Photo: Wikipedia / Alnus

Vinous-throated Parrotbill
粉紅鸚嘴

ASTR 660 / Class 11

Fluids, finite differences, 1-d advection

Astrophysical fluids

Most of the difficult problems in modern astrophysics involve some sort of fluid dynamics.

Astrophysical fluids are often rather “simple”, from the point of view of fluid dynamics.
Common complications include:

- Gravity;
- Multiphase media (not physically mixed, but separation not resolved);
- Very large dynamic range;
- Unresolved, complicated microphysics (star formation, supernovae, radiation, nuclear/chemical reactions) — handled through “sub-grid” models;
- Relativity.

SPH vs. Mesh

Two basic approaches to fluid dynamics:

Lagrangian (SPH): follow the properties of discrete fluid elements (particles) over time. Like an N-body simulation. Good at dealing with mixtures of different fluid phases but smooths out features like shocks.

Eulerian (grid/mesh): follow the evolution of fluid properties on a (regular) grid of (fixed) points in space. Good at capturing shocks, but tends towards over-mixing.

Most fluid dynamics codes are grid-based, but SPH turned out to be very well suited to many astrophysical problems.

Fluid equations

The core equations of fluid dynamics are PDEs in a set of state variables for the fluid, defined on a coordinate grid $\vec{x}$ (the *Eulerian* formulation).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \quad \text{(Continuity)}$$

$$\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} = -\frac{1}{\rho} \nabla p - \nabla \Phi \quad \text{(Energy / equation of motion)}$$

$$\nabla^2 \Phi = 4\pi G \rho \quad \text{(Gravity)}$$

$$p = p(\rho, S) \implies \nabla p = \left(\frac{\partial p}{\partial \rho} \right)_S \nabla \rho + \left(\frac{\partial p}{\partial S} \right)_\rho \nabla S \quad \text{etc.} \quad \text{(Equations of state; } S\text{: entropy)}$$

Partial Differential Equations

PDEs are complicated! They can be single (self-contained) or coupled (e.g., one describing heat flow, another describing advection, another describing gravity … all for the same fluid).

We'll deal almost entirely with *single* 2nd-order linear PDEs, like the heat equation, the wave equation and Poisson's equation.

For example, Poisson's equation for gravity, $\nabla^2\Phi = 4\pi G\rho$, in two dimensions is a second-order equation of the form:

$$\frac{d^2\Phi}{dx^2} + \frac{d^2\Phi}{dy^2} = f(x, y)$$

Advection

The amplitude of a wave, $u(x,t)$, obeys $\dfrac{\partial u}{\partial t} = -a\dfrac{\partial u}{\partial x}$.

(This is $\dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0$ in 1 dimension)

The solution is $u(x,t) = u_0(x - at)$.

As t increases, the wave moves to the right (*positive* x) with speed a.

Heath, Ch. 11.

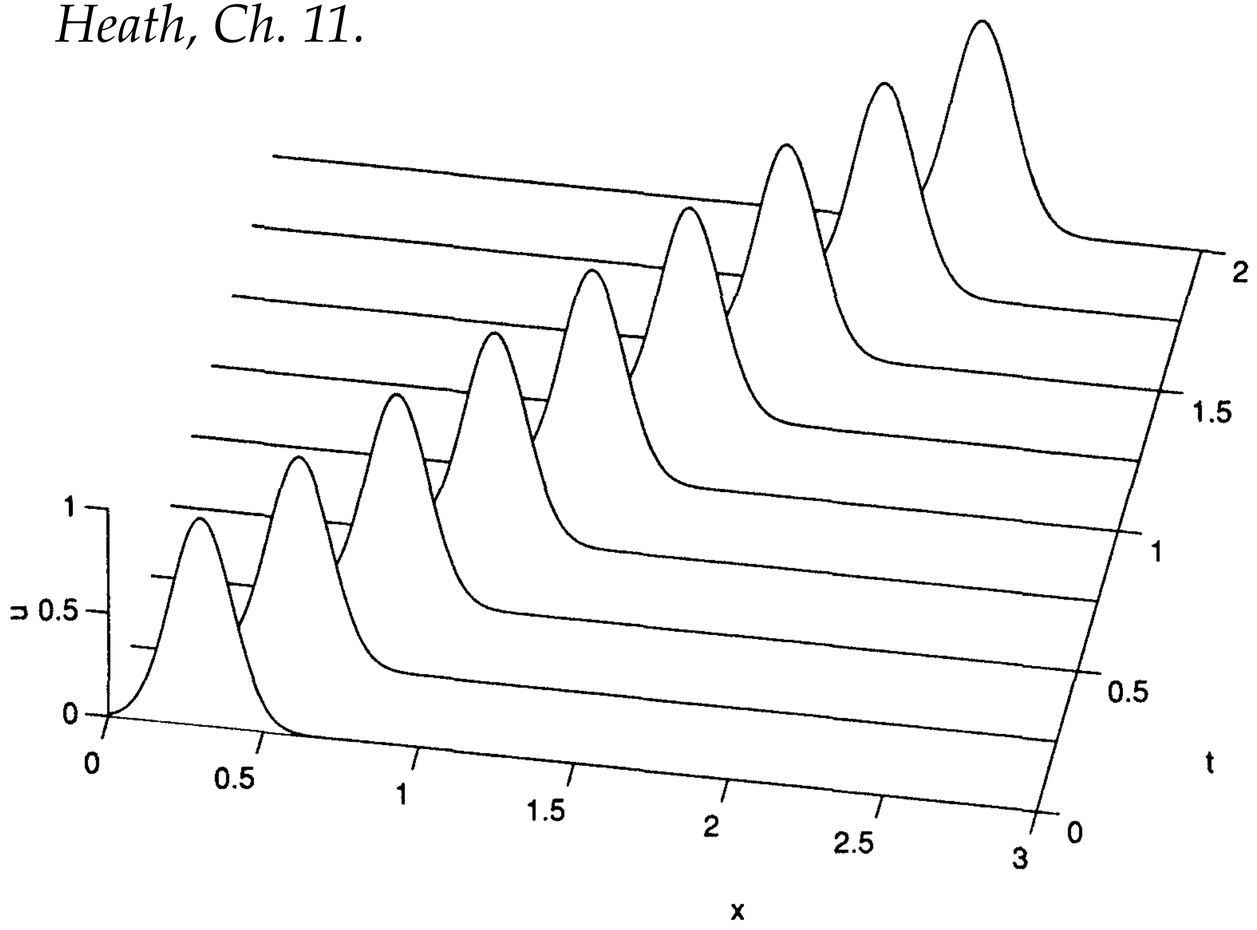


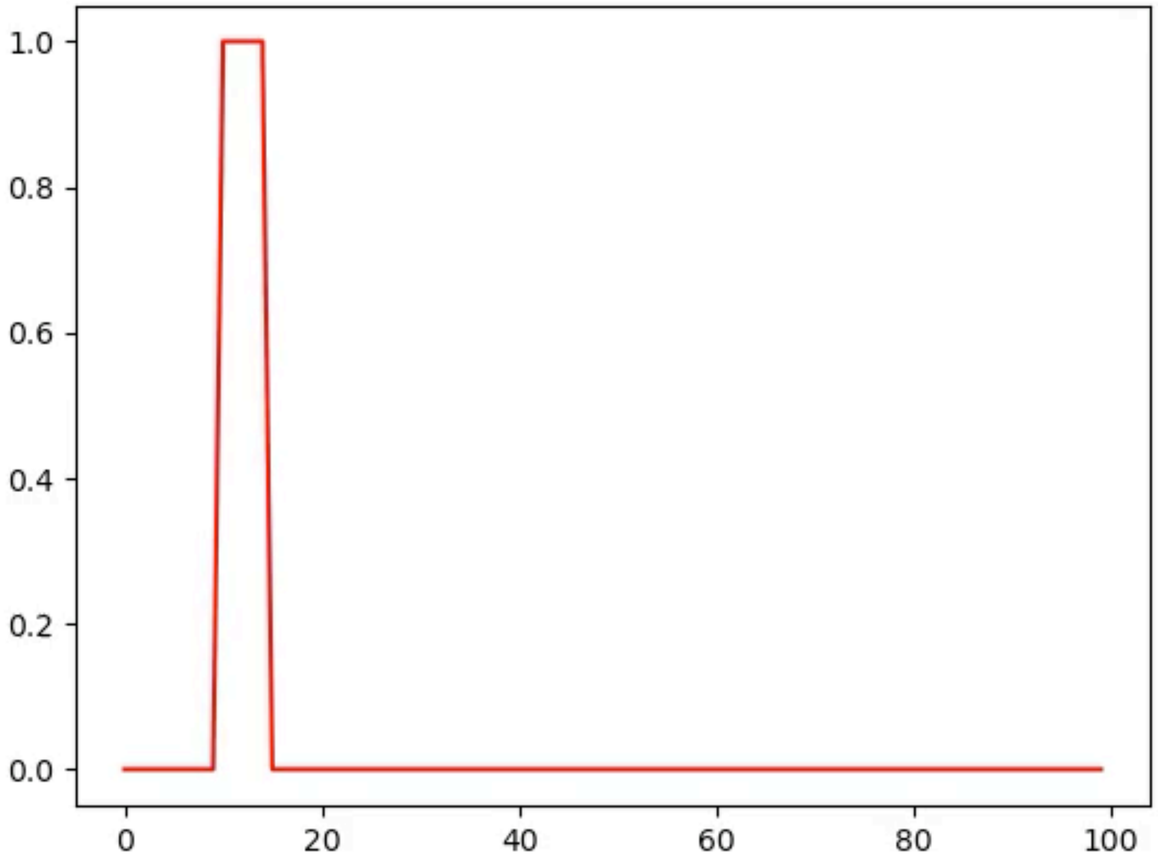
Figure 11.1: Typical solution of advection equation.

Advection: analytic solution

$$u(x,t) = u_0(x - at)$$

```
# Set up the time grid
nsteps = 1000
dt     = 10

# Set up the space grid
ngrid = 100
dx    = 1
igrid = np.arange(0,ngrid,dx)

# Set up the space-time array
u = np.zeros((nsteps,ngrid))
# Specify square wave ICs
u[0,10:15] = 1.0

a = 1 # Advection speed
```

```
for n in range(1,nsteps):
    i = igrid.take(igrid - a*n,mode='wrap')
    u[n,:] = u[0,i]
```



Advection: numerical algorithm

Our equation has derivatives in time and derivatives in space:

$$\frac{\partial u}{\partial t} = -a\frac{\partial u}{\partial x}.$$

Let's discretise the problem (consider a grid of n steps in t and m steps in x), and substitute the derivatives with the finite-difference formulae (see previous classes).

To start with, we will use forward differences (Euler's method) for time and space:

$$\implies \frac{u_i^{n+1} - u_i^n}{\Delta t} = -a\frac{u_{i+1}^n - u_i^n}{\Delta x}$$

The superscripts represent the time steps and the subscripts represent the space steps, so u_{i+1}^n stands for $u_i(t = t_n)$: the value at the i'th grid point, at the n'th timestep.

