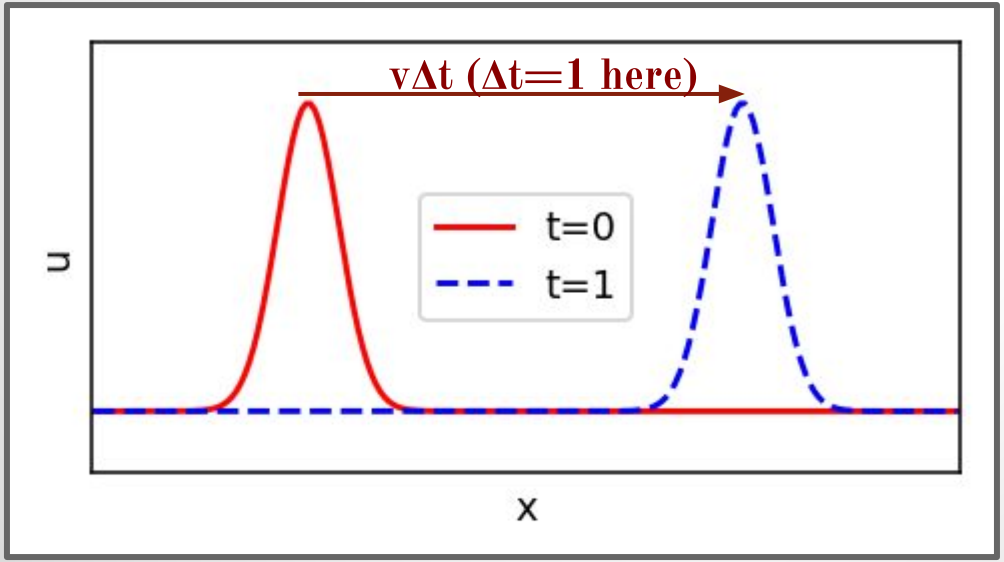
Initial-Value Problems

Hsi-Yu Schive (薛熙于)
National Taiwan University

Advection of a Scalar

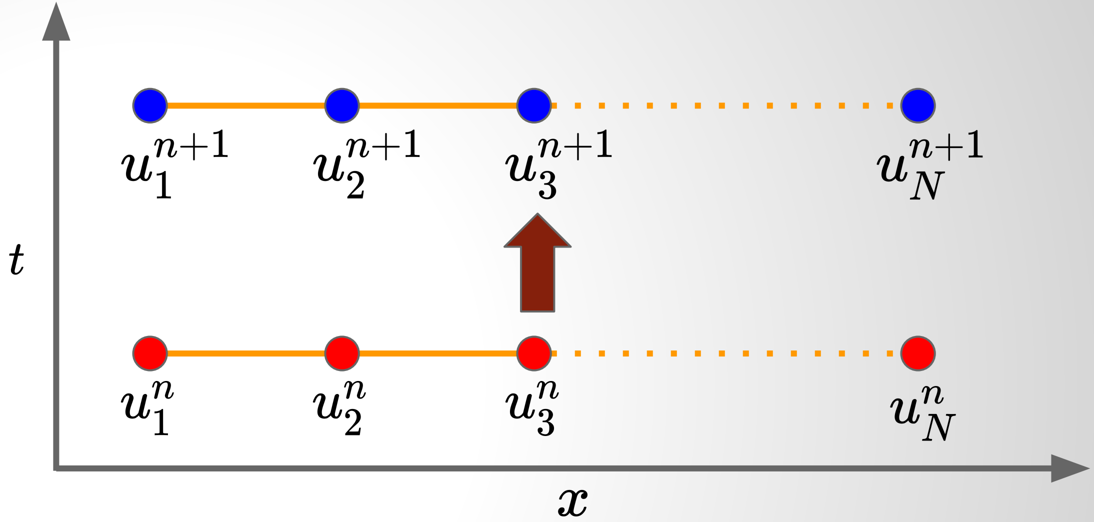
- **Governing eq.** $\boxed{\dfrac{\partial u(x,t)}{\partial t} = -v\dfrac{\partial u(x,t)}{\partial x}}$
  - **Scalar** u **is simply transported with a velocity** v
  - **Assuming** v **is constant**
  - u **is conserved** → $\int u(x,t)dx = constant$



