Partial Differential Equations – Hyperbolic Systems

Lecture 10, Computational Astrophysics (ASTR660)

Hsiang-Yi Karen Yang, NTHU, 11/17/2022

Announcements

- *HW4* will be posted on eLearn TODAY. *Due at 14:20 on 11/24 (Thu)*. Late submission within one week will receive *75%* of the credit
- Please help to provide comments/feedback to improve this course! The questionnaire can be found via this link or by scanning this QR code:

Previous lecture – boundary value problems (BVPs)

$$y^{(k)}(t) = f(t, y, y', \ldots, y^{(k-1)})$$

- *Boundary value problems (BVPs)* – obtaining unique solutions for ODEs when the boundary values, not the initial values, are provided
- Algorithms for BVPs:
  - *Shooting method* – starting from initial guess of $y'(a)$, vary $y'(a)$ until $y(b) = \beta$. Can be combined with bisection method to make it more efficient.
  - *Finite difference method* – introduce mesh points t_i between a and b, replace derivatives with finite difference approximations, and then solve a system of equations for $y(t_i)$

This lecture…

- Introduction to partial differential equations (PDEs)
- Finite difference methods & exercise
- Finite volume methods & exercise

Intro to PDEs

Partial differential equations (PDEs)

- *Partial differential equations (PDEs)* involve partial derivatives w.r.t more than one independent variable
- Independent variables typically include
  - One or more *spatial* dimensions
  - The *time* dimension as well
- Problem is more complex and one can have IVPs and/or BVPs
- Equation and boundary data may be defined over multi-D, irregular domain

Examples of PDEs in nature

Many of the basic law of nature are expressed as PDEs:

- Maxwell's equations (EM fields)
- Navier-Stokes equations (fluids)
- Elasticity equations (vibrations in solid states)
- Schrodinger's equations (quantum physics)
- Einstein's equation of GR (space-time evolution)
- ...

Order of PDEs

- ***Order*** of PDEs – the highest-order particle derivative appearing in the equation
- To simply the equations, let's use ***subscripts*** to denote derivatives w.r.t independent variables
  - $u_t = \frac{\partial u}{\partial t}$
  - $u_{xy} = \frac{\partial^2 u}{\partial x \partial y}$
- Example of ***1st-order*** PDE -- advection equation: $u_t = -cu_x$
- Examples of ***2nd-order*** PDE
  - Heat/diffusion equation: $u_t = u_{xx}$
  - Wave equation: $u_{tt} = u_{xx}$
  - Laplace equation: $u_{xx} + u_{yy} = 0$

Example of 1st-order PDE – advection equation

a scalar to be transported

- Advection equation: $$\frac{\partial u}{\partial t} + \nabla \cdot (u\boldsymbol{c}) = 0$$

speed of propagation (not speed of light!)

- In 1-D: $$u_t = -cu_x$$

Example of 1st-order PDE – advection equation

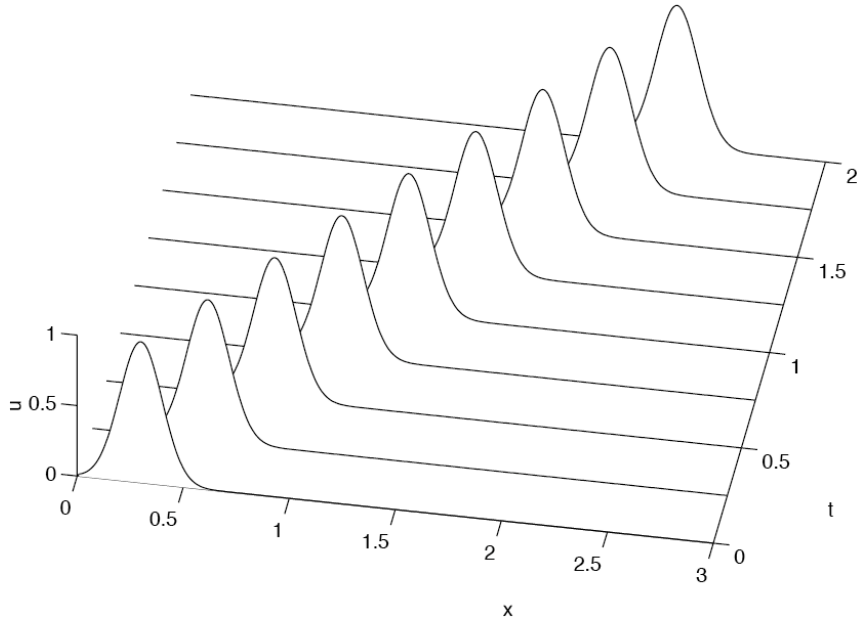
- Advection equation: $\frac{\partial u}{\partial t} + \nabla \cdot (u\boldsymbol{c}) = 0$

- In 1-D: $u_t = -cu_x$

Example of 1st-order PDE – advection equation

$$u_t = -cu_x$$

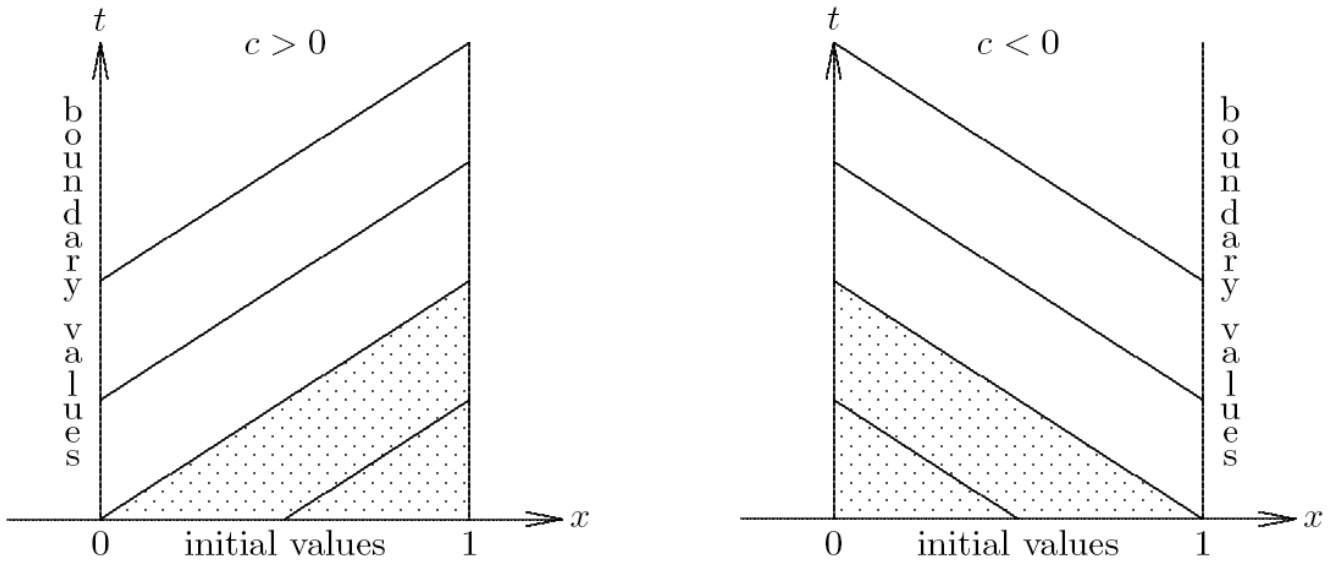
- Unique solution is determined by I.C.: $u(t = 0, x) = u_0(x)$
- Aim to find solution $u(t, x)$ for $t \geq 0$
- Solution is initial function $u_0(x)$ shifted by ct to the right if $c > 0$, or to the left if $c < 0$



Typical solution of advection equation, with the initial function *advected (shifted/平流)* over time

Characteristics

- ***Characteristics*** for PDEs are ***contours of solution***
- For advection equation $u_t = -cu_x$, characteristics are straight lines of slope c



- Characteristics determine where B.C. must be imposed

Types of 2nd-order PDEs

- 2nd-order linear PDEs have the general form:

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$$

- They are classified into three categories:
  - If $b^2 - 4ac > 0$: ***hyperbolic*** (e.g., wave equation, hydro equations)

$$\frac{\partial^2 \rho}{\partial t^2} - a^2 \nabla^2 \rho = 0$$

  - If $b^2 - 4ac = 0$: ***parabolic*** (e.g., heat equation)

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

  - If $b^2 - 4ac < 0$: ***elliptic*** (e.g., Laplace equation, Poisson equation)

$$\nabla^2 \phi = 4\pi G \rho$$

Types of 2nd-order PDEs

Properties of the PDEs are not so simply classified, but roughly speaking:

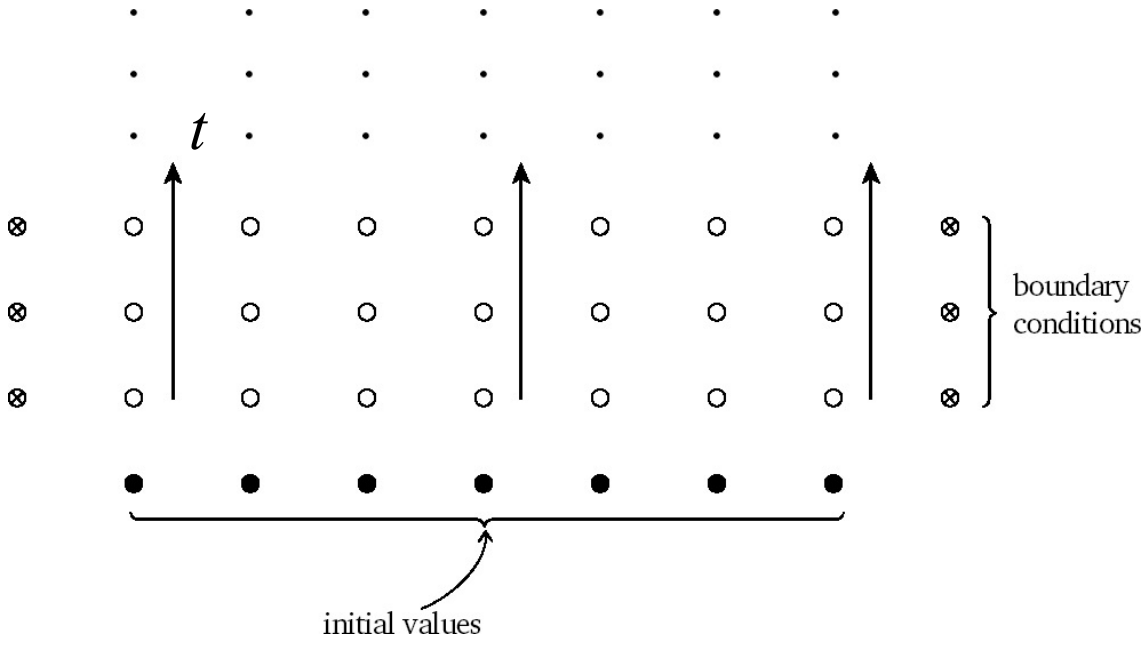
- **Hyperbolic** PDEs: *time-dependent* systems, *conservative* physical processes (e.g., advection, hydro), *not evolving toward steady state*
- **Parabolic** PDEs: *time-dependent* systems, *dissipative/damped* physical processes (e.g., diffusion), *are evolving toward steady state*
- **Elliptic** PDEs: *time-independent* systems (e.g., gravitational potential for a point mass), *already reached steady state*

Syllabus

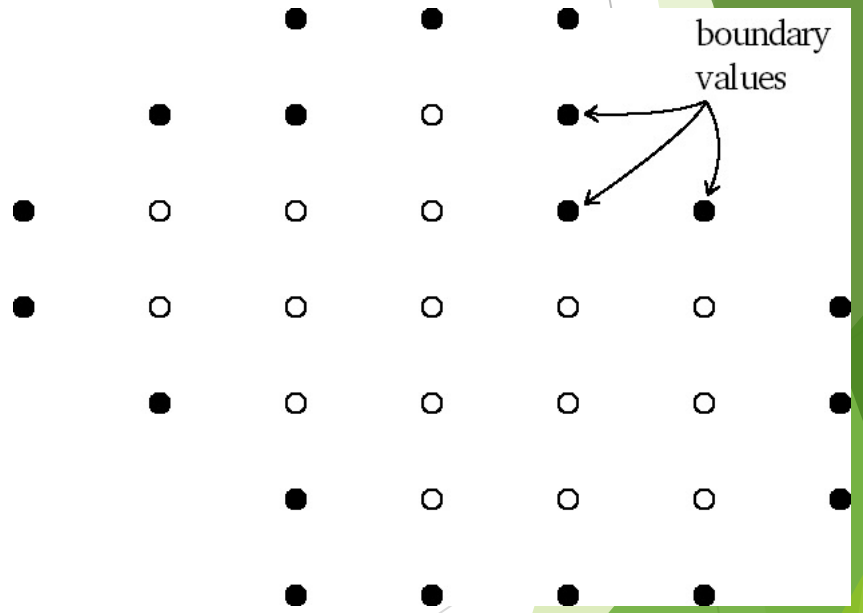
Week 10 (11/17) PDE: hyperbolic systems (advection equation)
Week 11 (11/24) PDE: astrophysics fluid dynamics I
Week 12 (12/1) PDE: astrophysics fluid dynamics & MHD II
Week 13 (12/8) PDE: elliptical systems (gravity)

Types of 2nd-order PDEs

- Time-dependent *parabolic* & *hyperbolic* equations are typically *IVPs + BVPs*



- Time-independent *elliptic* equations are typically *BVPs*

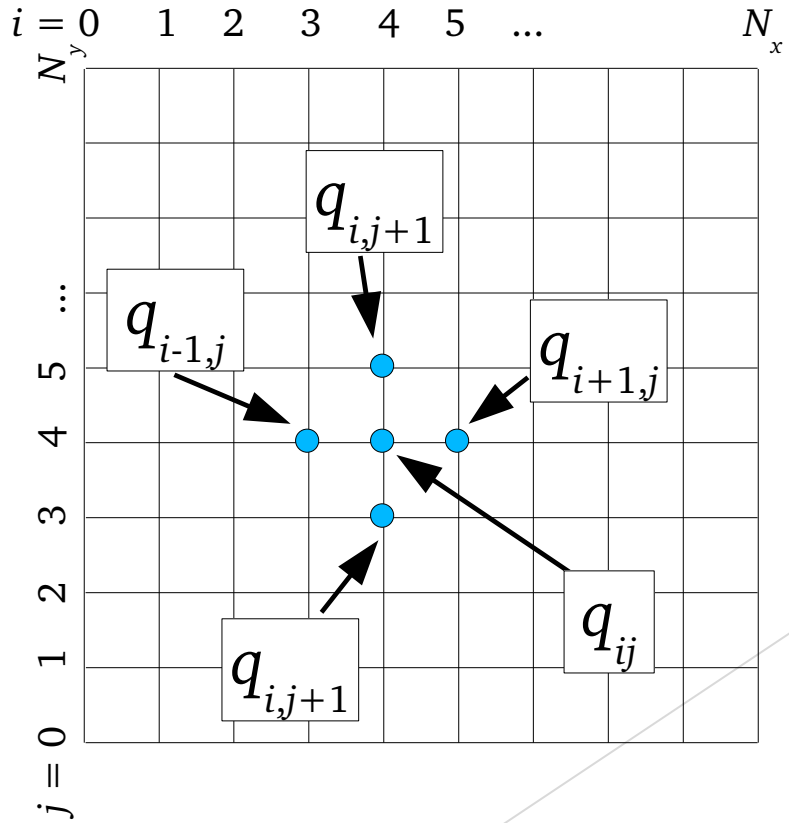


Discretization of PDEs

Method #1: Finite-difference methods

It works with solution values *at* a number of *specified points*:

$$q_{ij} \equiv q(x_i, y_j)$$
$$x_i = i\,\Delta x \qquad i = 0 \ldots N_x$$
$$y_j = j\,\Delta y \qquad j = 0 \ldots N_y$$



Discretization of PDEs

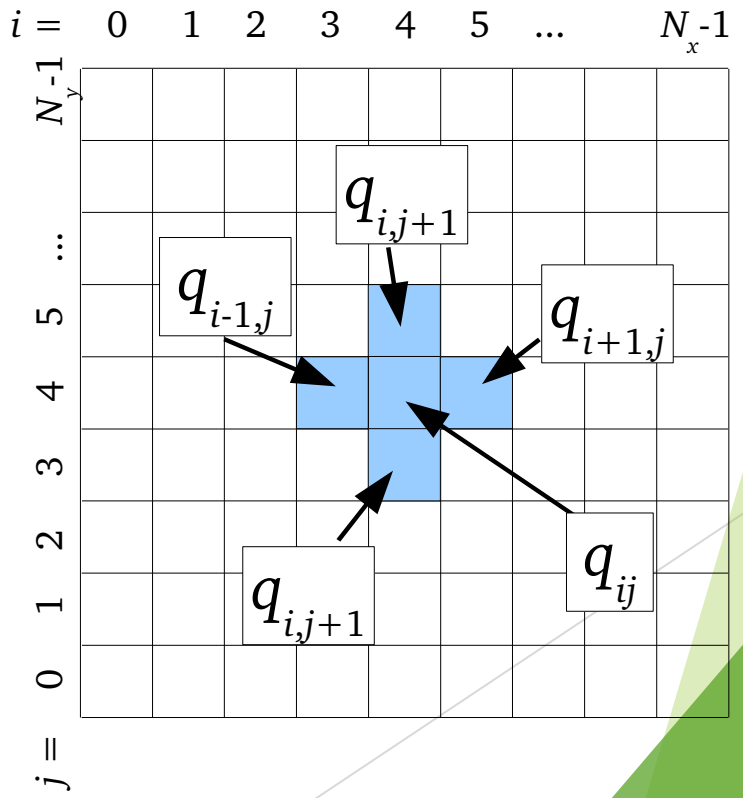
Method #2: Finite-volume methods

It works with *cell averages* of the solution:

$$q_{ij} \equiv \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} q(x, y)\, dx\, dy$$