Advection: forward time, forward space

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -a\frac{u_{i+1}^n - u_i^n}{\Delta x} \implies u_i^{n+1} = u_i^n - \frac{a\Delta t}{\Delta x}\left(u_{i+1}^n - u_i^n\right)$$

If we represent the domain of the problem as a "space time" grid $u[n, i]$, this suggests an "explicit" algorithm that starts from initial conditions $u[0, :]$ and calculates the other points:

```
for each n > 0:
  for each i:
    u[n,i] = u[n-1,i] - C * ( u[n-1,i] - u[n-1,i-1] )
```

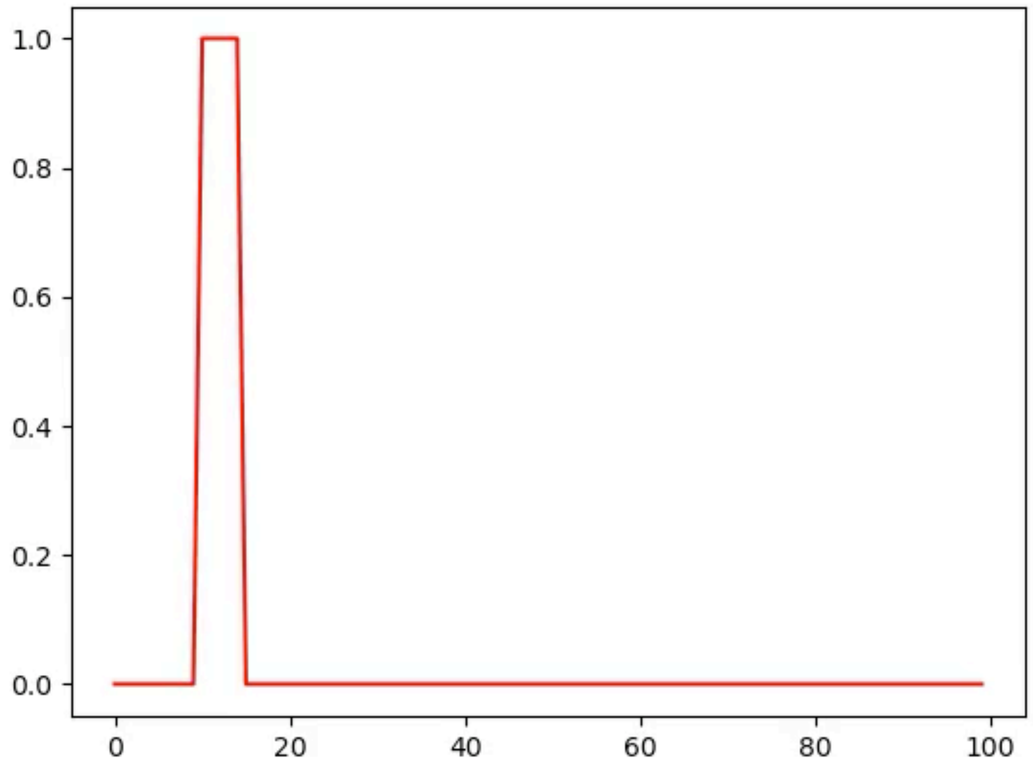
The factor is $C = \frac{a\Delta t}{\Delta x}$.

We need to choose a **timestep**, Δt, and a grid **spacing**, Δx. We also need to think about the **boundary conditions**: what do we take for u_{i-1} at $i = 0$?

Advection: forward time, forward space

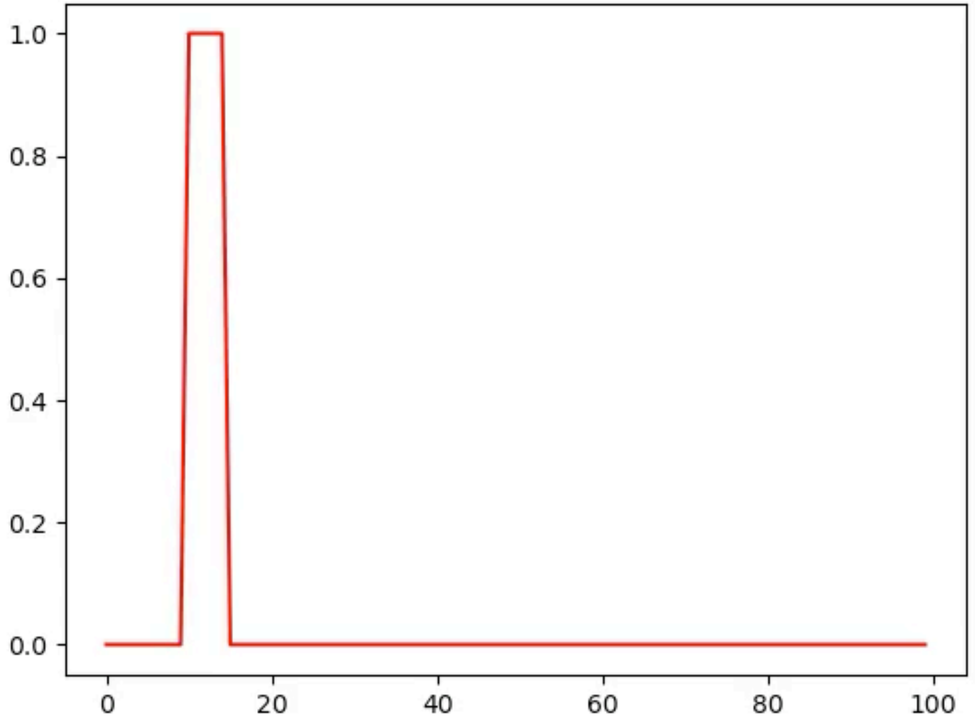
```
# Set up the time grid
nsteps = 1000
dt     = 10

# Set up the space grid
ngrid = 100
dx    = 1
igrid = np.arange(0,ngrid,dx)
(
# Set up the space-time array
u = np.zeros((nsteps,ngrid))
# Square wave ICs
u[0,10:15] = 1.0

a = 1 # Advection speed
C = -a /(dx/dt)

# Numpy: make a wrapped list of "i+1"
i_upper = igrid.take(igrid+1,mode='wrap')
```

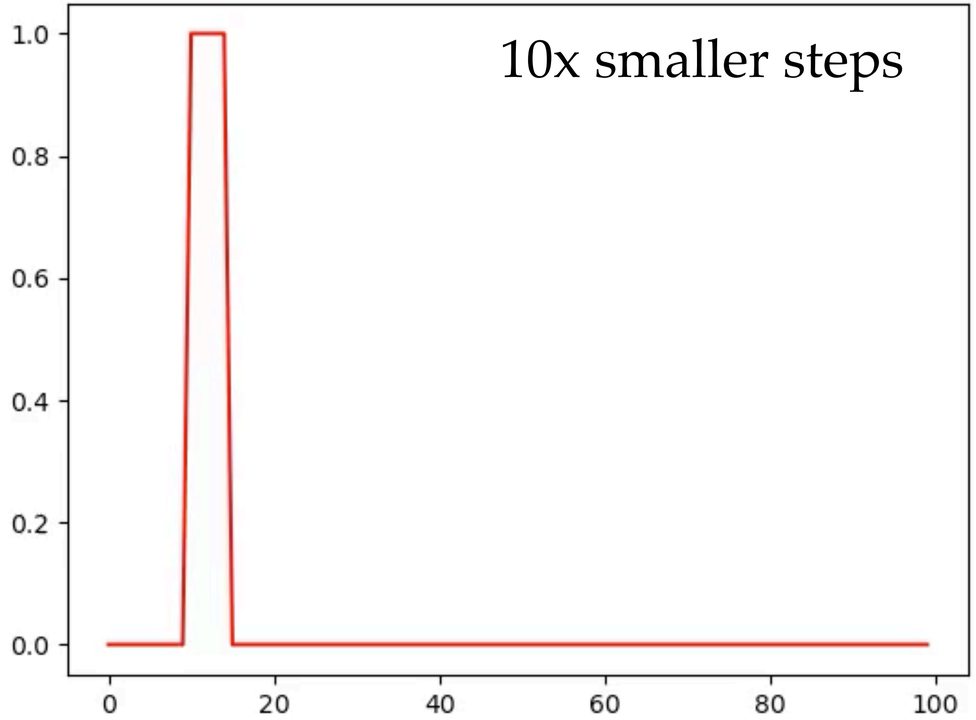
```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - C*(u[n-1,i_upper] - u[n-1,:])
```



Advection: forward time, forward space

```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - C*(u[n-1,i_upper] - u[n-1,:])
```

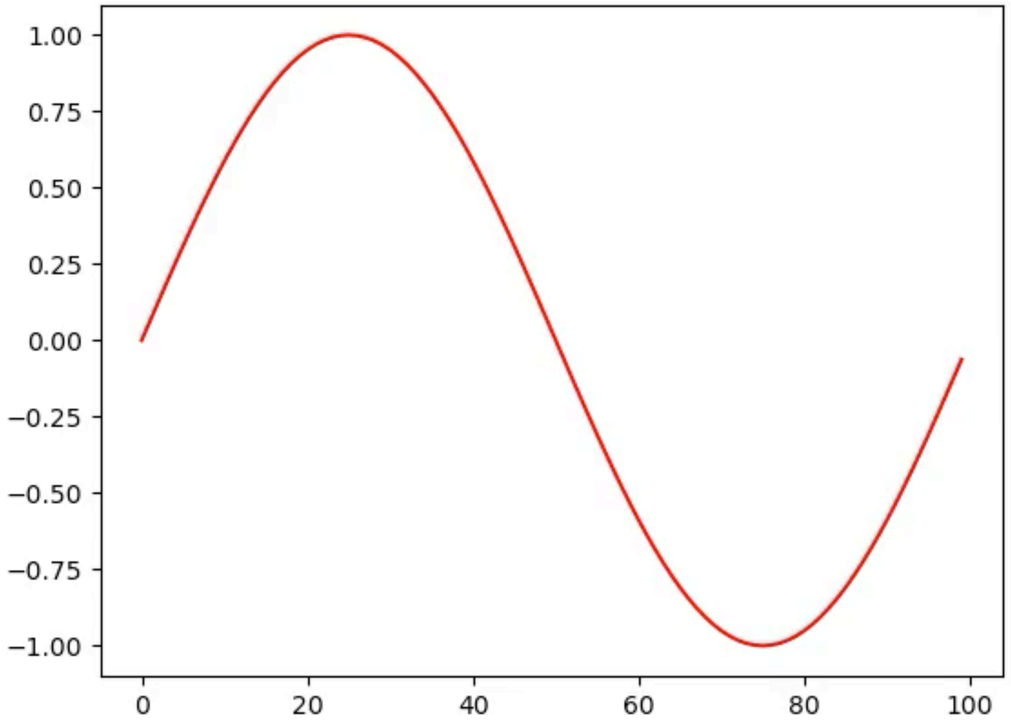




Advection: forward time, forward space

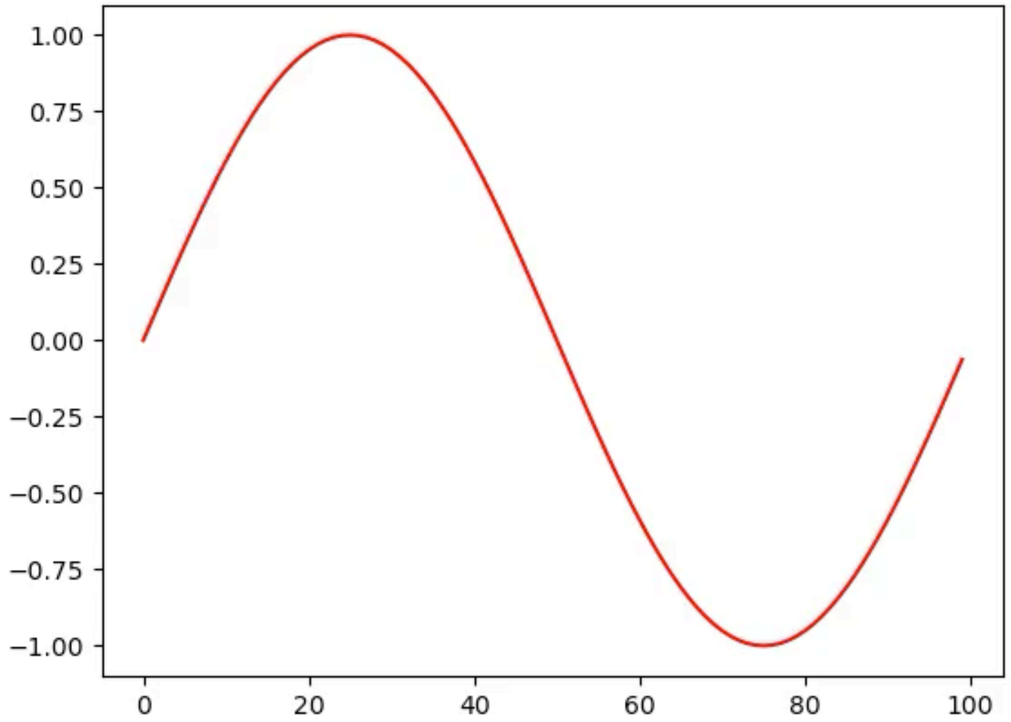
```
# Sine wave ICs
u[0,:] = np.sin(igrid*2*np.pi/ngrid)
```

```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - C*(u[n-1,i_upper] - u[n-1,:])
```