Finite Difference Approximation

- **Discretize space and time**

$$\boxed{\begin{aligned} &u(x,t) \Rightarrow u_j^n \\ &x_j = x_0 + j\Delta x \\ &t_n = t_0 + n\Delta t \end{aligned}}$$



- **Given u_j^n , solve u_j^{n+1}**
- **Taylor expansion**

$$\boxed{f(\alpha+\Delta\alpha) = f(\alpha) + f'(\alpha)\Delta\alpha + \frac{1}{2!}f''(\alpha)\Delta\alpha^2 + \frac{1}{3!}f'''(\alpha)\Delta\alpha^3 + \ldots}$$

  - **Use it to approximate partial derivatives by discrete u_j^n**
  - **That's what differentiates different numerical schemes**
    - **May NOT be as trivial as you think!**

Forward-Time Central-Space Scheme

- **Advection eq.** $$\boxed{\frac{\partial u(x,t)}{\partial t} = -v\frac{\partial u(x,t)}{\partial x}}$$
- **FTCS scheme:**

forward-time

$$\frac{\partial u(x_j,t_n)}{\partial t} \rightarrow \frac{u_j^{n+1}-u_j^n}{\Delta t} + \underline{O(\Delta t)}$$

errors

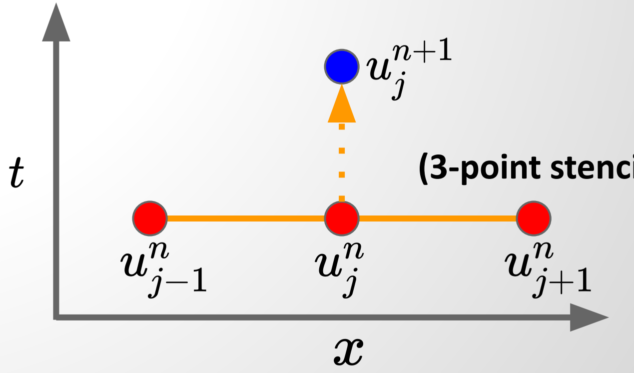
$$\frac{\partial u(x_j,t_n)}{\partial x} \rightarrow \frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x} + \underline{O(\Delta x^2)}$$

central-space

$$\boxed{u_j^{n+1} = u_j^n - \frac{v\Delta t}{2\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right)}$$

LHS: t=n+1

RHS: t=n



Forward-Time Central-Space Scheme

- Explicit scheme
  - u_j^{n+1} of each j can be computed explicitly from values at $t = t_n$
  - u_j^{n+1} of different j can be computed independently (and thus *in parallel*)
  - In comparison, implicit schemes solve coupled equations of u_j^{n+1} with multiple j simultaneously
    - Will be introduced shortly

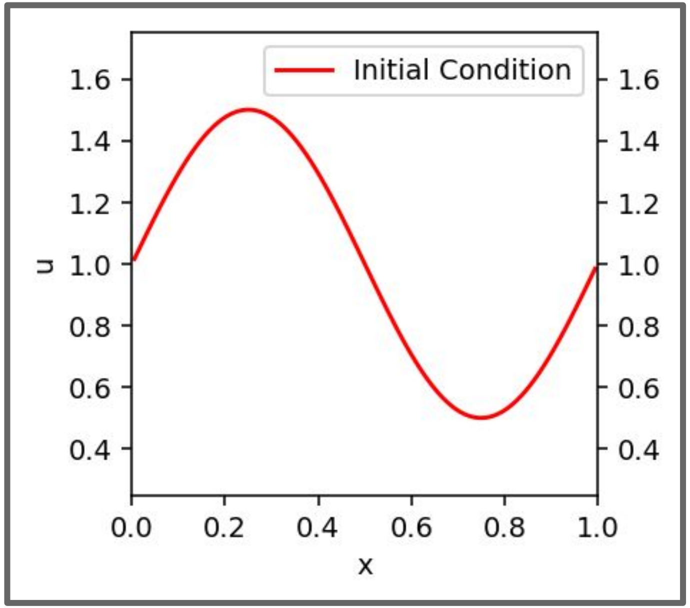
- FTCS scheme is very simple. But, it is UNSTABLE in general for hyperbolic equations!