$$x_i = \left(i + \frac{1}{2}\right) \Delta x \qquad i = 0 \ldots N_x - 1$$

$$y_j = \left(j + \frac{1}{2}\right) \Delta y \qquad j = 0 \ldots N_y - 1$$

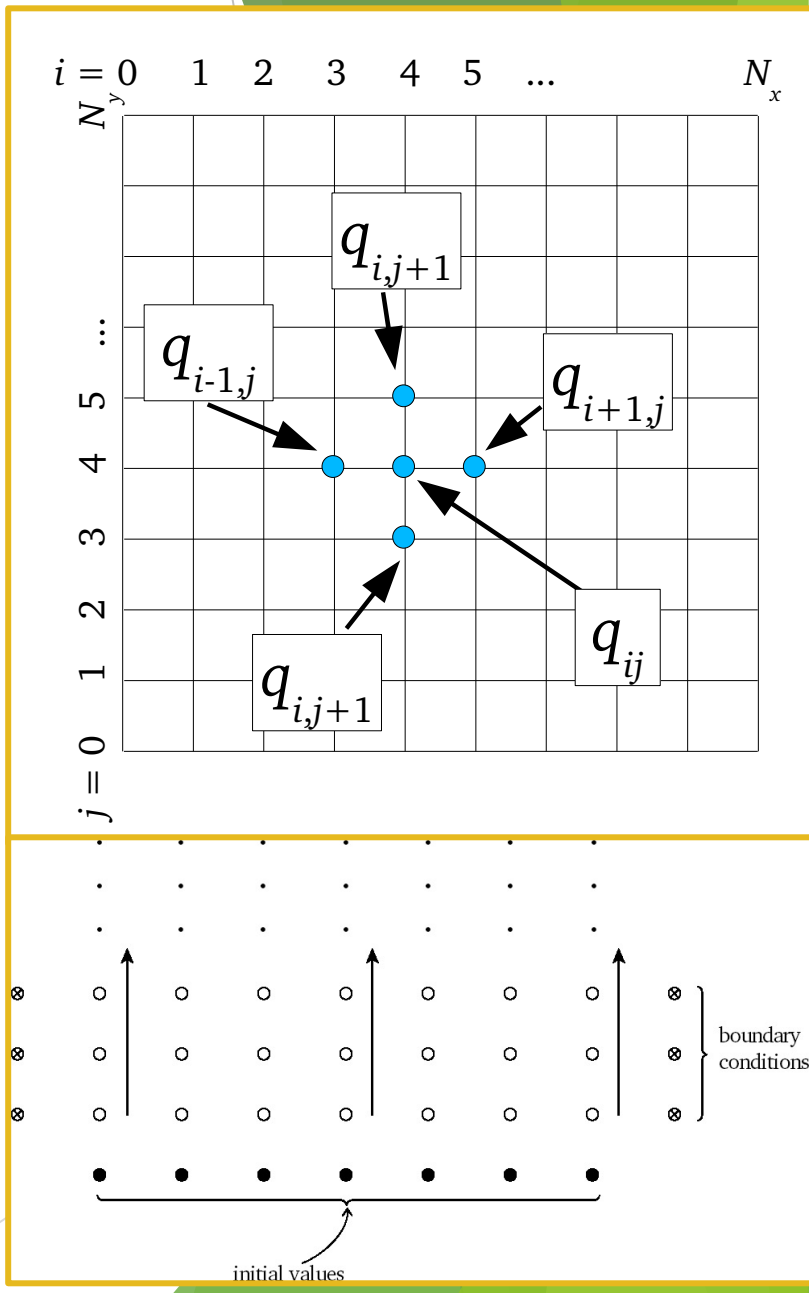


	Finite-difference methods Works with solutions at each mesh point	Finite-volume methods Works with cell averages of the solution on grid cells
Pros	Easy to implement Easy to obtain high-order derivatives	Can use unstructured grid Conservative
Cons	Need to use regular grid May not be conservative	Difficult to obtain higher-order derivatives

Finite-difference methods

Finite-difference methods

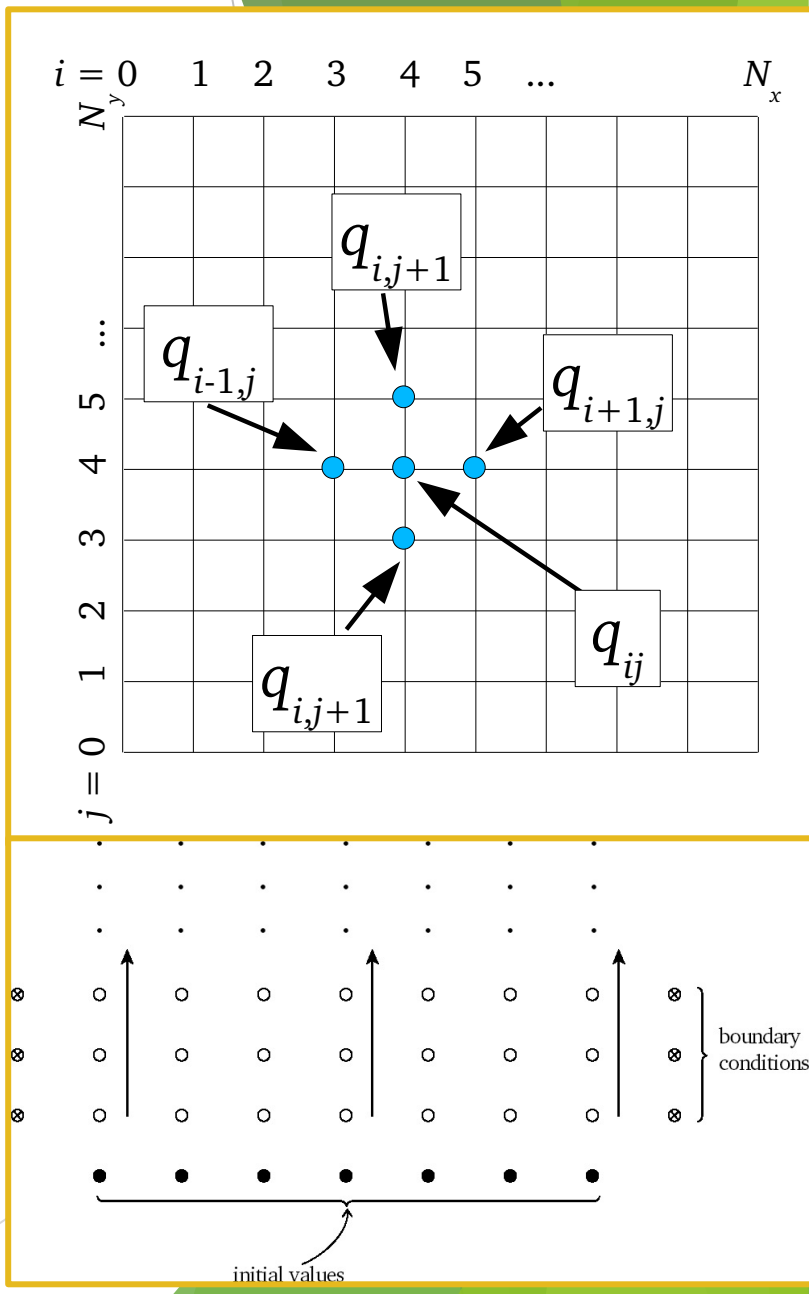
- Similar to finite-difference methods for solving BVP of ODEs, we seek ***solutions at fixed mesh points***
- For time-dependent PDEs, we often use fully-discrete methods, which discretize in both time and space dimensions
- We could then ***replace all derivatives by finite-difference approximations***
- Solution is obtained by starting with initial values along boundary of the domain and marching forward in time step by step
- Time-stepping procedure may be ***explicit*** or ***implicit***
  - Explicit method – only use info at t_n
  - Implicit method – also use info at t_{n+1}



Finite-difference methods

Similar to IVPs of solving ODEs, things to consider include:

- ***Accuracy*** of the solution would depend on
  - Step sizes in both space and time
  - Truncation error of the finite-difference approximations
- For ***stability***, step sizes must satisfy the stability criterion of a given numerical method
- For reducing computational ***cost***, enlarge step sizes as much as possible



Example: advection equation

$$u_t = -cu_x$$

- Let's discretize $\partial/\partial t$ with forward difference and $\partial/\partial x$ with centered difference ("*Forward Time Centered Space*" or *FTCS*):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -c\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}$$

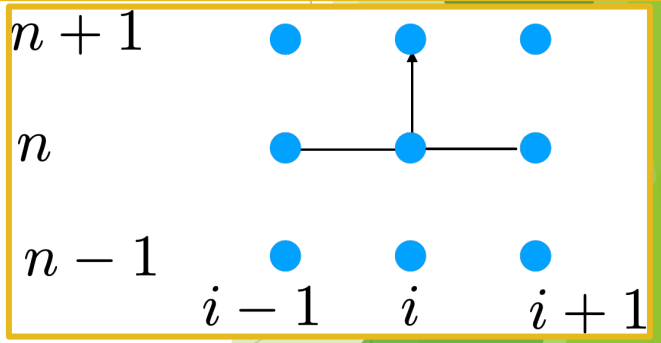
- Rearrange to create an *iterative* stepping algorithm:

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n)$$

- This is an *explicit* method since solution at t_{n+1} depends only on quantities at t_n

$$u_i^n \equiv u(t = t_n, x = x_i)$$

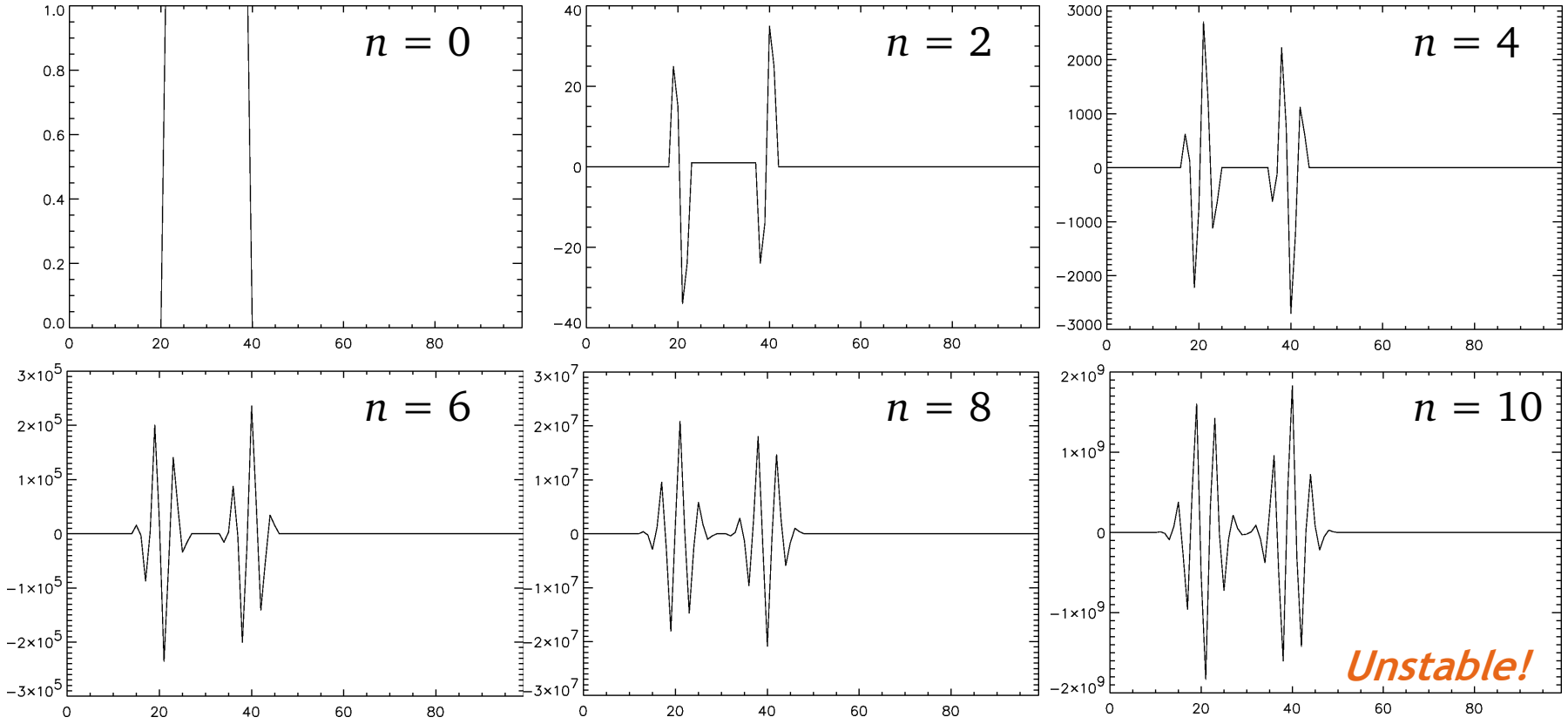
"*Stencil*": pattern of mesh points involved at each level



Example: advection equation

$$u_t = -cu_x$$

- Let's propagate a top-hat function!
- Use $N = 100, \frac{c\Delta t}{\Delta x} = 10$, $u_i^0 = 1$ for $20 < i < 40$ and $u_i^0 = 0$ elsewhere
- Results using ***FTCS***:



von Neumann stability analysis

- ***Difference equations have their own solutions, which are not necessarily the same as the true PDE solutions***
- Let's try solution of the form $u_j^n = \xi(k)^n e^{ikj\Delta x}$ in the FTCS algorithm:

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n)$$

$$\xi^{n+1} e^{ikj\Delta x} = \xi^n e^{ikj\Delta x} - \frac{c\Delta t}{2\Delta x}\left[\xi^n e^{ik(j+1)\Delta x} - \xi^n e^{ik(j-1)\Delta x}\right]$$

$$\xi = 1 - \frac{c\Delta t}{2\Delta x}\left[e^{ik\Delta x} - e^{-ik\Delta x}\right]$$

$$\xi = 1 - i\frac{c\Delta t}{\Delta x}\sin(k\Delta x)$$

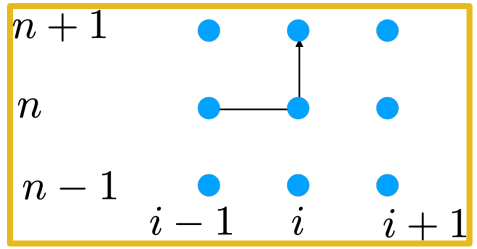
$$|\xi| = \left[1 + \left(\frac{c\Delta t}{\Delta x}\sin(k\Delta x)\right)^2\right] > 1$$

The FTCS method is unconditionally unstable - all modes grow exponentially with time!

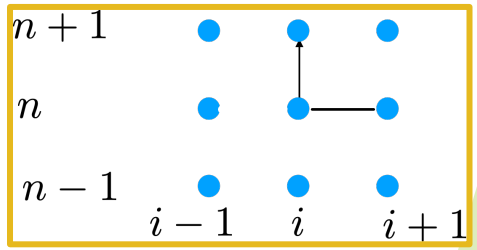
Alternative: the upwinding method

- One can obtain a ***conditionally stable*** scheme by using one-sided spatial difference:

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \qquad if\ c > 0$$



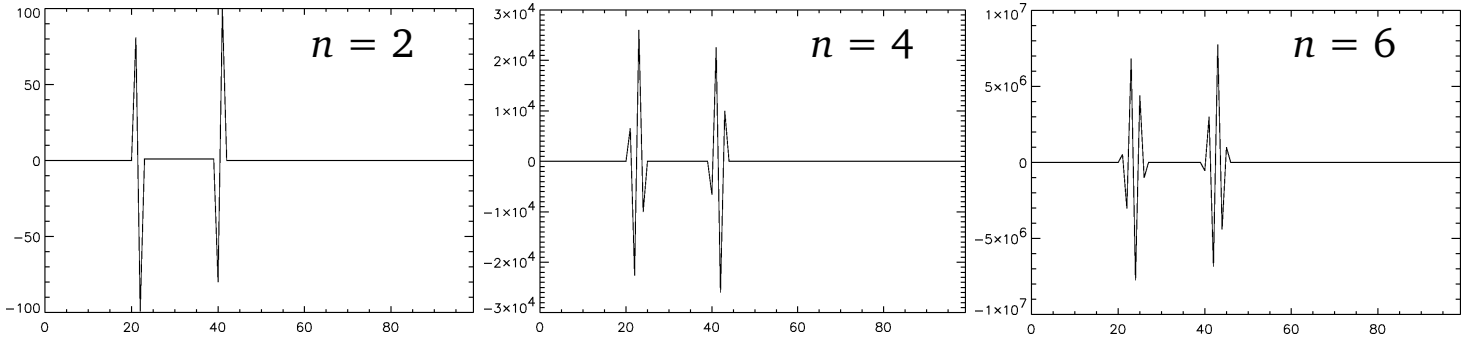
$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta x}(u_{i+1}^n - u_i^n) \qquad if\ c < 0$$



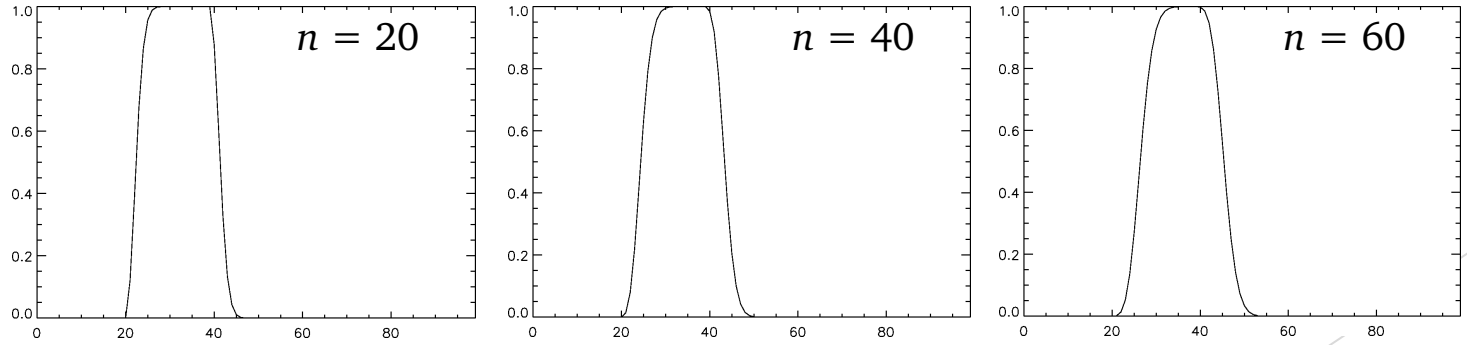
- This is called an ***upwind*** scheme because it uses info only from the "upwind" side

Alternative: the upwinding method

- With $\frac{c\Delta t}{\Delta x} > 1$, the upwind method still blows up:



- But with $\frac{c\Delta t}{\Delta x} < 1$, the method is stable:



Stability criterion for the upwind method

- von Neumann stability analysis yields:

$$\xi = 1 - \frac{c\Delta t}{\Delta x}\left[1 - e^{-ik\Delta x}\right] \qquad for\ c > 0$$

$$\xi = 1 - \frac{c\Delta t}{\Delta x}\left[1 - e^{ik\Delta x}\right] \qquad for\ c < 0$$

$$\Longrightarrow |\xi| < 1 \quad \text{as long as} \quad 0 < \frac{c\Delta t}{\Delta x} < 1$$

- This is the ***Courant-Friedrick-Lewy (CFL) criterion***, which constrains the time step to be

$$\Delta t < \frac{\Delta x}{c} \quad \text{or} \quad \Delta t = CFL \times \frac{\Delta x}{c}, where\ CFL \lesssim 1$$

CFL criterion

- Define the ***Courant number*** $C \equiv \frac{c\Delta t}{\Delta x}$, the CFL criterion says $\mathbf{C \leq C_{max}}$ for obtaining stable solutions
- Typically $C_{max} = 1$ for explicit methods; larger for implicit methods
- Intuitive meaning of the CFL criterion:

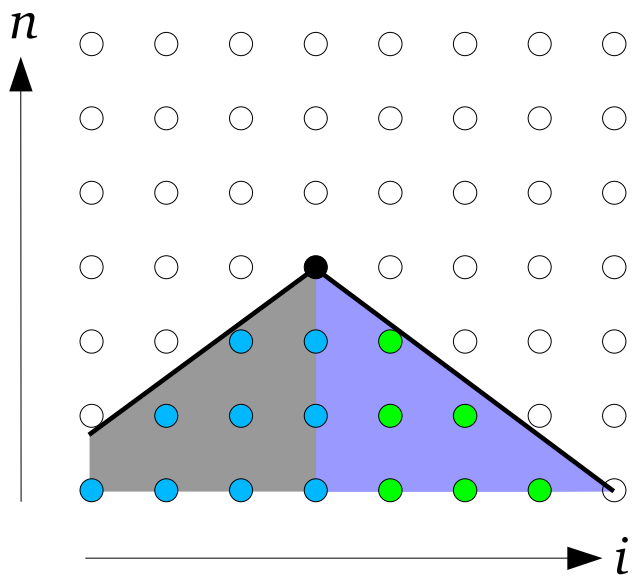
$$\mathrm{C} \equiv \frac{c}{\frac{\Delta x}{\Delta t}} = \frac{\text{physical speed of signal}}{\text{numerical speed of information propagation}}$$

CFL criterion

$$C \equiv \frac{c}{\frac{\Delta x}{\Delta t}} = \frac{\text{physical speed of signal}}{\text{numerical speed of information propagation}}$$

Domains of dependence

	c>0	c<0
Numerical (difference eqn)	●	●
Continuum (PDE)	◢	◣



$|c\ \Delta t/\Delta x| > 1$ unstable

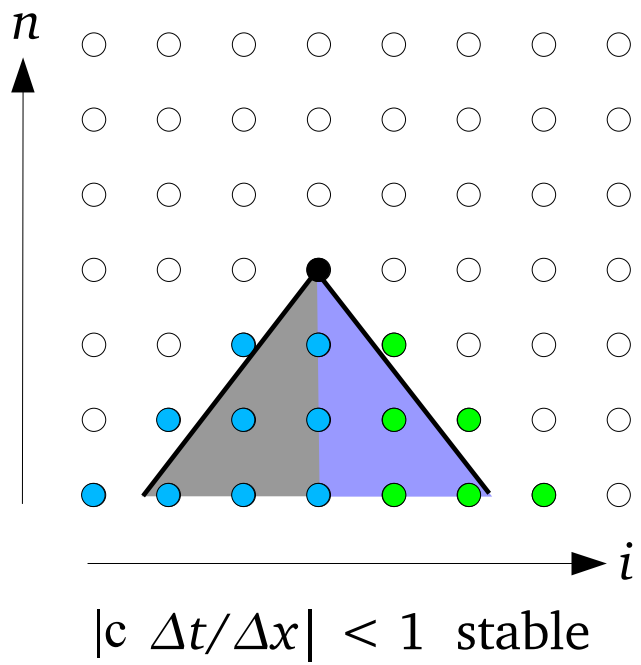
- ***Dots*** - updated numerical solutions
- ***Shaded region*** - info needed to obtain solution (black dot)
- If $C > 1$, propagation speed of numerical info is ***slower*** than physical speed, so there is ***insufficient info*** to obtain the physical solution

CFL criterion

$$C \equiv \frac{c}{\frac{\Delta x}{\Delta t}} = \frac{\text{physical speed of signal}}{\text{numerical speed of information propagation}}$$

Domains of dependence

	c>0	c<0
Numerical (difference eqn)	●	●
Continuum (PDE)	◢	◣

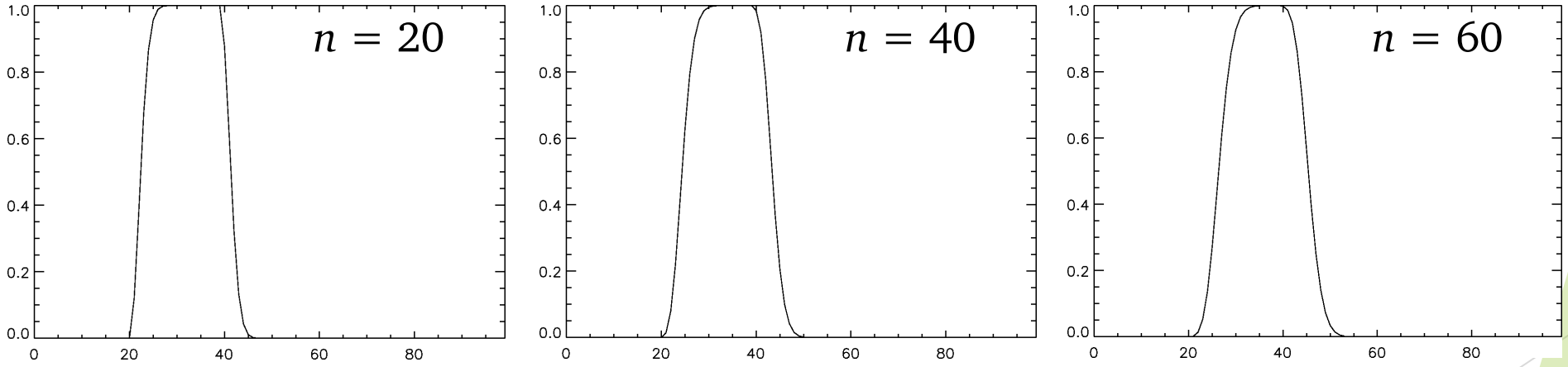


$|c\ \Delta t/\Delta x| < 1$ stable

- If $C < 1$, propagation speed of numerical info is ***faster*** than physical speed, so there is ***sufficient info*** to obtain the physical solution

Truncation errors of the upwind method

- Derived from Taylor expansion, the upwind method has truncation errors $\propto \mathcal{O}(\Delta t \Delta x^2)$
- The error acts like a ***diffusion*** term, smearing out discontinuities

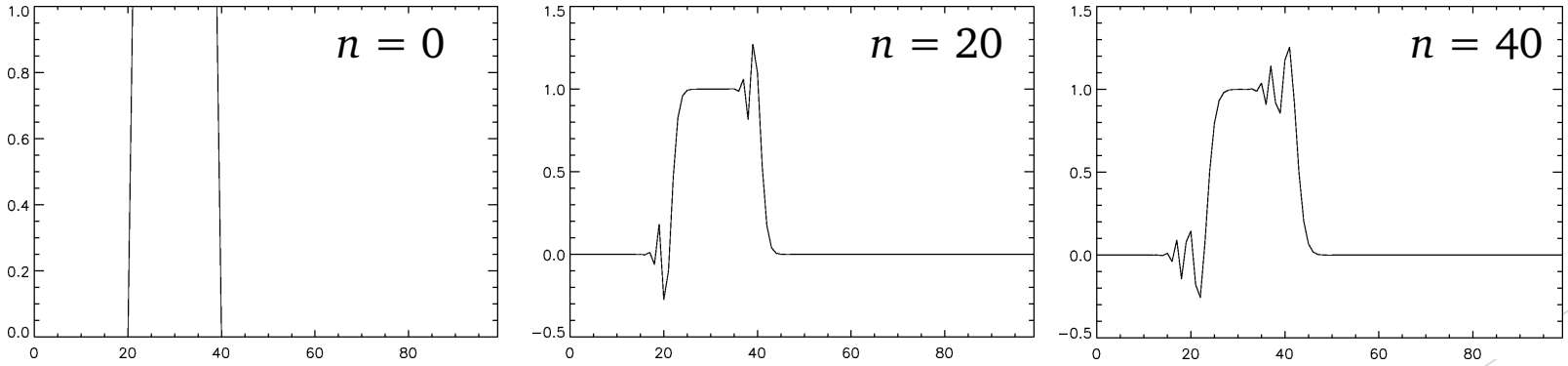


Lax-Wendroff method

- Use centered spatial difference and use Taylor expansion to higher-order terms:

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \frac{c^2\Delta t^2}{2\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

- Truncation error is ***2nd-order*** in space and time
- The error is ***dispersive*** (produce ripples) rather than diffusive

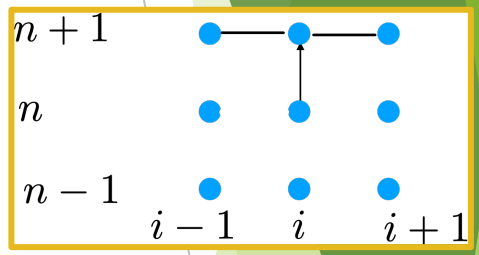


Implicit methods: BTCS for advection equation

- An ***implicit*** method uses spatial derivatives at t_{n+1}
- An example is "***backward time centered space***" or ***BTCS*** method:

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

Stencil

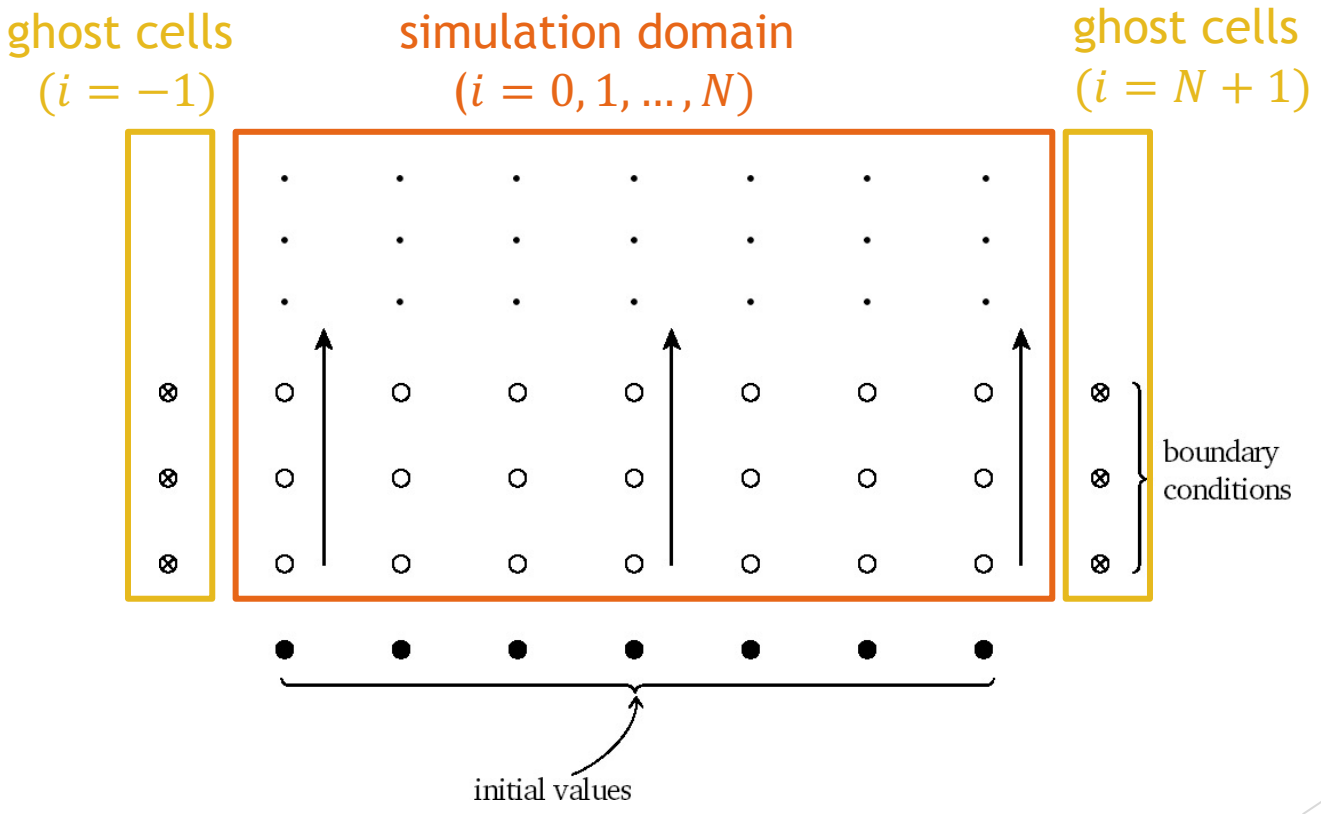


- Solution can be obtained by solving the following linear system of equations:

$$\begin{bmatrix} 1 & \sigma & & & & \\ -\sigma & 1 & \sigma & & & \\ & -\sigma & 1 & \sigma & & \\ & & & \ddots & & \\ & & & -\sigma & 1 & \sigma \\ & & & & -\sigma & 1 \end{bmatrix} \begin{bmatrix} u_0^{n+1} \\ u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \\ u_N^{n+1} \end{bmatrix} = \begin{bmatrix} u_0^{n} \\ u_1^{n} \\ u_2^{n} \\ \vdots \\ u_{N-1}^{n} \\ u_N^{n} \end{bmatrix}, \qquad \sigma \equiv \frac{c\,\Delta t}{2\,\Delta x}$$

Boundary conditions

- ***Boundary conditions (BCs)*** are needed for updating solutions within the simulation domain
- Typically, we establish ***ghost cells*** around the boundary and set their values at each timestep



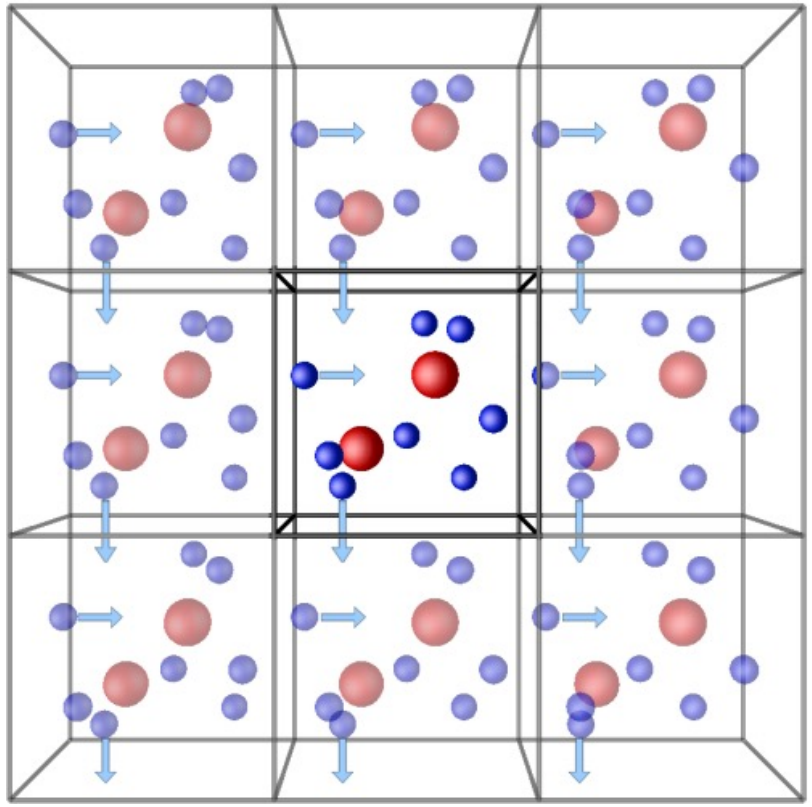
***Higher-order methods would require more ghost cells

Commonly used BCs

- ***Periodic BCs*** – require all solution variables $X = (\rho, P, v_x, v_y, v_z)$ to be periodic functions of the simulation box
- This is to set the boundary values at each timestep:

$$X_{-1} = X_N$$

$$X_{N+1} = X_1$$



Commonly used BCs

- ***Outflow BCs*** – allow material to flow out of the grid: $X_{-1} = X_0$

$$X_{N+1} = X_N$$

- ***Reflect BCs*** – used for solid surfaces and symmetry axes. Preserve all quantities except assigning negative signs to the velocity component normal to the surfaces:

$$v_{x,0} = v_{x,N} = 0$$

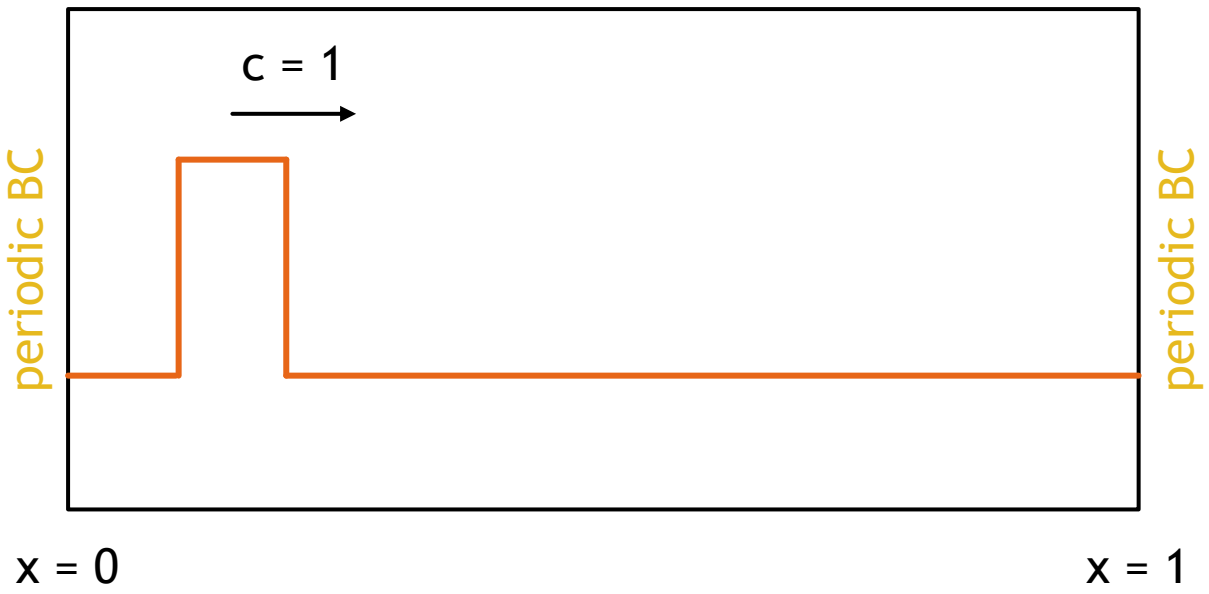
$$X_{-1} = X_1 \qquad v_{x,-1} = -v_{x,1} \qquad v_{y,-1} = v_{y,1}$$

$$X_{N+1} = X_{N-1} \qquad v_{x,N+1} = -v_{x,N-1} \qquad v_{y,N+1} = v_{y,N-1}$$

Exercise #1 – solving advection equation using finite-difference methods

$$u_t = -cu_x$$

- Let's reproduce the results for propagating a top-hat function using the *FTCS* & *upwind* methods