$dt = 1 \implies C = 1$

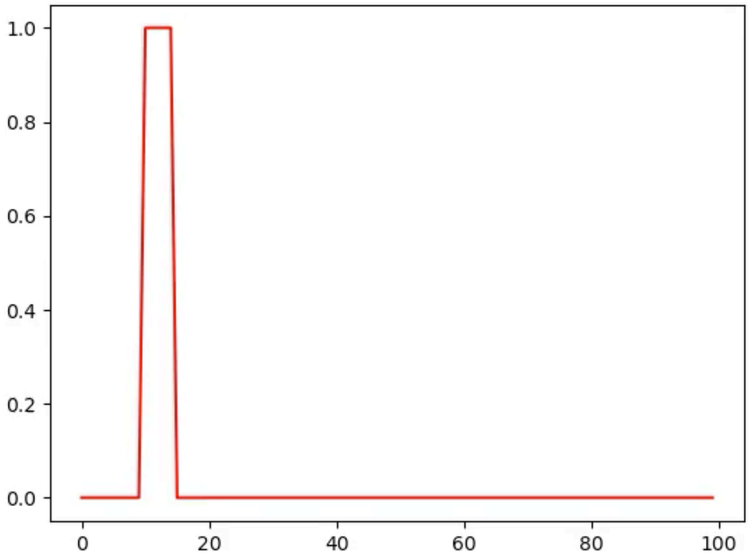
10x frame rate

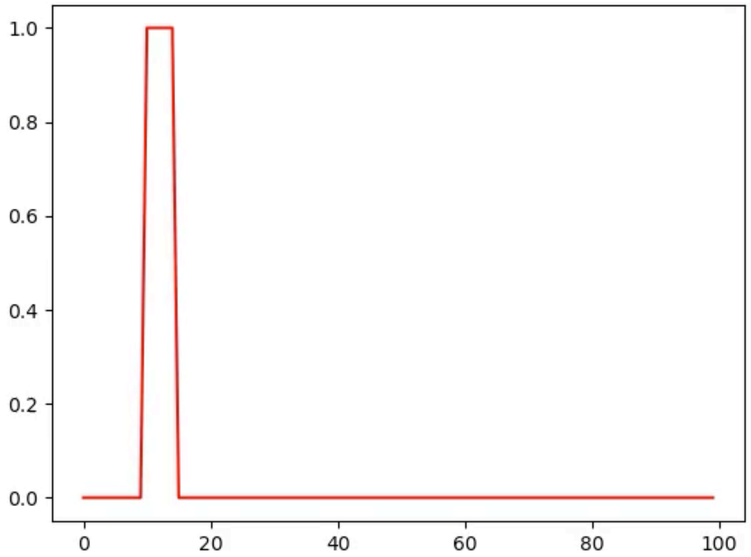


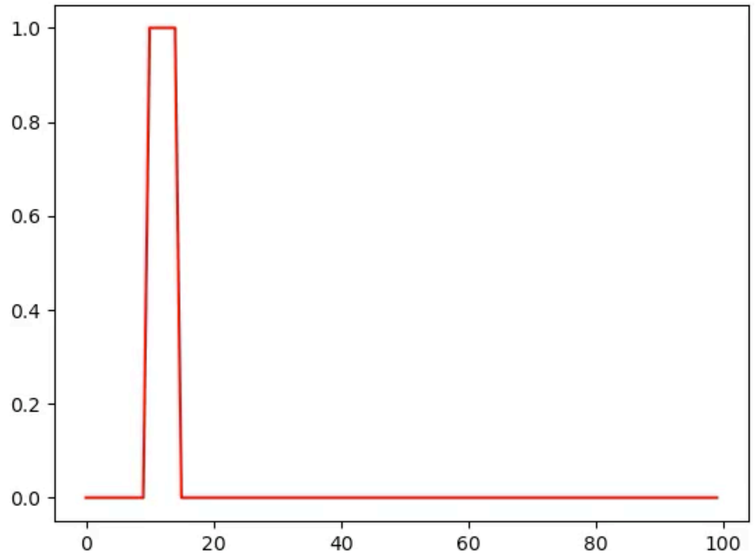
$dt = 0.1 \implies C = 0.1$

Advection: forward time, backwards space

```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - C*(u[n-1,:] - u[n-1,i_lower])
```







$dt = 0.1 \implies C = 10$

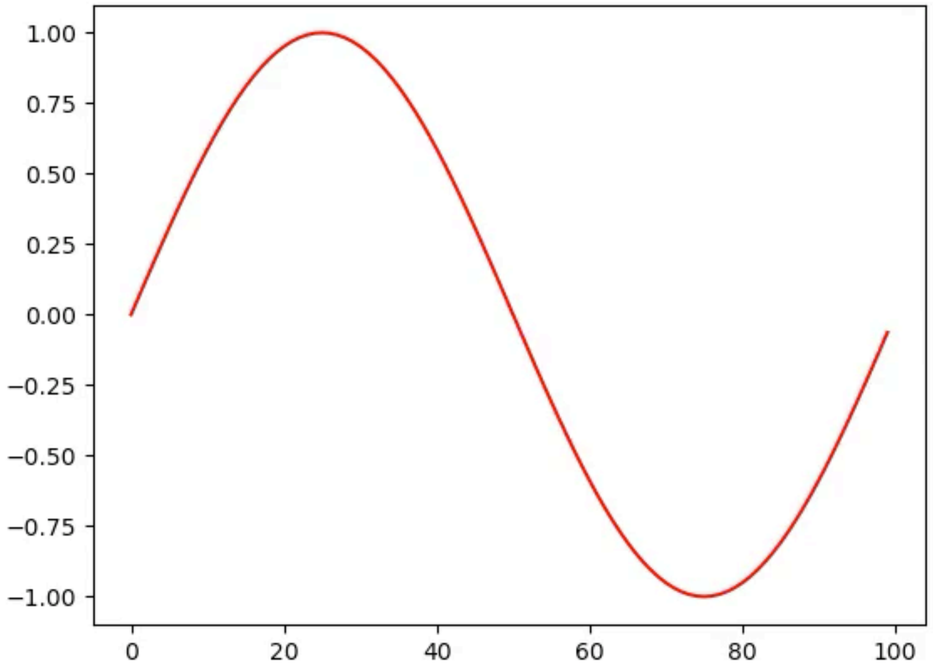
10x frame rate

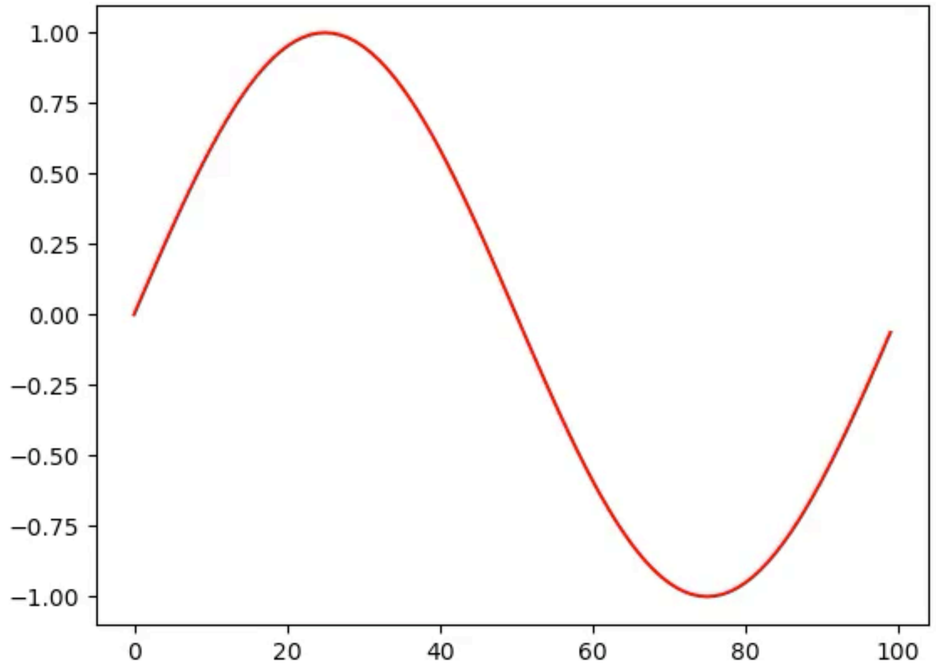
$dt = 1 \implies C = 1$

$dt = 2 \implies C = 2$

Advection: forward time, backwards space

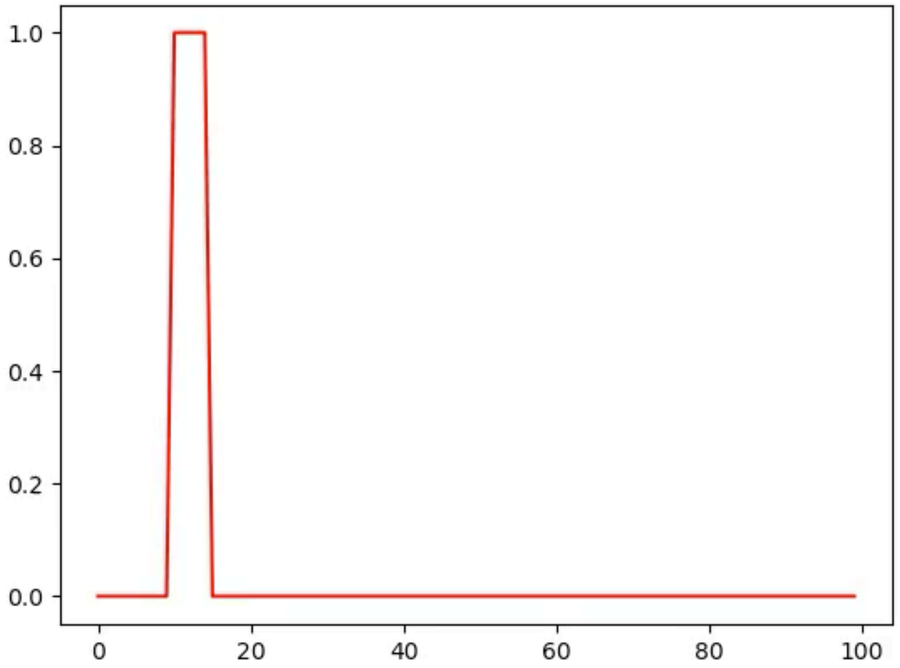
```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - C*(u[n-1,:] - u[n-1,i_lower])
```