Demo: FTCS Scheme for Advection

```
# constants
L   = 1.0   # 1-D computational domain size
N   = 100   # number of computing cells
v   = 1.0   # advection velocity
u0  = 1.0   # background density
amp = 0.5   # sinusoidal amplitude
cfl = 0.8   # Courant condition factor
```

↖️ proportional to time-step (will be introduced later)

```
# derived constants
dx     = L/N      # spatial resolution
dt     = cfl*dx/v # time interval for data update
period = L/v      # time period
```



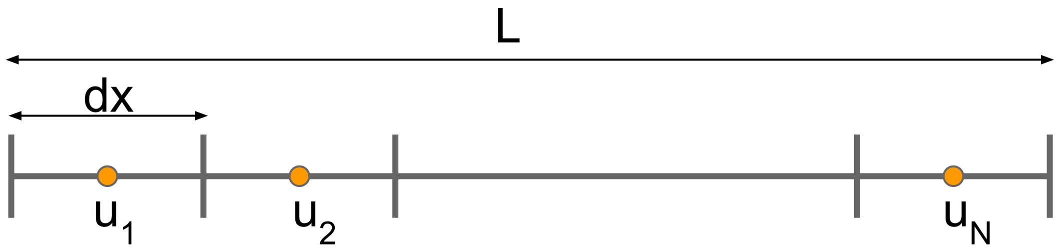
lec02-demo01.py

Demo: FTCS Scheme for Advection

```
# define a reference analytical solution
def ref_func( x, t ):
   k = 2.0*np.pi/L   # wavenumber
   return u0 + amp*np.sin( k*(x-v*t) )
```

$\leftarrow u(x, t) = f(x - vt)$

```
# initial condition
t = 0.0
x = np.arange( 0.0, L, dx ) + 0.5*dx   # cell-centered coordinates
u = ref_func( x, t )  # initial density distribution
```



lec02-demo01.py

Demo: FTCS Scheme for Advection

```
def update( frame ):
   for step in range( nstep_per_image ):
#     back up the input data
      u_in = u.copy()   ← to avoid overwriting the input data before updating them
#     update all cells
      for i in range( N ):
         ip = (i+1  ) % N   # periodic boundary condition
         im = (i-1+N) % N
#        FTCS scheme
         u[i] = u_in[i] - dt*v*( u_in[ip] - u_in[im] )/(2.0*dx)
#     update time                              ↑central-difference
      t = t + dt
      if ( t >= end_time ):   break
#  calculate the reference analytical solution and estimate errors
   u_ref = ref_func( x, t )
   err   = np.abs( u_ref - u ).sum()/N
```

i=N-1 ⇨ ip=0

i=0 ⇨ im=N-1

↓ L1 error ≡ ∑|numerical-analytical|/N

lec02-demo01.py

Run **lec02-demo01.py**

Lessons Learned from FTCS

- Numerical errors are dominated by amplitude errors
  - Both phase and dispersion errors are negligible
- Amplitude errors increase with time
  - Low-k (long-wavelength) errors dominate first
  - High-k (short-wavelength) errors appear later but grow faster
  - Amplitude increases instead of decreases → sign of instability
  - Smaller Δt → errors decrease, but still unstable!
- Is mass conserved?

von Neumann Stability Analysis

- **Linear PDE with periodic boundary conditions**
  - **Plane-wave solution:** $u(x,t) = \sum_k A_k e^{i(kx-wt)} \equiv \sum_k A_k u_k$
  - **Let** $w = w_R + iw_I \rightarrow u_k(x,t) = e^{w_I t} e^{i(kx - w_R t)}$
    - w_R **: oscillating mode**
    - w_I **: growing or damping mode**
  - **Stability criterion:** $\boxed{w_I \leq 0}$
    - **Compute** w_I **by inserting** u_k **into the linear PDE and solving the dispersion relation** $w(k)$
- **Similarly, for a finite difference scheme**
  - $\boxed{u_j^n = e^{i(kj\Delta x - wn\Delta t)} \equiv \xi^n e^{i(kj\Delta x)}}$,

    where $\xi \equiv e^{-i(w\Delta t)} = e^{w_I \Delta t} e^{-iw_R \Delta t}$, $|\xi| = e^{w_I \Delta t}$
  - **Stability criterion:** $\boxed{|\xi| \leq 1}$