Exercise #1 – solving advection equation using finite-difference methods

- Connect to CICA and load the Fortran compiler:

  ```
  module load pgi
  ```

- Go to your exercise directory on CICA:

  ```
  cd astr660/exercise
  ```

- Copy the template Fortran files to the current directory:

  ```
  cp -r /data/hyang/shared/astro660CompAstro/L10exercise ./
  ```

- Enter the fdm directory and see what files are there:

  ```
  cd L10exercise/fdm
  ls
  ```

Exercise #1 – solving advection equation using finite-difference methods

- Take a look at the files and make sure you understand how they work
  - advection.f90 – main program
  - Simulation_data.f90 – where the simulation parameters are defined (mesh points, coordinates, CFL number, etc)
  - initial.f90 – where the top-hat function is initialized
  - evolution.f90 – where the solutions are advanced in time and subroutine update is implemented
  - boundary.f90 – where the periodic BCs are defined
  - IO.f90 – where the output format is controlled
  - Makefile

Exercise #1 – solving advection equation using finite-difference methods

- Implement the ***upwind*** method by filling in subroutine update in evolution.f90
- Implement the ***periodic BC*** in boundary.f90
- Compile and run the codes by

  ```
  make
  ./advection
  ```

- Use your favorite plotting routine to plot the results from output files "advection_*.d"

Upwind method for c > 0

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

Exercise #1b – solving advection equation using finite-difference methods

- Implement the ***FTCS*** method by modifying subroutine update in evolution.f90
- Compile and run the codes by

  ```
  make
  ./advection
  ```

- Use your favorite plotting routine to plot the results from output files "advection_*.d"
- You could try using a larger CFL number (=1.2) to see how the solution goes unstable
- To get the bonus credit, please submit your code and plots ***for both methods*** to the TAs by end of today (11/17/2022)

FTCS method

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n)$$

Finite-volume methods

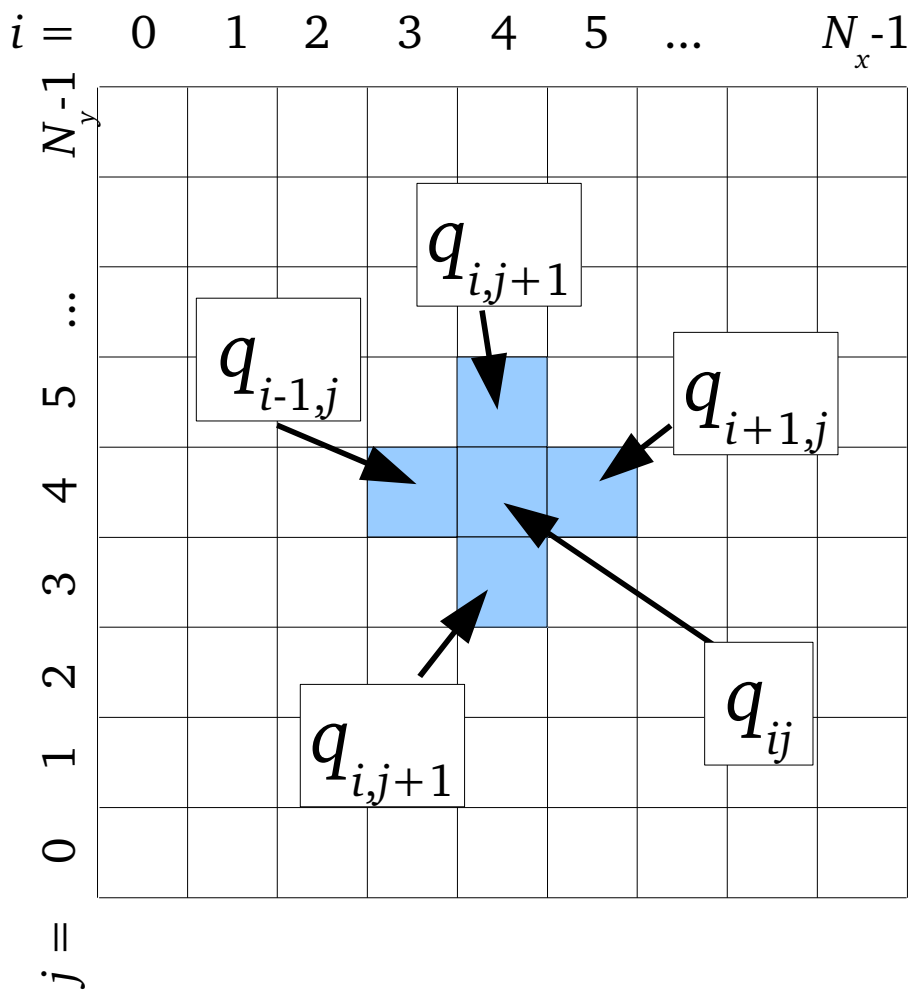
Finite-volume methods

- This method is widely used ***grid-based*** hydro codes
- Mesh points -> ***grid cells***
- $q_i^n = q(t = t_n, x = x_i)$ now represents ***cell-averaged*** values instead of cell-centered values:

$$q_{ij} \equiv \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} q(x, y)\, dx\, dy$$

- We can write down a general ***conservation*** law:

$$\frac{\partial q}{\partial t} + \nabla \cdot \boldsymbol{F}(q, t) = 0$$

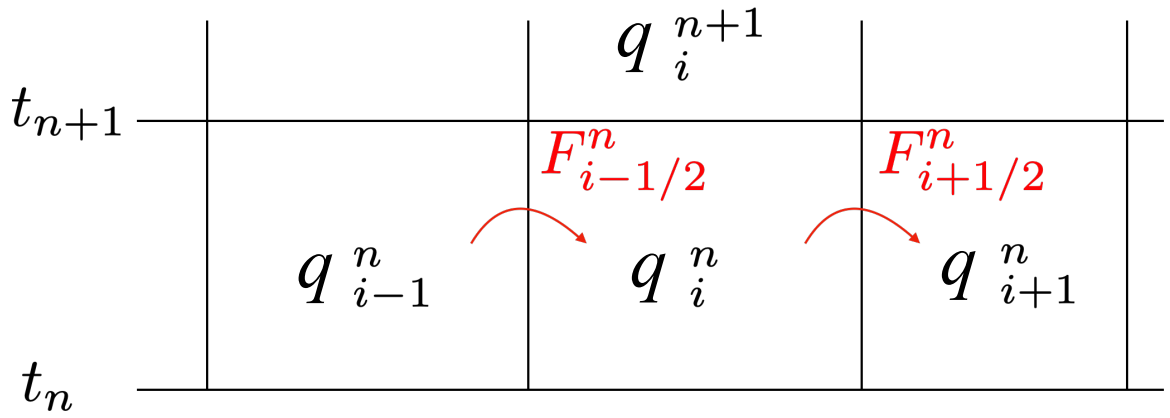


Finite-volume methods

- Cell values can be updated by evaluating ***fluxes*** at cell edges (in 1D):

★ *Step 2*

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right)$$



- The above equation is exact. So the problem becomes how to

★ ***approximate the fluxes at cell edges!!***

Step 1

Recall

$$\frac{\partial q}{\partial t} + \nabla \cdot \boldsymbol{F}(q,t) = 0$$

Example: advection equation $u_t = -cu_x$

- Rewrite the advection equation in the conservative form $\frac{\partial q}{\partial t} + \nabla \cdot \boldsymbol{F}(q,t) = 0$

  then $\boldsymbol{q = u, F(q,t) = cu}$

- Now use the ***finite-volume*** method: $q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right)$

- The simplest approach to approximate fluxes at cell edges as simple ***arithmetic averages***:

$$F_{i-1/2}^n = \frac{1}{2}\left[F(q_{i-1}^n) + F(q_i^n)\right]$$

-> This method is unstable!

Algorithms for evaluating the fluxes

Lax-Friedrichs method:

$$F^n_{i-1/2} = \frac{1}{2}[F(q^n_{i-1}) + F(q^n_i)] - \frac{\Delta x}{2\Delta t}(q^n_i - q^n_{i-1})$$

Upwind method

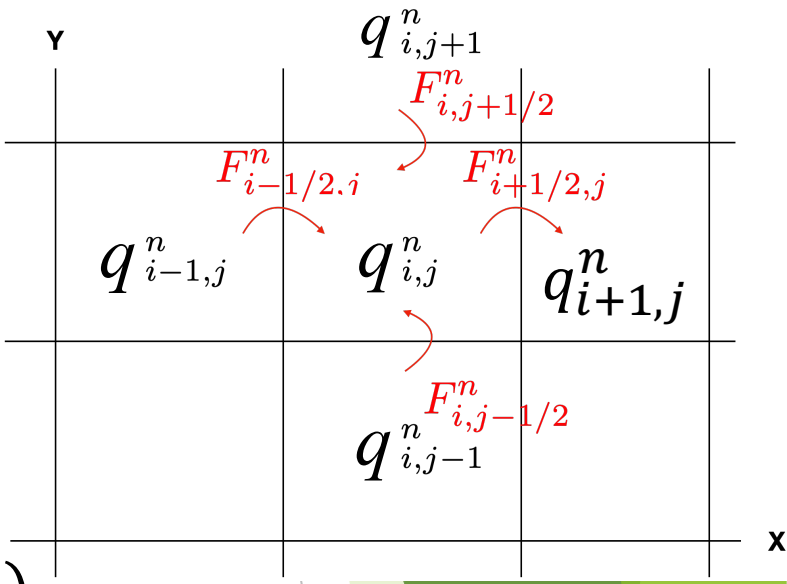
$$F^n_{i-1/2} = F(q^n_{i-1})$$

Lax-Wendroff method:

$$q^{n+1/2}_{i-1/2} = \frac{1}{2}(q^n_i + q^n_{i-1}) - \frac{\Delta t}{2\Delta x}[F(q^n_i) - F(q^n_{i-1})]$$

$$q^{n+1}_i = q^n_i - \frac{\Delta t}{\Delta x}[F\left(q^{n+1/2}_{i+1/2}\right) - F(q^{n+1/2}_{i-1/2})]$$