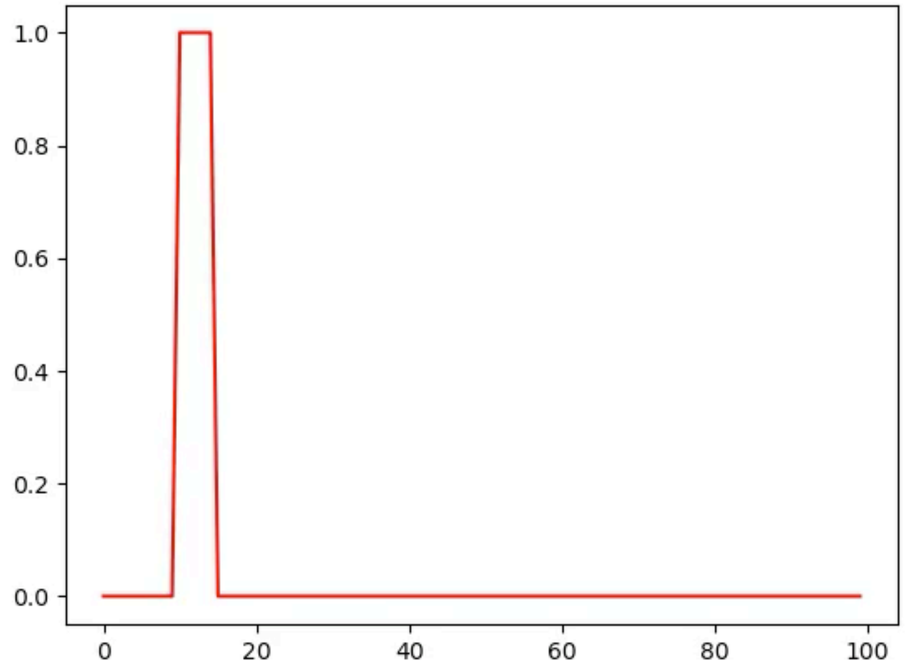




Centered difference in space

```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - (C/2)*(u[n-1,i_upper] - u[n-1,i_lower])
```





$dt = 0.1 \implies C = 10$

$dt = 2 \implies C = 1$

The Courant Condition

Properly, the ***Courant, Friedrichs & Lewy*** *(CFL) condition.*

A necessary but not sufficient requirement for a finite difference advection solver to be stable is:

$$\Delta t \leq \frac{\Delta x}{a} \qquad \text{The CFL condition}$$

Here a is the maximum speed at which information can propagate across the domain. We can see this is equivalent to a restriction on the *Courant* number,

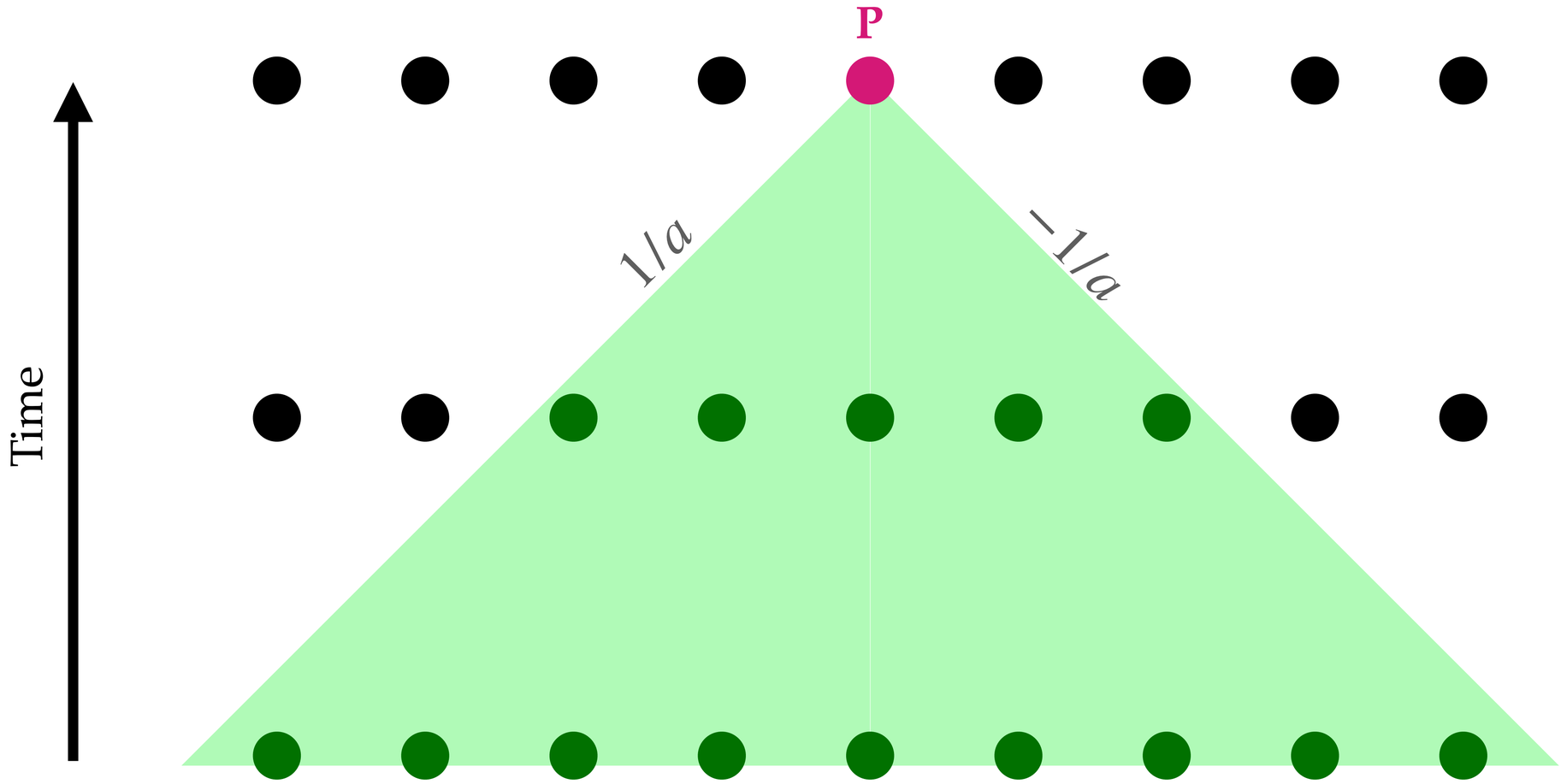
$$C = \frac{a}{\Delta x / \Delta t} \leq 1$$

Note: $\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2} = c \frac{\partial^2 u}{\partial x^2}$ (wave equation), so for the advection equation, $a \equiv \sqrt{c}$.

Propagating information

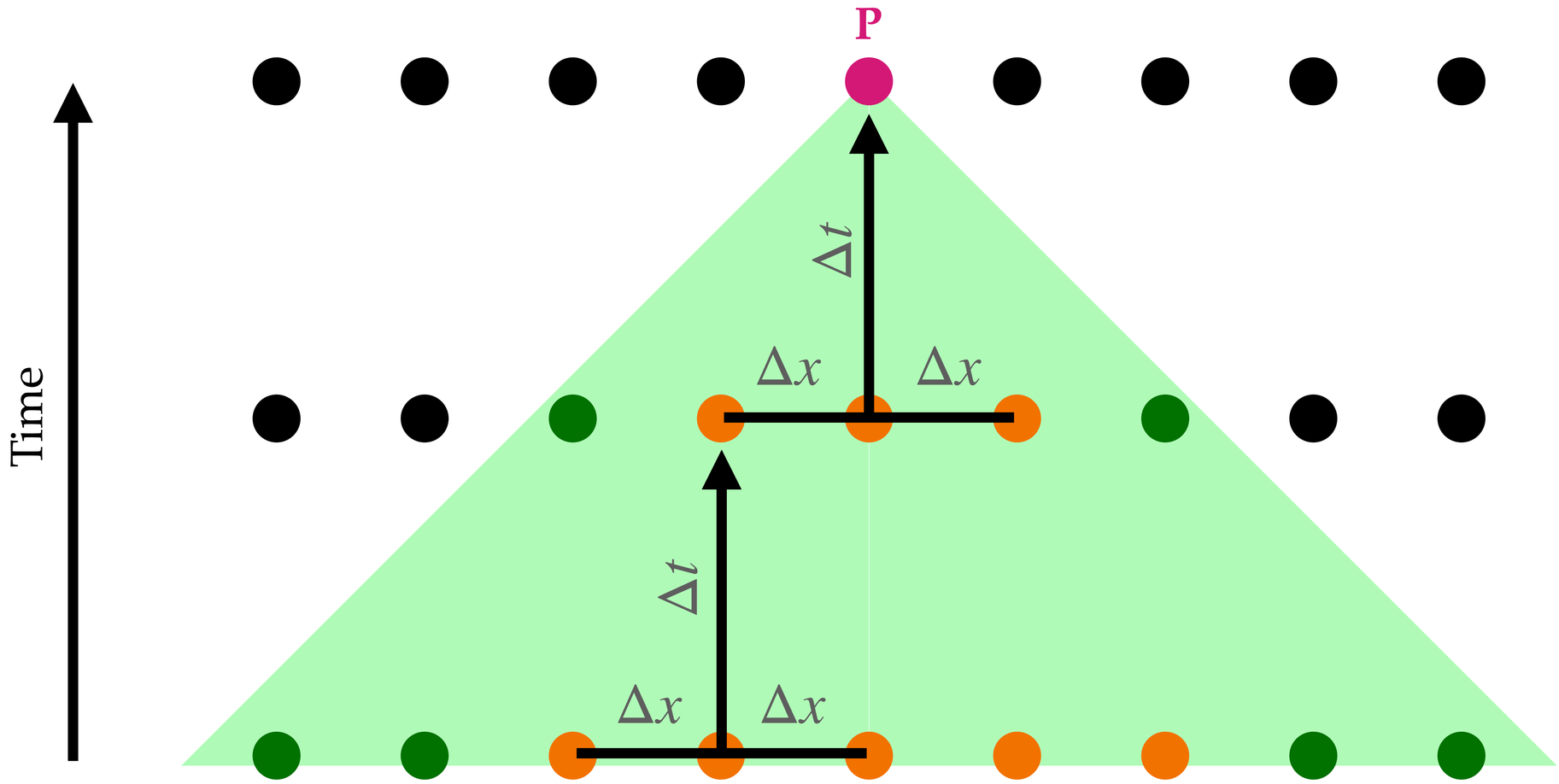
The green points are within the backwards lightcone of **P** (also know as the *domain of dependence*).

A **causal relationship** exists between the state at those points and the state at **P**.



Propagating information

The orange points are used to calculate the state at **P** by central finite differences.