Stability Analysis for FTCS

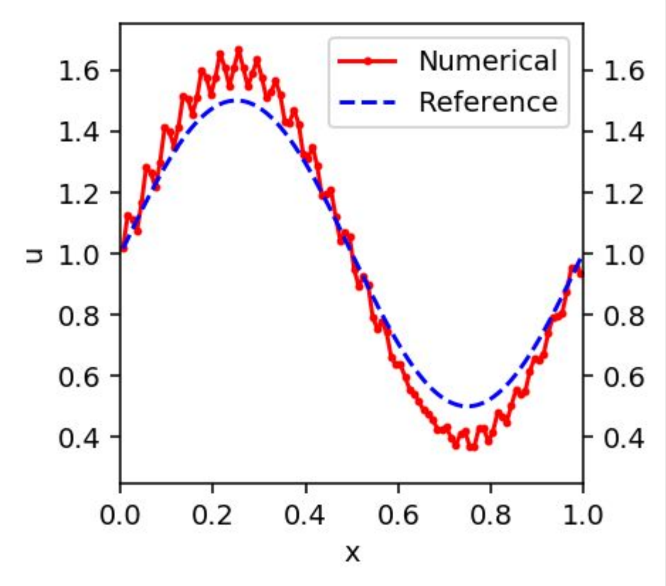
- Insert $u_j^n = \xi^n e^{i(kj\Delta x)}$ into $u_j^{n+1} = u_j^n - r(u_{j+1}^n - u_{j-1}^n)/2, \text{where } r \equiv \dfrac{v\Delta t}{\Delta x}$

$$\rightarrow \xi^{n+1} e^{ikj\Delta x} = \xi^n e^{ikj\Delta x} - r(\xi^n e^{ik(j+1)\Delta x} - \xi^n e^{ik(j-1)\Delta x})/2$$

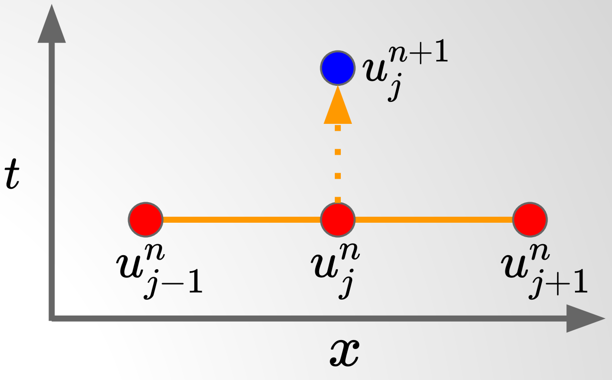
$$\rightarrow \xi = 1 - r(e^{ik\Delta x} - e^{-ik\Delta x})/2 = 1 - ir\sin(k\Delta x)$$

$$\rightarrow \boxed{|\xi|^2 = 1 + r^2 \sin^2(k\Delta x) \geq 1}$$

- FTCS scheme is **unconditionally unstable!**
  - In other words, it is unstable for all k and r
- In general, high-k modes are more unstable
  - Lead to grid-scale ($k\Delta x$~1) oscillations



Lax Scheme



- $$\boxed{u_j^{n+1} = \boxed{\frac{1}{2}(u_{j+1}^n + u_{j-1}^n)} - \frac{v\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n)}$$

- **Insert** $u_j^n = \xi^n e^{i(kj\Delta x)}$ **and let** $r \equiv \dfrac{v\Delta t}{\Delta x}$

  $\rightarrow \xi^{n+1} e^{ikj\Delta x} = (\xi^n e^{ik(j+1)\Delta x} + \xi^n e^{ik(j-1)\Delta x})/2 - r(\xi^n e^{ik(j+1)\Delta x} - \xi^n e^{ik(j-1)\Delta x})/2$

