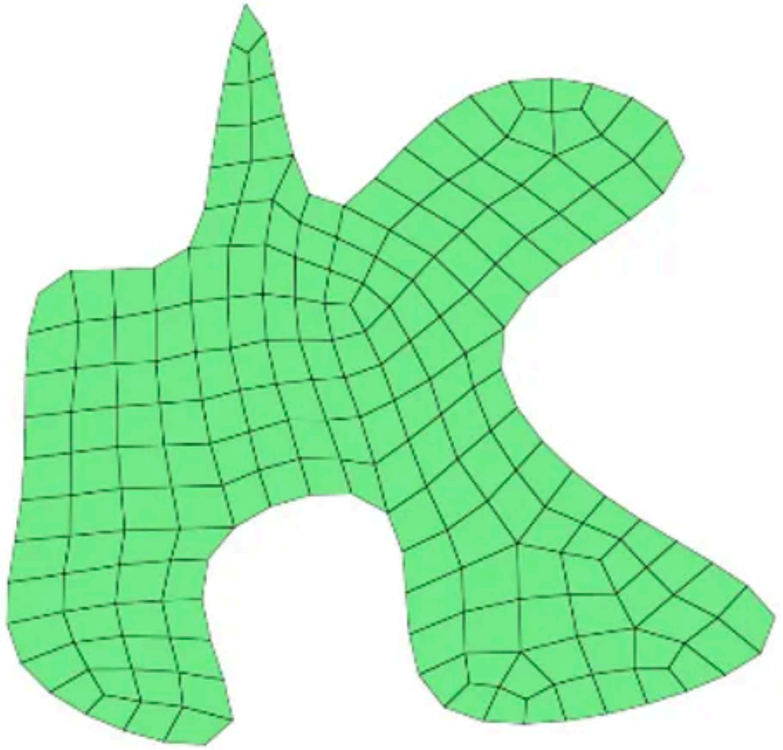
Unstructured grids don’t have this simple pattern.

The main **advantage** of these grids is that you can easily represent irregularly-shaped domains. The **disadvantage** is that the data structures required to describe the grid are more complicated than a simple array (and tend to have more inefficient memory access).

Structured vs. Unstructured Grids



Structured Mesh



Unstructured Mesh

https://onscale.com/blog/meshing-in-fea-structured-vs-unstructured-meshes/

Numerical Grids

Once a grid is established, the system of PDEs is converted into a system of discrete equations on the grid.

Finite-difference and **finite-volume** methods can both be applied to *structured grids.*

The main difference between these methods is that the **finite-difference methods** build from the differential form of PDEs while the **finite-volume methods** build from the integral form of the PDEs.

The attractiveness of **finite-volume methods** is that conservation is a natural consequence of the discretisation.

Advection Equation

The linear advection equation is simply:

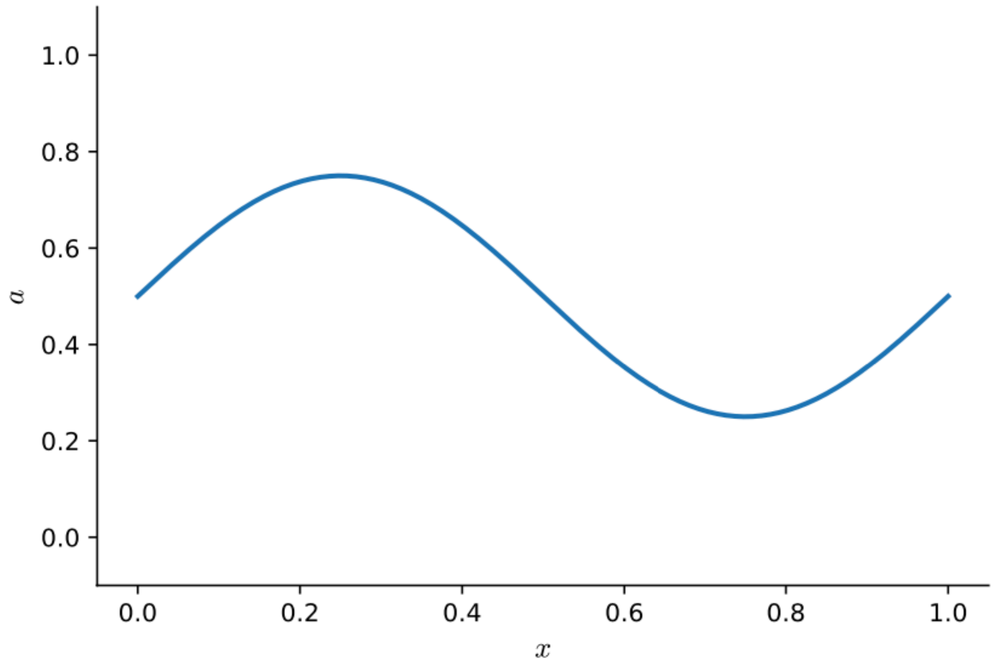
$$a_t + ua_x = 0$$

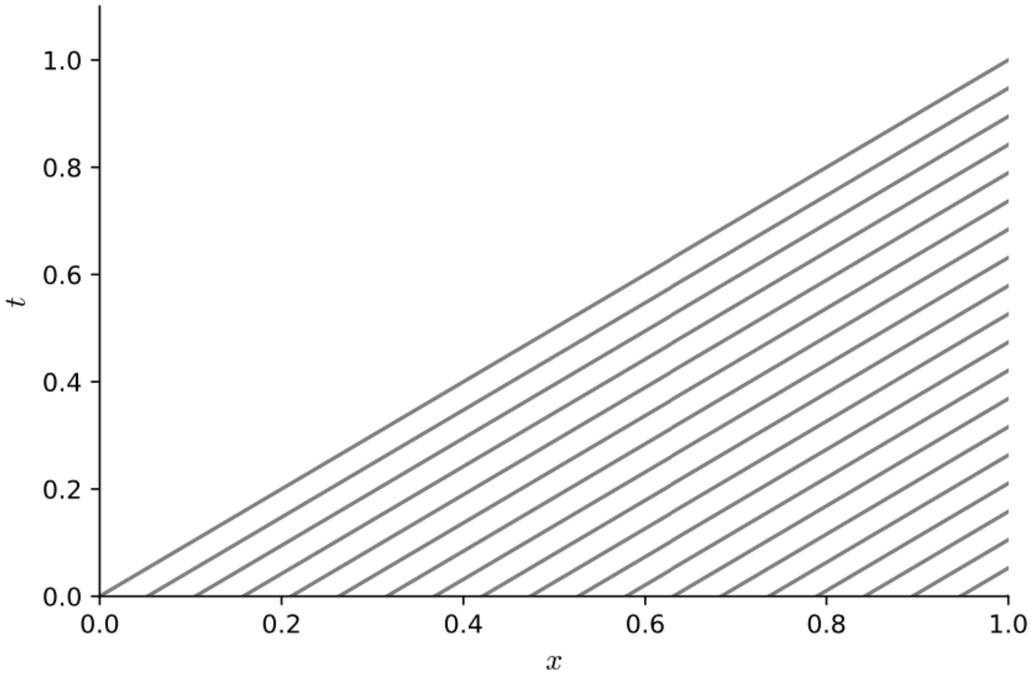
where $a(x,t)$ is some scalar quantity and u is the velocity at which it is advected ($u > 0$ advects to the right).

The solution is to simply take the initial data, $a(x,t = 0)$, and displace it to the right at a speed u. The shape of the initial data is preserved in the advection.

Many hyperbolic systems of PDEs, e.g. the equations of hydrodynamics, can be written in a form that looks like a system of (nonlinear) advection equations, so the advection equation provides important insight into the methods used for these systems.

Advection Equation





These panels show the initial profile, $a(x)$, and the corresponding characteristics in the t-x plane.

The solution is constant along these characteristics, so each point simply follows the characteristic curve, resulting in a shift of the profile to the right.

Advection Equation: Stability

An important concept to consider is *stability*.

Not every discretisation that we write down will be well behaved. For some, our initial state will begin to “blow-up”, and take on very large and unphysical values after just a few steps. This is the hallmark of a method that is unstable.

Some methods have restrictions on the size of the timestep that result in a stable methods as well.

First-order advection in 1-d and finite-differences

To get a flavour of the methods for advection, we will use a simple finite-difference discretisation—here, the domain is divided into a sequence of points where we store the solution.

$$a_t + ua_x = 0$$

We will solve this equation numerically by discretising the solution at these points.

The index *i* denotes the point's location, and *ai* denotes the discrete value of *a*(*x*) in zone *i*. The data in each zone can be initialised as *ai* = *a*(x_i).