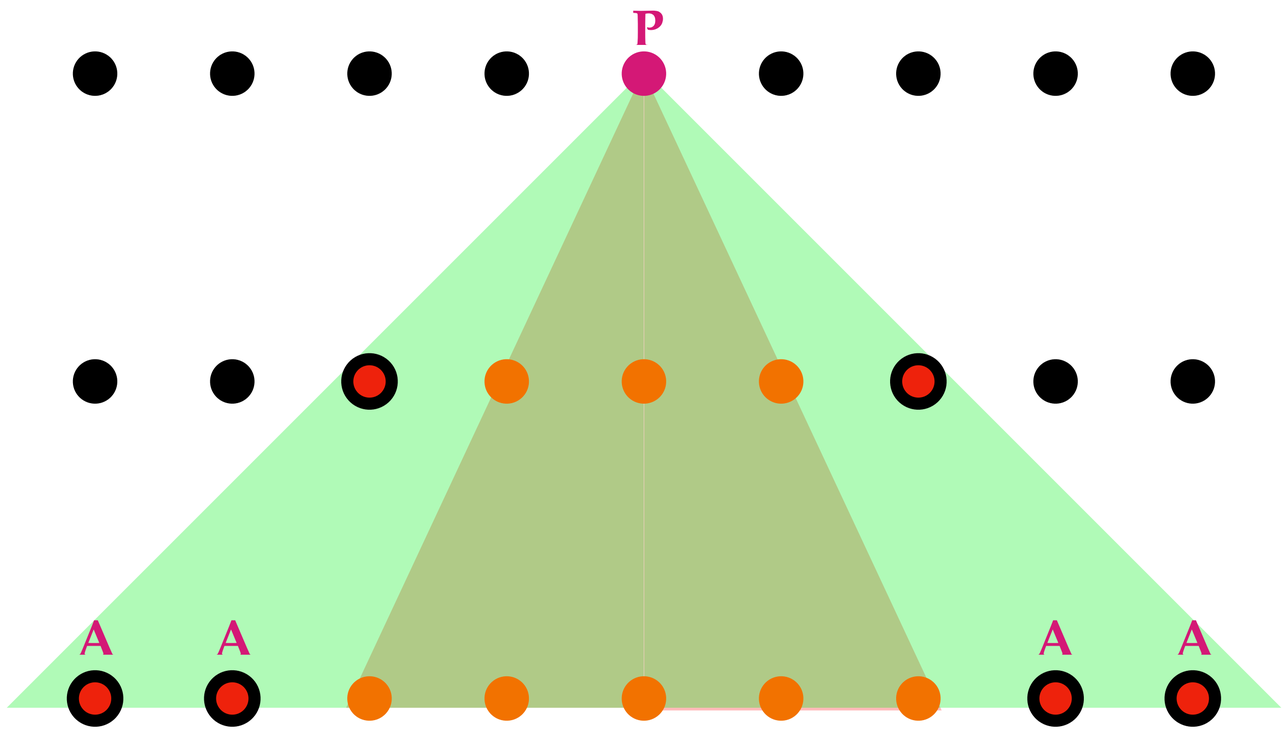
Propagating information

The red points are inside the *physical* domain of dependence, but not the *numerical* domain of dependence.

P *should* know about the initial conditions at **A**, but it doesn't: the FD scheme is too "slow"

$\Longrightarrow$ **Instability.**

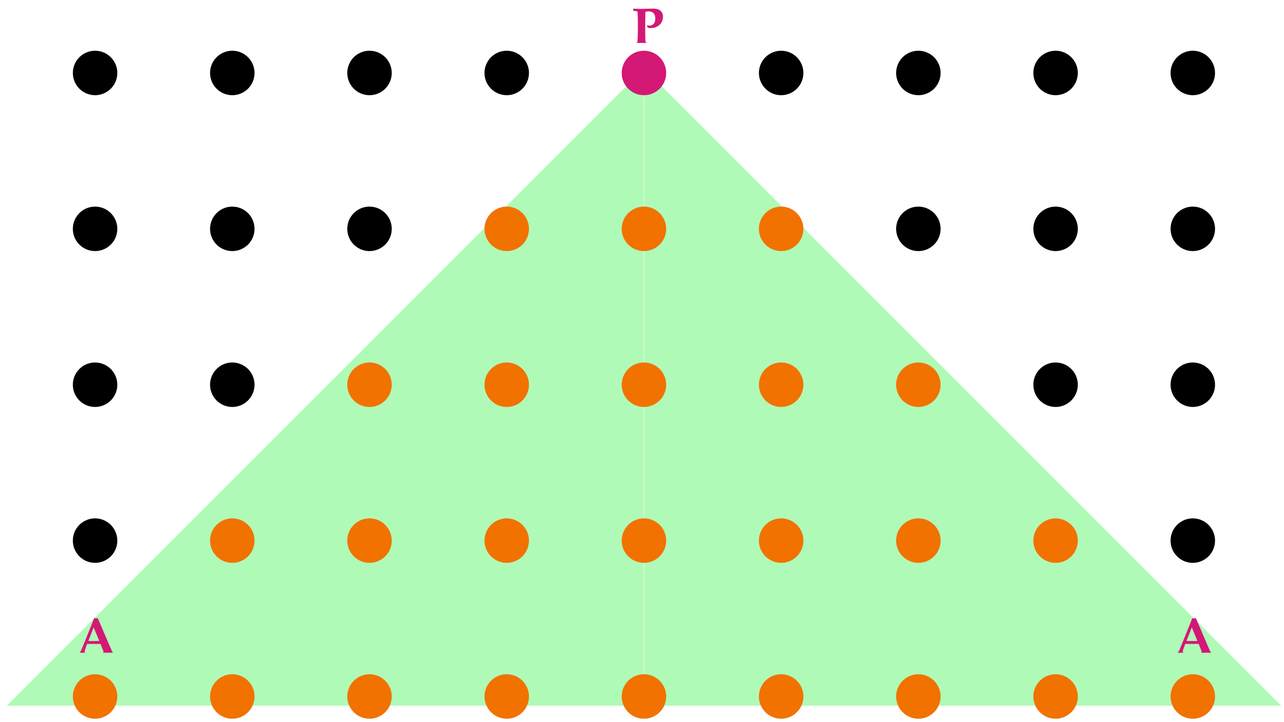
$$C = \frac{a}{\Delta x / \Delta t} > 1$$



Propagating information

By doubling the number of timesteps over the same interval, we match the two domains of dependence.

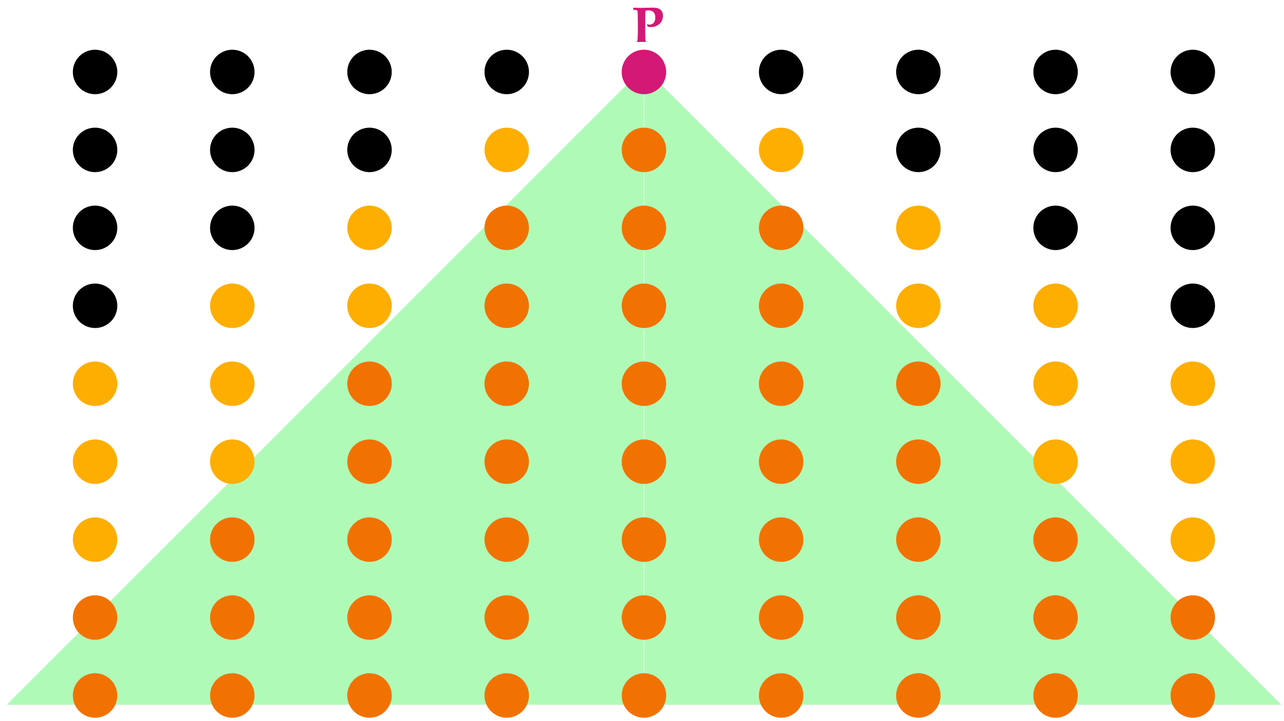
$$C = \frac{a}{\Delta x / \Delta t} = 1$$



Propagating information

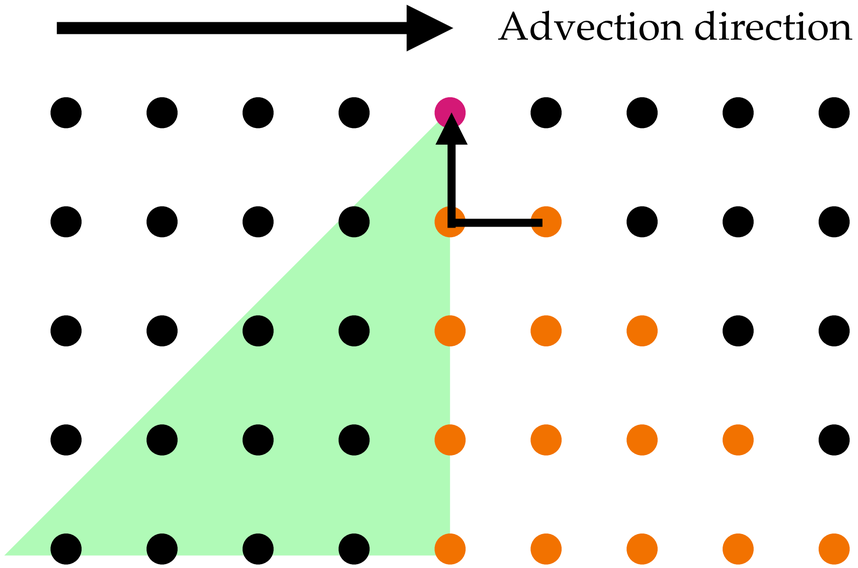
Halve the timestep again. No problem to have the FD scheme include points outside the physical domain: the physical speed sets the limit.

$$C = \frac{a}{\Delta x / \Delta t} < 1$$



Forwards and backwards differences

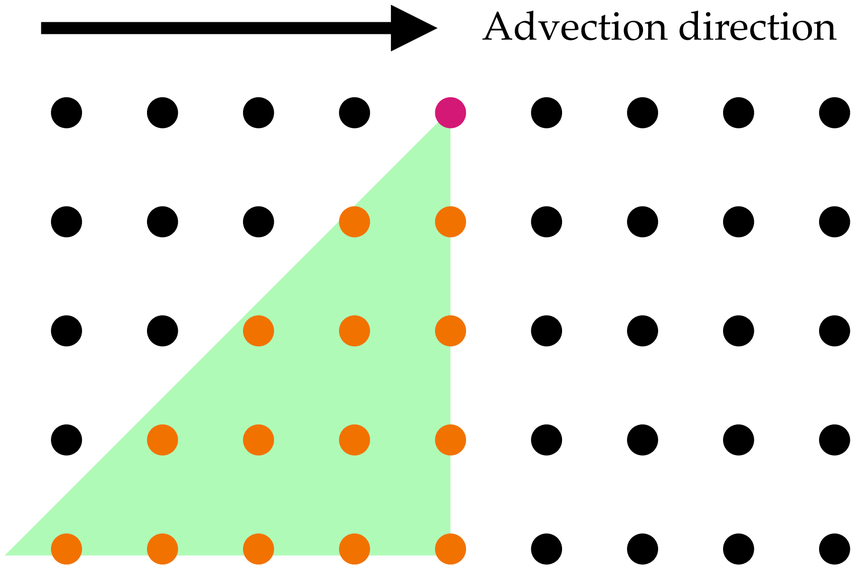
Forward difference



Information is propagated from 'downwind' of the front.