  $\rightarrow \xi = (e^{ik\Delta x} + e^{-ik\Delta x})/2 - r(e^{ik\Delta x} - e^{-ik\Delta x})/2 = \cos(k\Delta x) - ir\sin(k\Delta x)$

  $\rightarrow \boxed{|\xi|^2 = 1 + (r^2 - 1)\sin^2(k\Delta x)}$

- **For stability,** $|\xi|^2 \le 1 \rightarrow r^2 \le 1 \rightarrow \boxed{\Delta t \le \dfrac{\Delta x}{v}}$ **Courant-Friedrichs-Lewy (CFL) condition**

- $\dfrac{v\Delta t}{\Delta x}$ **: CFL number**

- $|\xi|^2 < 1 \rightarrow$ **numerical dissipation**

Lax Scheme

- Lax scheme is <u>conditionally stable</u>
  - CFL condition must be satisfied
- But why?
  - For a time-step $\varDelta t$, the maximum distance information can propagate is $v\varDelta t$
  - But our finite difference scheme only collect data from $\varDelta x$
  - If $v\Delta t > \Delta x$, the correct update requires information <u>more distant than the finite difference scheme knows</u>
- Numerical dissipation
  - The Lax scheme can be rewritten as

$$u_j^{n+1} = \underbrace{u_j^n - \frac{v\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n)}_{\text{original FTCS scheme}} + \underbrace{\boxed{\frac{1}{2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n)}}_{\text{numerical dissipation}} \quad \frac{(\Delta x)^2}{2\Delta t}\frac{\partial^2 u}{\partial x^2}$$

Demo: Lax Scheme for Advection

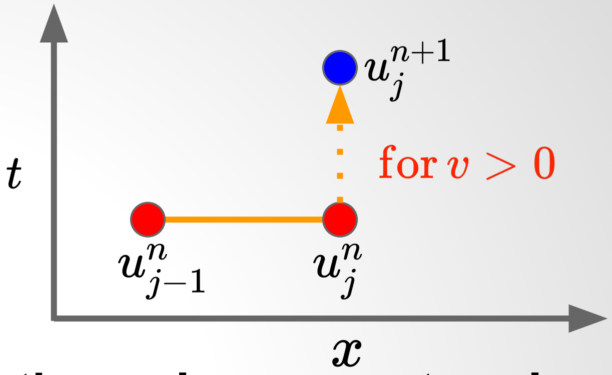
```
# (1) FTCS scheme (unconditionally unstable)
  u[i] = u_in[i] - dt*v*( u_in[ip] - u_in[im] )/(2.0*dx)

# (2) Lax scheme (conditionally stable)
  u[i] = 0.5*( u_in[im] + u_in[ip] ) - dt*v*( u_in[ip] - u_in[im] )/(2.0*dx)
```

lec02-demo01.py

Run lec02-demo01.py

Upwind Scheme

- $$\boxed{u_j^{n+1} = \begin{cases} u_j^n - \dfrac{v\Delta t}{\Delta x}\boxed{(u_j^n - u_{j-1}^n), \quad v > 0} \\ u_j^n - \dfrac{v\Delta t}{\Delta x}(u_{j+1}^n - u_j^n), \quad v < 0 \end{cases}}$$



- **Take into account the fact that <u>information only propagates along the v direction</u> (i.e., from upwind to downwind)**
  - **Only upwind cells should affect the solution**
  - **There can be more than one characteristic speeds (e.g., in hydro/MHD)**
  - **In general, $v \to v_j^n$, so the upwind direction can change**
- **Stability criterion:** $|\xi|^2 = 1 - 2|r|(1 - |r|)(1 - \cos(k\Delta x)) \leq 1, \text{where } r \equiv v\Delta t/\Delta x$

$$\to \boxed{\Delta t \leq \frac{\Delta x}{v}}$$ CFL condition again

→ unconditionally unstable if switching to downwind

Demo: Upwind Scheme for Advection

```
# (1) FTCS scheme (unconditionally unstable)
  u[i] = u_in[i] - dt*v*( u_in[ip] - u_in[im] )/(2.0*dx)

# (2) Lax scheme (conditionally stable)
  u[i] = 0.5*( u_in[im] + u_in[ip] ) - dt*v*( u_in[ip] - u_in[im] )/(2.0*dx)

# (3) upwind scheme (assuming v>0; conditionally stable)
  u[i] = u_in[i] - dt*v*( u_in[i] - u_in[im] )/dx

# (4) downwind scheme (assuming v>0; unconditionally unstable)
  u[i] = u_in[i] - dt*v*( u_in[ip] - u_in[i] )/dx
```