First-order advection in 1-d and finite-differences

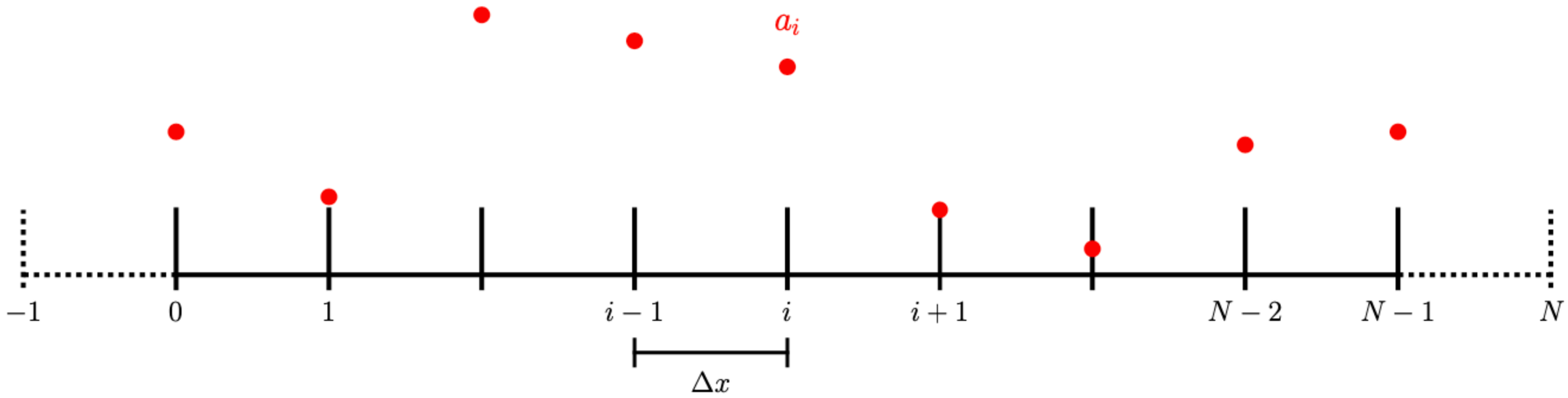


Figure 4.2: A simple finite-difference grid. The solution is stored at each of the labeled points. The dotted lines show the ghost points used to extend our grid past the physical boundaries to accommodate boundary conditions. Note that if we are periodic, then points 0 and $N-1$ are at the same physical point in space, so we would only need to update one of them.

First-order advection in 1-d and finite-differences

We also need to discretize in time. We denote the time-level of the solution with a superscript, so $a_i^n = a(x_i, t^n)$. For a fixed Δt, time level n corresponds to a time of $t = n\Delta t$.

A simple first-order accurate discretisation is:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -u \frac{a_i^n - a_{i-1}^n}{\Delta x}$$

This is an *explicit* method, since the new solution, a_i^{n+1}, depends only on information at the old time level, n.

First-order advection in 1-d and finite-differences

Finally, we also need to specify a boundary condition for this. Our choice of spatial derivative is one-sided—it uses information from the zone to the left of the zone we are updating.

This is because information is flowing from left to right in this problem ($u > 0$). This choice of the derivative is called *upwinding*—this choice of derivative results in a stable method.

First-order advection in 1-d and finite-differences

The Ghost Zone

Notice that if we use this method to update the data in the first zone inside the boundary, we need data to the left of this zone—outside of the domain.

This means that we need a single *ghost point* to implement the boundary conditions for our method. The presence of the ghost points allow us to use the same update equation for all zones in the domain.

First-order advection in 1-d and finite-differences
Time-stepping & the CFL condition

The last piece of information needed to update the solution is the timestep, Δt.

It can be shown that for the solution to be *stable*, the time-step must be less than the time it takes information to propagate across a single zone. That is:

$$\Delta t \leq \frac{\Delta x}{u}$$

This is called the **Courant-Friedrichs-Lewy or CFL condition.**

First-order advection in 1-d and finite-differences
Time-stepping & the CFL condition

A dimensionless quantity called the *CFL number* is defined as: $$\mathcal{C} = \frac{\Delta t u}{\Delta x}$$

Stability requires C ≤ 1. We traditionally write the time-step as: $$\Delta t = \mathcal{C}\frac{\Delta x}{u}$$

and specify C as part of the problem (a typical value may be C = 0.7).

First-order advection in 1-d and finite-differences

Time-stepping & the CFL condition

When you use C = 1 in the first-order differenced advection equation, you advect the profile exactly, without any numerical error.

However, in general, we will be solving a non-linear system of equations, so it is not possible to run with C = 1, since *u* (and therefore C) will change from zone to zone.

Instead, one looks at **the most restrictive time-step over all the zones and uses that for the entire system.**

Introduction to CFD: discretisation

Exercise 4.3

Write a code to solve the 1-d linear advection equation using the discretization of Eq. 4.2 on the domain $[0,1]$ with $u = 1$ and periodic boundary conditions. For initial conditions, try both a Gaussian profile and a top-hat:

$$a = \begin{cases} 0 & \text{if} \quad & x & < 1/3 \\ 1 & \text{if} \quad 1/3 \leq & x & < 2/3 \\ 0 & \text{if} \quad 2/3 \leq & x & \end{cases} \tag{4.6}$$

Note: For a general treatment of boundary conditions, you would initialize the ghost points to their corresponding periodic data and apply the difference equations to zones $0, \ldots, N-1$. However, for periodic BCs on this grid, points 0 and $N-1$ are identical, so you could do the update in this special case on points $1, \ldots, N-1$ without the need for ghost points and then set $a_0 = a_{N-1}$ after the update.

Run you program for one or more periods (one period is $T = 1/u$) with several different CFL numbers and notice that there is substantial numerical dissipation (see Figure 4.3).