Unstable, regardless of C.

Backward difference



Information is propagated from 'upwind' of the front. **Stable** for $C \leq 1$.

Aside: stability to numerical errors

The **Von Neumann** method of stability analysis proceeds as follows. Consider the FTCS finite difference equation: $u_j^{n+1} = u_j^n - \frac{C}{2}\left(u_{j+1}^n - u_{j-1}^n\right)$ (j: space grid)

The error between the answer we actually get with finite precision, N_j^n, and the solution we would get if we had infinite numerical precision (no roundoff error) is $\epsilon_j^n = N_j^n - u_j^n$.

Since the PDE is linear, the error defined in this way is also a solution of the finite difference equation.

Represent the error as a Fourier series; each mode, m, is $\epsilon_j^n = f(t)g(x) = \xi_m(t)\ \exp\left(ik_m x\right)$

We get away with only studying one mode because they're not coupled, and we sweep under the carpet the issue of whether the system is periodic or not. See elsewhere for rigour!

Aside: stability to numerical errors

Substituting $\epsilon_j^n = f(t)g(x) = \xi_m(t)\ \exp\left(ik_m x\right)$ into the finite difference equation, we get:

$$\xi_m(t+\Delta t)\, e^{ik_m x} = \xi_m(t)\, e^{ik_m x} - \frac{C}{2}\left[\xi_m(t) e^{ik_m(x+\Delta x)} - \xi_m(t) e^{ik_m(x-\Delta x)}\right]$$

$$\frac{\xi_m(t+\Delta t)}{\xi_m(t)} = 1 - \frac{C}{2}\left(e^{ik_m \Delta x} - e^{-ik_m \Delta x}\right) = 1 - iC \sin k_m \Delta x$$

$$\text{Squaring: } \left|\frac{\xi_m(t+\Delta t)}{\xi_m(t)}\right|^2 = 1 + C^2 \sin^2(k_m \Delta x)$$

This says the amplitude of the perturbation will grow for all nonzero wavenumbers k_m; the method is unstable to perturbations introduced by numerical errors, whatever the value of C.

Applied to the FTBS scheme, this gives $0 < C \leq 1$ as the condition for stability, as expected.

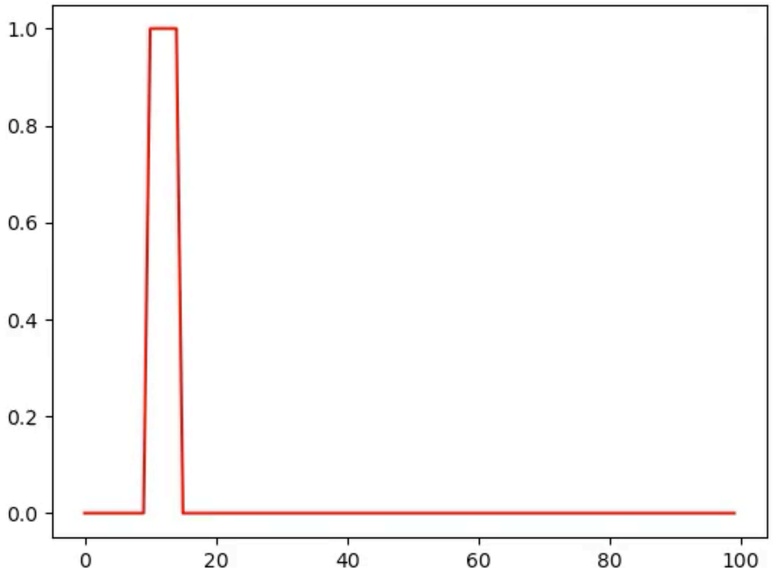
Upwind, downwind

Centered difference schemes are accurate for finite differencing in general, but not good for advection problems!

They mix in information from ahead (“downwind”) of the wavefront.

Backward differences (relative to the direction of propagation) are more stable because they only use information from “**upwind**” of the wavefront.

However, even when we satisfy the Courant condition, we don’t recover the analytic solution (except for the special case $C = 1$). Why?



Numerical diffusion

The finite difference equations come from a truncated Taylor expansion. Specifically, going back to our earlier definitions (class 6, with $h \to \Delta x$ or Δt and keeping one higher order):

$$\frac{u_i^n - u_{i-1}^n}{\Delta x} = \frac{\partial u}{\partial x} - \frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2} + O\left(\Delta x^2\right)$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\partial u}{\partial t} + \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + O\left(\Delta t^2\right)$$

Numerical diffusion

The **true** solution PDE for these truncated versions is not the same as the solution to the original PDE. Writing the advection equation in terms of these truncated series instead of the (equivalent) finite differences gives:

$$\frac{\partial u}{\partial t} + \frac{\Delta x}{2}\frac{\partial^2 u}{\partial t^2} + O\left(\Delta t^2\right) = -a\left[\frac{\partial u}{\partial x} - \frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2} + O\left(\Delta x^2\right)\right]$$

Rearranging: $$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = -\left[\frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + a\frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2}\right] + \ldots$$

$$\frac{d^2u}{dt^2} = a^2\frac{d^2u}{dx^2} \qquad \therefore \quad \frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = -\frac{a}{2}\left[a\Delta t - \Delta x\right]\frac{d^2u}{dx^2} = \frac{a\Delta x}{2}[1 - C]\frac{d^2u}{dx^2}$$

Numerical diffusion

Thus the actual PDE solved by a (stable) first-order backward finite difference approximation of the advection equation is:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = \frac{a\Delta x}{2}[1 - C]\frac{d^2u}{dx^2}$$

When $C = 1$, this reduces to the original advection equation. That matches what we saw for the backwards difference method with $C = 1$.

When $C \ll 1$, the equation is $\frac{\partial u}{\partial t} \approx -a\frac{\partial u}{\partial x} + \frac{a\Delta x}{2}\frac{d^2u}{dx^2}$.

This is the so-called **advection-diffusion equation.**

Numerical diffusion

The pure diffusion equation (the heat equation) is $\frac{\partial u}{\partial t} = a\frac{d^2u}{dx^2}$.

The solution is non-trivial (exponentially growing modes, decaying modes or oscillations, depending on the boundary conditions).

The forward difference method corresponds to a sign change, such that the diffusion term is $\propto [C - 1]$. Any choice that satisfies the Courant condition will give a negative RHS. This is inherently unstable.

Artificial Viscosity

By adding a **artificial viscosity** term (negative diffusion) to the numerical solution, we can counteract the diffusive effect in the original approximation.

For a central difference method:

$$u_i^{n+1} = u_i^n - \frac{a\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \frac{\beta}{2}\left[u_{i+1}^n - 2u_i^n + u_{i-1}^n\right]$$

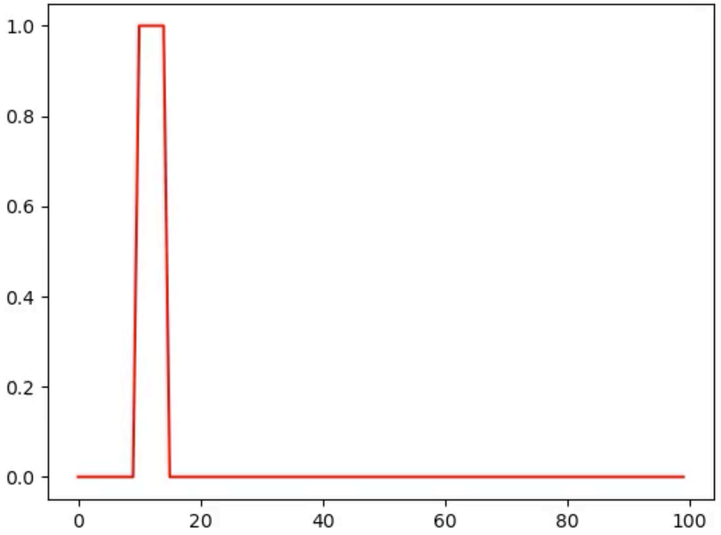
$$\beta = 1\text{: “Lax-Friedrichs”;}\quad \beta = \frac{a^2(\Delta t)^2}{(\Delta x)^2}\text{ : “Lax-Wendroff”}$$

Top-hat with artificial viscosity

Central Difference

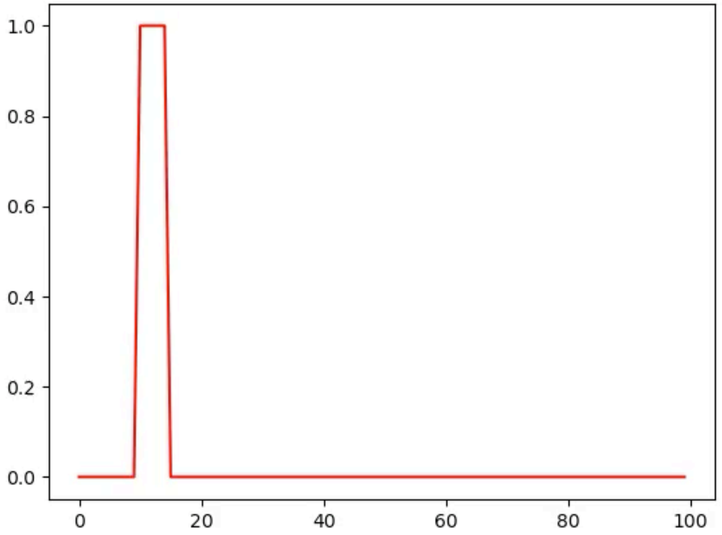
Artificial Viscosity

```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - (C/2)*(u[n-1,i_upper] - u[n-1,i_lower]) + (beta/2)*(u[n-1,i_upper] - 2*u[n-1,:] + u[n-1,i_lower])
```