lec02-demo01.py

Run lec02-demo01.py

Lessons Learned from Upwind

- Numerical errors are dominated by amplitude errors
  - Both phase and dispersion errors are negligible
  - Similar to FTCS
- But amplitude decreases instead of increases → sign of stability
  - Numerical dissipation (diffusion)
  - No dissipation when CFL=1 → Perfect! But why?
- Low-k (long-wavelength) errors dominate all the time
- First- or second-order accuracy? (HW)
  - How do errors scale with Δx for a fixed CFL number?
- Is mass conserved?
- Downwind scheme is unconditionally unstable

Second-Order Accuracy in Time

- Aforementioned schemes are all <u>first-order accurate in time</u>

$$u'(t) \approx \frac{u(t+\Delta t) - u(t)}{\Delta t} + O(\Delta t)$$

$$\rightarrow u(t+\Delta t) \approx u(t) + \boxed{u'(t)\Delta t} + \underline{O(\Delta t^2)}$$

- How to improve it?

$$u(t+\Delta t) \approx u(t+\frac{1}{2}\Delta t) + u'(t+\frac{1}{2}\Delta t)\frac{\Delta t}{2} + \cancel{\frac{1}{2}u''(t+\frac{1}{2}\Delta t)\left(\frac{\Delta t}{2}\right)^2} + O(\Delta t^3)$$

$$u(t) \approx u(t+\frac{1}{2}\Delta t) - u'(t+\frac{1}{2}\Delta t)\frac{\Delta t}{2} + \cancel{\frac{1}{2}u''(t+\frac{1}{2}\Delta t)\left(\frac{\Delta t}{2}\right)^2} - O(\Delta t^3)$$

$$\rightarrow u(t+\Delta t) \approx u(t) + \boxed{u'(t+\tfrac{1}{2}\Delta t)\Delta t} + \underline{O(\Delta t^3)}$$

2nd-order Runge-Kutta
(or mid-point) method

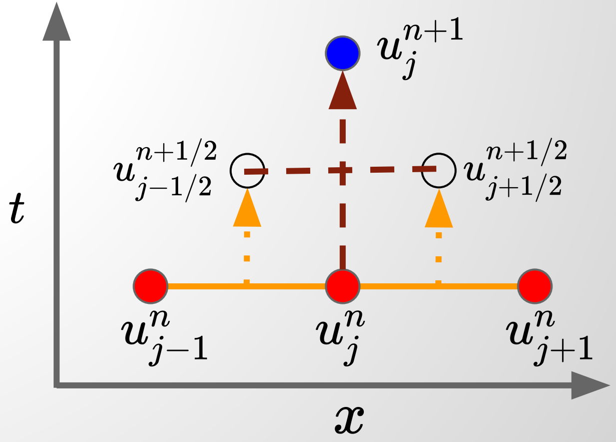
Lax-Wendroff Scheme

- Two-step approaches
  - Step 1: evaluate $u_{j+1/2}^{n+1/2}$ defined at the half time-step $n+1/2$ and the cell interface $j+1/2$ with the Lax scheme

$$\boxed{u_{j+1/2}^{n+1/2} = \frac{1}{2}(u_{j+1}^n + u_j^n) - \frac{v\Delta t}{2\Delta x}(u_{j+1}^n - u_j^n)}$$

  - Step 2: use $u_{j+1/2}^{n+1/2}$ to evaluate the half-step fluxes for the full-step update

$$\boxed{u_j^{n+1} = u_j^n - \frac{v\Delta t}{\Delta x}(u_{j+1/2}^{n+1/2} - u_{j-1/2}^{n+1/2})}$$



Lax-Wendroff Scheme

- **Stability criterion**

$$|\xi|^2 = 1 - r^2(1 - r^2)(1 - \cos(k\Delta x))^2 \le 1, \text{where } r \equiv \frac{v\Delta t}{\Delta x}$$