Multi-D finite-volume methods



2D:

$$q_{i,j}^{n+1} = q_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(F_{i+1/2,j}^{n} - F_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(F_{i,j+1/2}^{n} - F_{i,j-1/2}^{n}\right)$$

3D:

$$q_{i,j}^{n+1} = q_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(F_{i+1/2,j}^{n} - F_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(F_{i,j+1/2}^{n} - F_{i,j-1/2}^{n}\right) - \frac{\Delta t}{\Delta z}\left(F_{i,j+1/2}^{n} - F_{i,j-1/2}^{n}\right)$$

Dimensional splitting - updating the x-, y-, and z-fluxes separately

Operator splitting

- Creating multi-D differencing methods is complex, so ***operator splitting*** is a useful technique
- Suppose we have a difference operator D with truncation error $\mathcal{O}(\Delta t)$:

$$D = D_1 + D_2 + D_3$$

- Then we can approximate D by

$$D[q] = D_1[D_2[D_3[q]]] + O(\Delta t)$$

- If D is $\mathcal{O}(\Delta t^2)$, we can do better by symmetrizing the order:

$$D[q] = D_1^{1/2}[D_2^{1/2}[D_3^{1/2}[D_3^{1/2}[D_2^{1/2}[D_1^{1/2}[q]]]]]] + O(\Delta t^2)$$

- This is called ***Strang splitting***. Each operator is applied for $(\frac{1}{2})\Delta t$

Example: advection equation in 2D

$$u_t = -c_x u_x - c_y u_y$$

- Define the two Lax-Wendroff operators:

$$D_x\left[u_{i,j}, \Delta t\right] = u_{i,j} - \frac{c_x \Delta t}{2\Delta x}\left(u_{i+1,j} - u_{i-1,j}\right) + \frac{{c_x}^2 \Delta t^2}{2\Delta x^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right)$$

$$D_y\left[u_{i,j}, \Delta t\right] = u_{i,j} - \frac{c_y \Delta t}{2\Delta y}\left(u_{i,j+1} - u_{i,j-1}\right) + \frac{{c_y}^2 \Delta t^2}{2\Delta y^2}\left(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right)$$

- Then we can create the following 2nd-order 2D method:

$$u_{i,j}^{(1)} = D_x\left[u_{i,j}^{n}, \frac{\Delta t}{2}\right]$$

$$u_{i,j}^{(2)} = D_y\left[u_{i,j}^{(1)}, \frac{\Delta t}{2}\right]$$

$$u_{i,j}^{(3)} = D_y\left[u_{i,j}^{(2)}, \frac{\Delta t}{2}\right]$$

$$u_{i,j}^{n+1} = D_x\left[u_{i,j}^{(3)}, \frac{\Delta t}{2}\right]$$

PDEs -- summary

- *Partial differential equations (PDEs)* involve partial derivatives w.r.t more than one independent variable
- Three types of 2nd-order PDEs:
  - *Hyperbolic*: time-dependent systems, conservative physical processes (e.g., advection, hydro), not evolving toward steady state, IVP+BVP
  - *Parabolic*: time-dependent systems, dissipative physical processes (e.g., diffusion), are evolving toward steady state, IVP+BVP
  - *Elliptic*: time-independent systems (e.g., gravitational potential for a point mass), already reached steady state, BVP

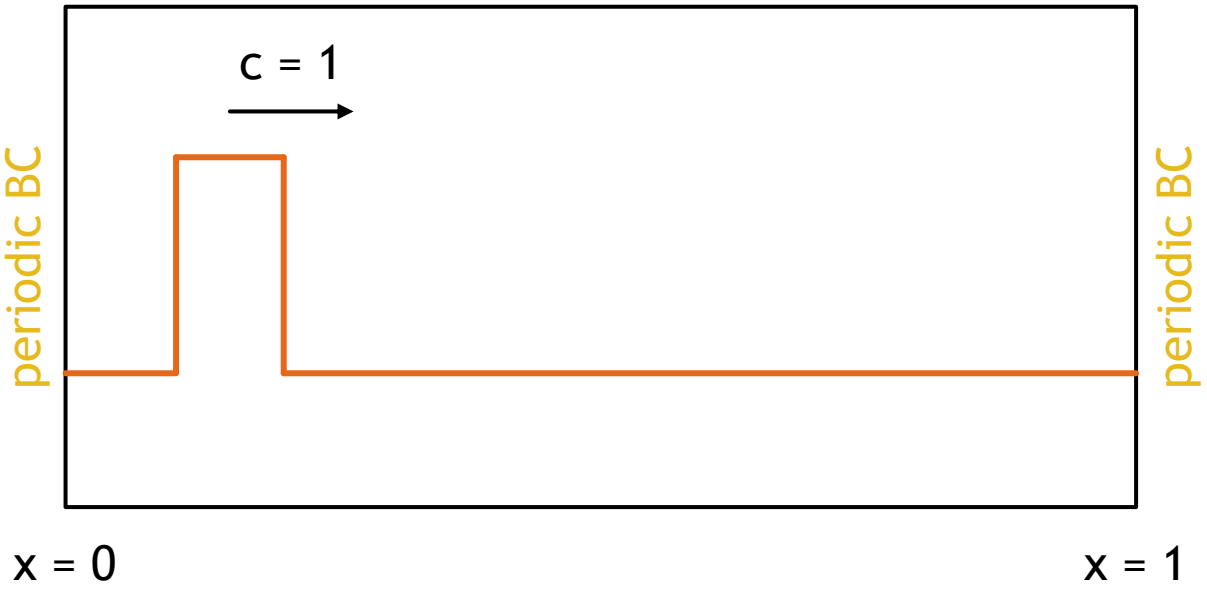
PDEs -- summary

- Two ways to discretize PDEs:
  - *Finite-difference* methods – replacing derivatives using finite-difference approximations at fixed *mesh points*
  - *Finite-volume* methods – solving *cell-averaged* quantities on *grid cells*, key is to approximate *fluxes* at cell edges
- For time-dependent PDEs, stability criterion can be obtained using the von Neumann stability analysis
- *CFL criterion*: Courant number $C \equiv \frac{c\Delta t}{\Delta x} \leq C_{max} \approx 1$
- Commonly used BCs: periodic, outflow, and reflect
- *Operator splitting* is often used for multi-D simulations

Exercise #2 – solving advection equation using finite-volume methods

$$u_t = -cu_x$$

- Let's propagate the top-hat function using the ***finite-volume*** methods



$$\frac{\partial q}{\partial t} + \nabla \cdot \boldsymbol{F}(q,t) = 0$$

Rewrite the advection equation in the conservative form then

$$\boldsymbol{q = u, F(q,t) = cu}$$

Exercise #2 – solving advection equation using finite-volume methods

- Create a directory fvm under L10exercise:

  ```
  mkdir L10exercise/fvm
  ```

- Copy the *.f90 files and Makefile you wrote from the fdm directory to fvm
- Modify the update subroutine in evolution.f90 in order to implement the ***finite-volume*** method

evolution.f90 - where the update subroutine & fluxes are evaluated

Update solutions using fluxes at cell edges

This is where the fluxes can be evaluated using different methods

```
subroutine update(time, dt)
    use Simulation_data
    implicit none
    real, intent(in) :: time ,dt
    integer :: i
    real    :: FL, FR

    uold  = u
    do i = istart, iend
        call flux(i,dt,FL,FR)
        u(i) = uold(i) - dt/dx*(FR-FL)
    enddo

end subroutine update

!
! Routine to evaluate flux cell edges
!
subroutine flux(i,dt,FL, FR)
    use Simulation_data
    implicit none
    integer, intent(in) :: i
    real, intent(in)    :: dt
    real, intent(out)   :: FL, FR

    ! Arithmetic average method
    FL = ! TODO
    FR = ! TODO

    ! The Lax-Friedrichs Method
    !FL = ! TODO
    !FR = ! TODO

    ! The upwind method
    !FL = ! TODO
    !FR = ! TODO

    ! The Lax-Wendroff Method
    !FL = uold(i-1)*c
    !FR = uold(i)*c

end subroutine flux
```

Exercise #2 – solving advection equation using finite-volume methods

- Implement the ***arithmetic average*** method by modifying the update and flux subroutines in evolution.f90
- Compile and run the codes by

  ```
  make
  ./advection
  ```

- Use your favorite plotting routine to plot the results and see whether the solution is stable or not
- Try other methods for evaluating the fluxes and compare the results

Arithmetic average method

$$F^n_{i-1/2} = \frac{1}{2}\left[F(q^n_{i-1}) + F(q^n_i)\right]$$

References & acknowledgements

- Course materials of Computational Astrophysics from Prof. Kuo-Chuan Pan (NTHU)
- Course materials of Computational Astrophysics from Prof. Hsi-Yu Schive (NTU)
- Course materials of Computational Astrophysics and Cosmology from Prof. Paul Ricker (UIUC)
- “Computational Physics” by Rubin H. Landau, Manuel Jose Paez and Cristian C. Bordeianu
- “Scientific Computing – An Introductory Survey” by Michael T. Heath