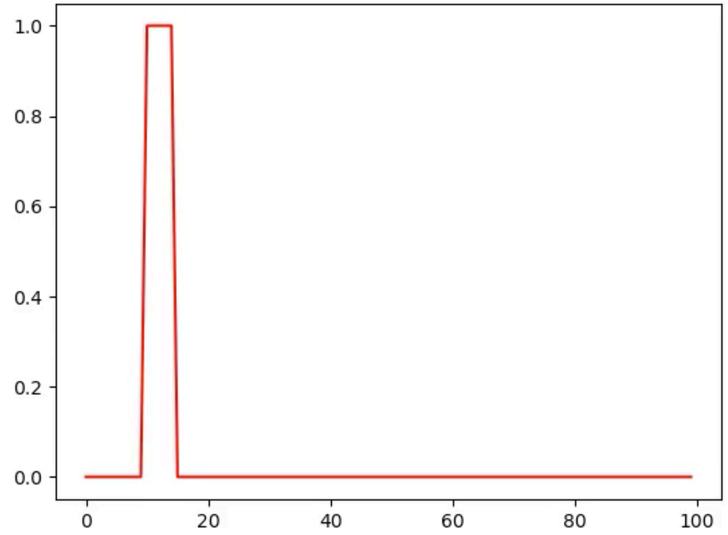
$C = 1, \beta = 1$

Lax-Friedrichs



$C = 0.5, \beta = 1$

Lax-Friedrichs



$C = 0.5, \beta = 0.25$

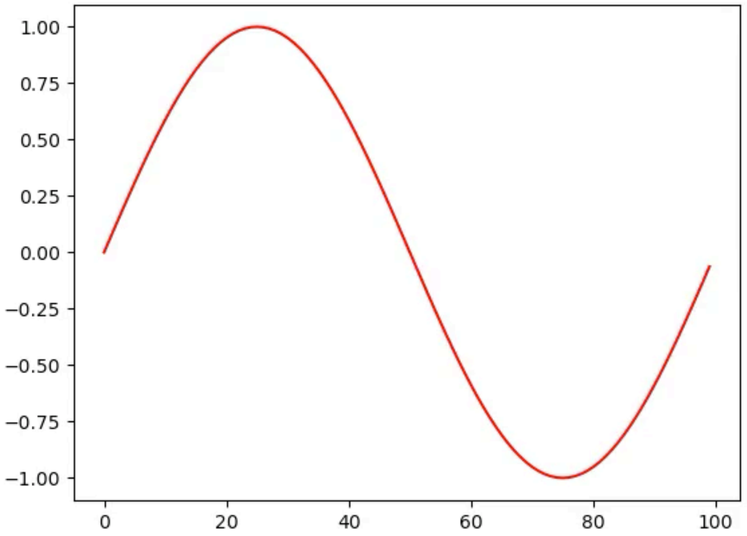
Lax-Wendroff ($\beta = C^2$)

Sine wave with artificial viscosity

Central Difference

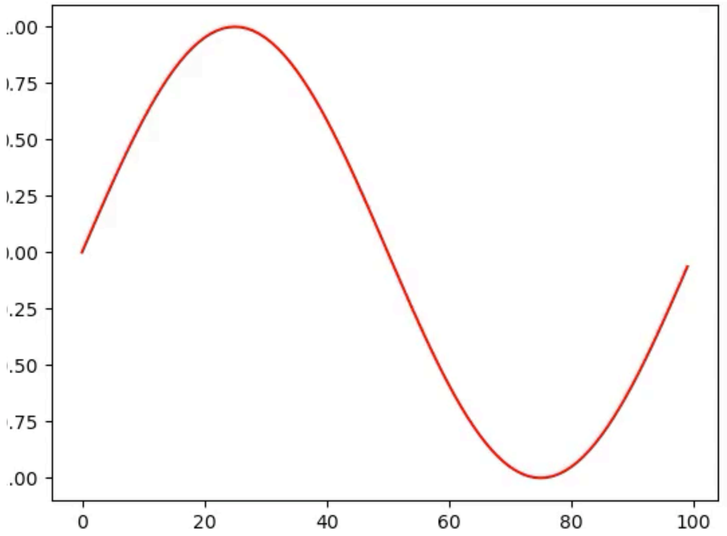
Artificial Viscosity

```
for n in range(1,nsteps):
    u[n,:] = u[n-1] - (C/2)*(u[n-1,i_upper] - u[n-1,i_lower]) + (beta/2)*(u[n-1,i_upper] - 2*u[n-1,:] + u[n-1,i_lower])
```



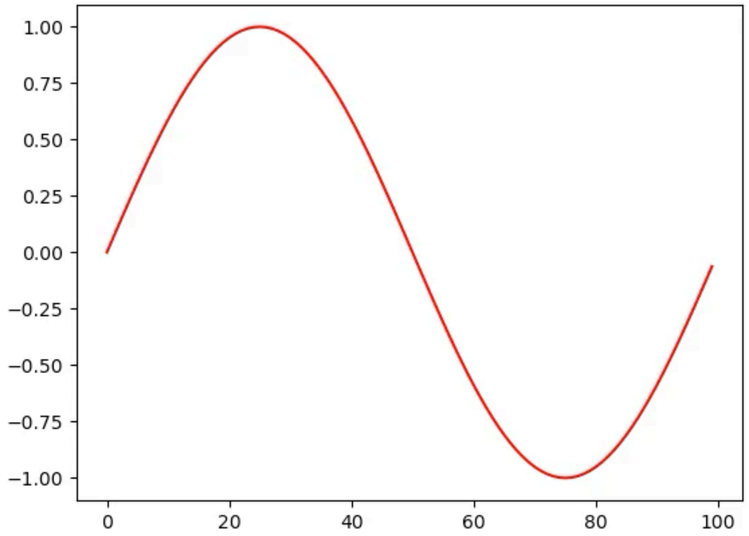
$C = 1, \beta = 1$

Lax-Friedrichs



$C = 0.5, \beta = 1$

Lax-Friedrichs



$C = 0.5, \beta = 0.25$

Lax-Wendroff ($\beta = C^2$)

Summary so far

Most of the above considerations of the advection equation are rather special cases. The specifics don't apply to PDEs in general.

However, they illustrate a lot of the main concerns of solving PDEs with finite difference methods:

- Stability (numerical errors grow catastrophically);
- Courant condition/ choice of Courant number — these differ from problem to problem, but the idea is the same: the timestep is constrained by the grid spacing;
- Addition of artificial viscosity to counter diffusion due to approximation;
- Directional differences / discontinuities.

Implicit finite difference methods

It is also possible to create finite difference schemes in which the time step solution is **implicit**.

As for the N-body problem, implicit methods have larger regions of stability, but need to be solved as a linear algebra problem for each step.

However, there is a much better general idea: the **finite volume** approach.

The finite volume approach is the basis of most practical, general mesh codes for hydrodynamics. We will look at it briefly in a moment, and in more detail next week.

Photo: 隆大爺

Taiwan Cupwing
臺灣鷦眉 (小滷蛋)

ASTR 660 / Class 11

Partial differential equations

The zoo of PDEs

We will review of the jargon associated with classifying different types of PDE and their boundary conditions.

2nd-order linear PDEs belong to one of these three classes:

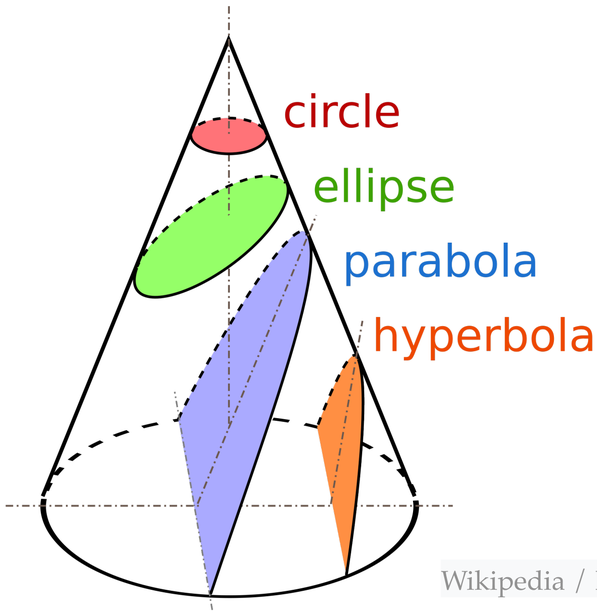
- **Hyperbolic**
- **Parabolic**
- **Elliptic**

Origin of the conic jargon

Writing d^2u/dx^2 as u_{xx}, $d^2u/dxdy$ as u_{xy} etc., any 2nd order linear PDE can be rearranged to look like $au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$.

By analogy with quadratic equations, the discriminant $b^2 - 4ac$ separates different classes of solution (hence language of "conic sections"):

- **Hyperbolic:** $b^2 - 4ac > 0$
- **Parabolic:** $b^2 - 4ac = 0$
- **Elliptic:** $b^2 - 4ac < 0$

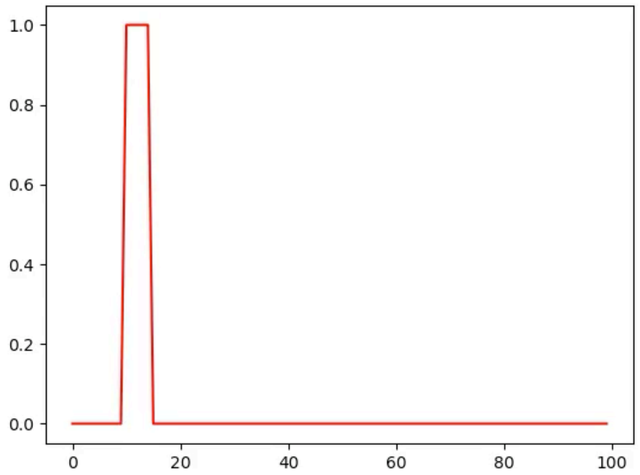


Wikipedia / Magister_Mathematicae

Hyperbolic PDEs

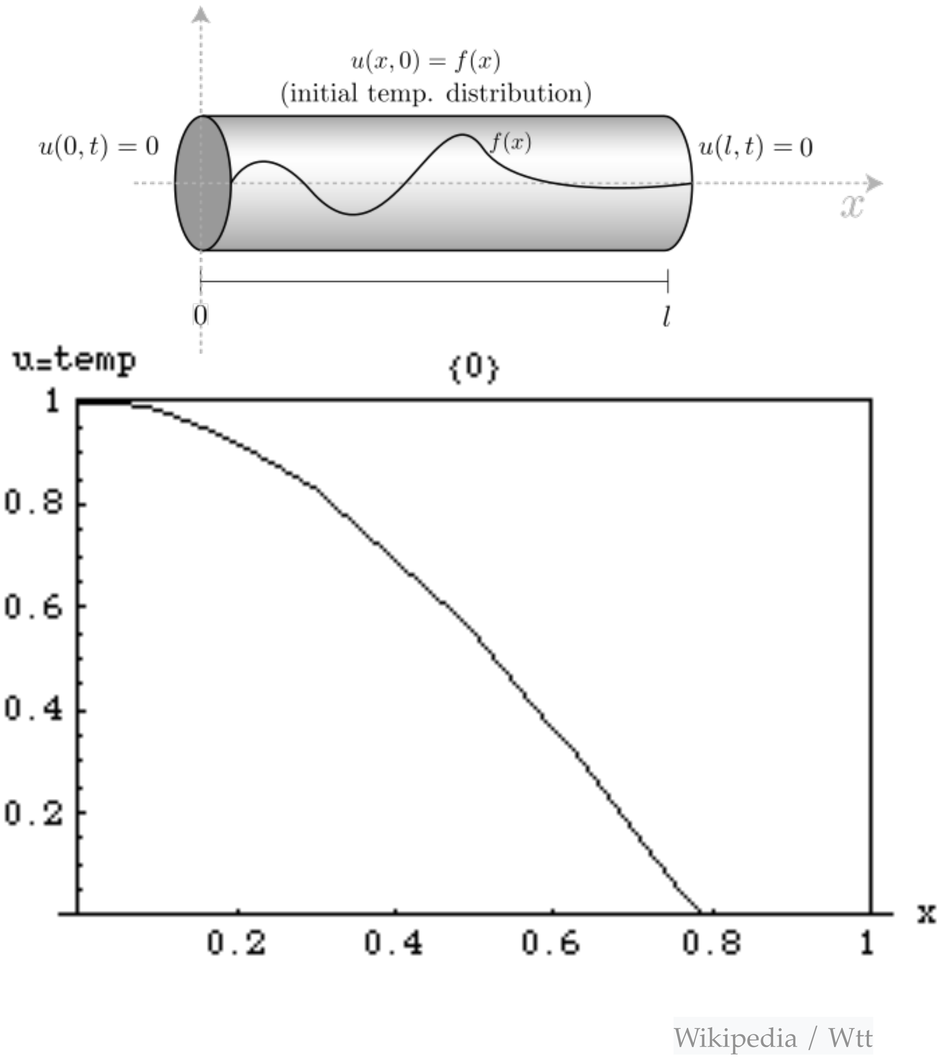
Examples: $u_{tt} = u_{xx}$ (wave equation); $u_t = au_x$ (advection equation)

- Time dependent.
- Not evolving towards equilibrium.
- Conserve energy.
- Time-reversible.
- Discontinues preserved / grow / move around (e.g. convection).
- Initial conditions remembered.
- Information moves at finite speed.
- Initial conditions and boundary conditions are needed.



Parabolic PDEs

Example: $u_t = u_{xx}$ (heat equation)



Wikipedia / Wtt

- Time dependent.
- Evolving towards equilibrium.
- Dissipating energy.
- Not time-reversible (diffusive).
- Discontinues smoothed out over time.
- Initial conditions erased.
- Information propagates instantly (not intuitive…).
- Initial conditions and boundary conditions are needed.

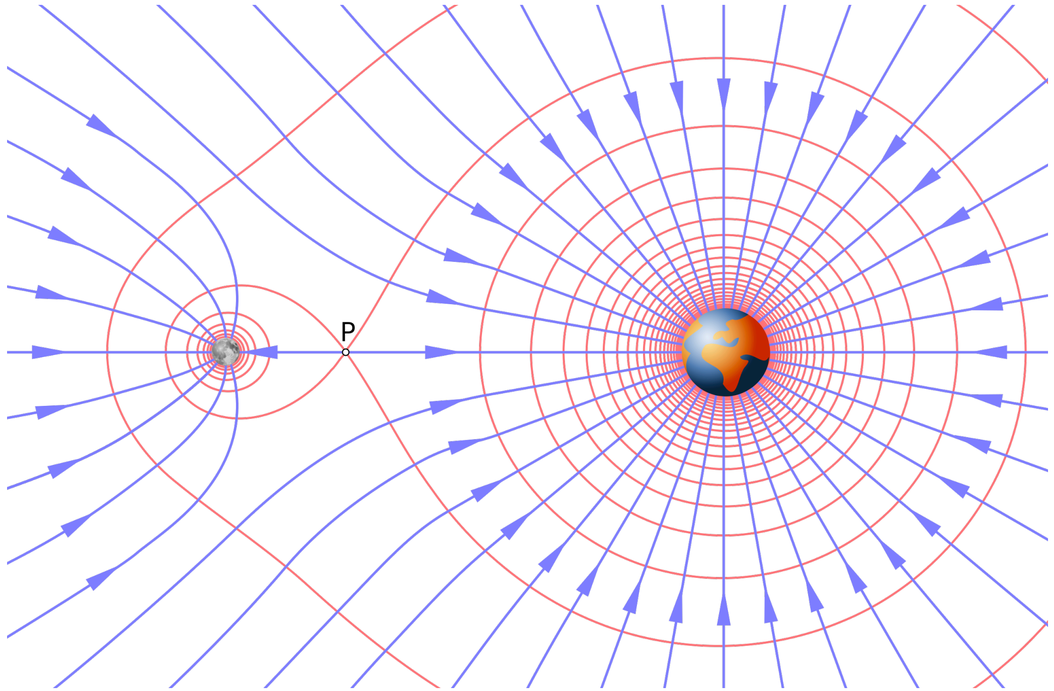
Elliptic PDEs

Example: $u_{xx} + u_{yy} = 0$ (Laplace equation)

- Not time-dependent.
- In equilibrium.
- Boundary value problem only.

Bonus jargon for boundary conditions:

- **Dirichlet**: solution is fixed on the boundary.
- **Neumann**: derivatives are fixed on the boundary.
- Mixed: some combination of the above

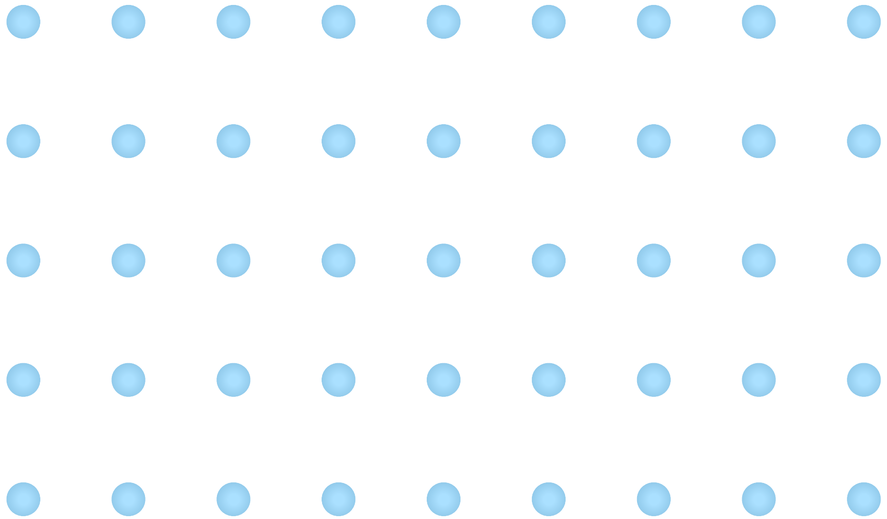


Wikipedia / MikeRun

Finite Difference Methods

We only solve the equations exactly at the “gridpoints”.

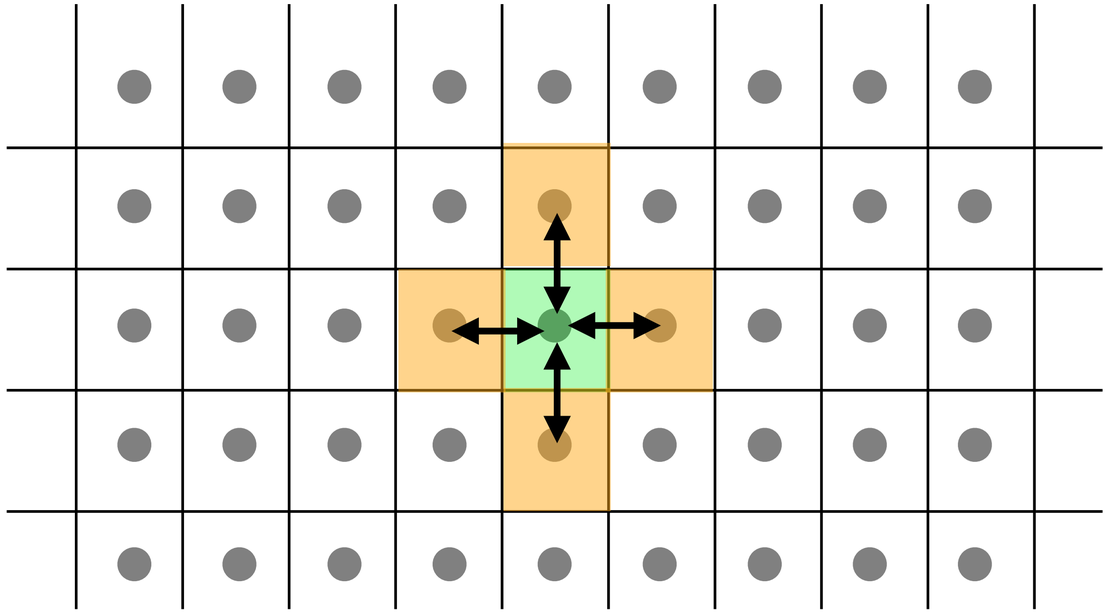
If we want information between the gridpoints, we can interpolate.



Finite Volume Methods

Each “gridpoint” is the midpoint of a **cell** around it (area in 2-d, volume in 3-d).

Rather the calculating properties only at the gridpoints, we track totals (or averages) of quantities, e.g. mass (density) over cells. We solve **conservative** equations for **fluxes** into and out of the cells.



Challenges

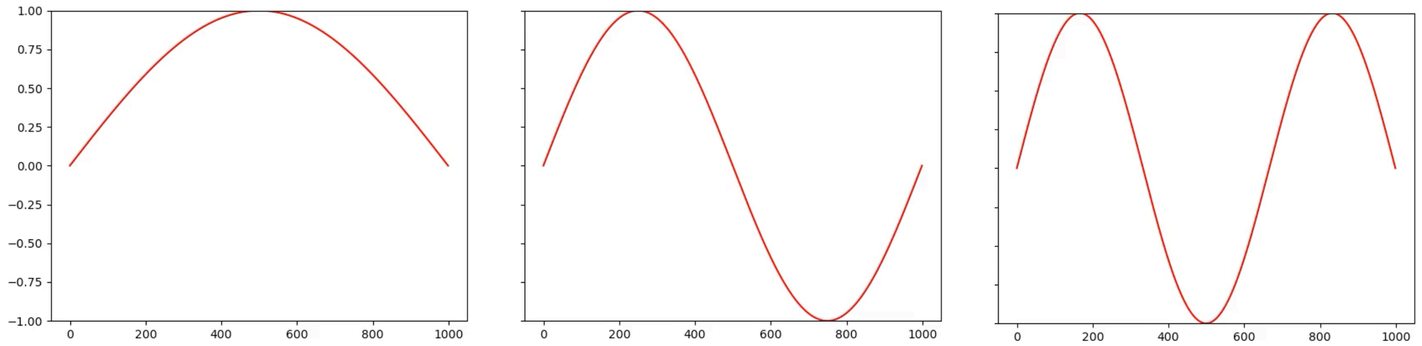
To practice the idea of finite difference solutions, try to make some of the advection movie plots. Code to make the animation is on the Github (`examples/advection`).

Then try to adapt the idea to solve the wave equation, $u_{tt} = c^2 u_{xx}$, using central differences in both space and time.

You will need to think carefully about the boundary conditions. Consider the approach of introducing special "boundary points" (typically called "ghost cells") where the solution is determined "by hand".

You will also need to think about the initial conditions in the case where more than one previous time step is needed for a solution. How do you "start" the calculation?

Try wave modes in a box:



Sources for today's slides

These slides build on and adapt previous lecture notes for this course, by Kuo-Chuan Pan, Karen Yang, and (possibly!) Hsi-Yu Schive.

I also made use of Wikipedia (von Neumann analysis), hydrodynamics notes by Diemer and Zigale (see our wiki for links), and the textbook "Scientific Computing – An Introductory Survey" by Heath (2008).