↖️ **HW1: prove it!**

$$\rightarrow \Delta t \le \frac{\Delta x}{v}$$ **CFL condition again**

- **Numerical dissipation for long-wavelength modes (i.e., small $k\Delta x$)**
  - **Lax-Wendroff:** $|\xi|^2 - 1 \approx -r^2(1 - r^2)\frac{(k\Delta x)^4}{4} \propto \Delta x^4$
  - **Lax:** $|\xi|^2 - 1 \approx -(1 - r^2)(k\Delta x)^2 \propto \Delta x^2$

Demo: Lax-Wendroff for Advection

```
def update( frame ):
   for step in range( nstep_per_image ):
#     back up the input data
      u_in = u.copy()

#     calculate the half-timestep solution
      u_half = np.empty( N )
      for i in range( N ):
#        u_half[i] is defined at the left face of cell i
         im = (i-1+N) % N  # periodic boundary condition
         u_half[i] = 0.5*(u_in[i]+u_in[im]) - 0.5*dt*v*(u_in[i]-u_in[im])/dx

#     update all cells for a full timestep
      for i in range( N ):
         ip = (i+1) % N    # periodic boundary condition
         u[i] = u_in[i] - dt*v*( u_half[ip] - u_half[i] )/dx
```

lec02-demo02.py

Run lec02-demo02.py

Compare with lec02-demo01.py

Check how errors scale with N

Lessons Learned from Lax-Wendroff

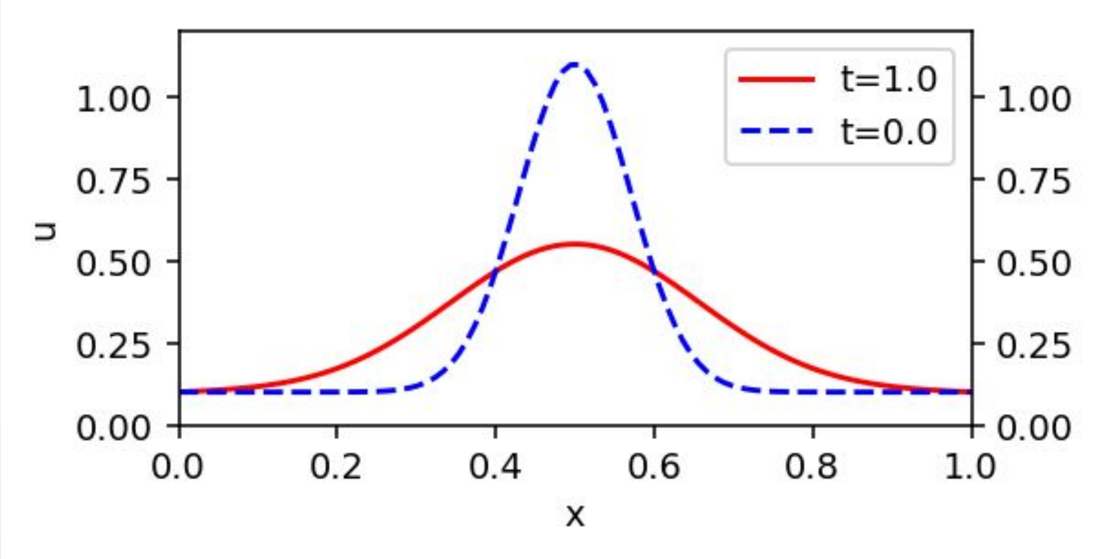
- General features of errors → similar to upwind
  - But dissipation errors are much smaller!
  - No dissipation when CFL=1 again
- First- or second-order accuracy? (HW)
- Is mass conserved?
- We will discuss its disadvantages when applying it to hydrodynamics in the next lecture

On-Course Exercise

- Play with different schemes, time-step, and spatial resolution
  - See how they affect things like
    - Numerical dissipation
    - Numerical dispersion (i.e., phase errors)
    - Stability
- Is total mass conserved? Why?
- Change the initial condition: sinusoidal → Gaussian $e^{-(x-L/2)^2/\sigma^2}$
- Change the boundary condition: periodic → inflow at x=0 and outflow at x=L

Diffusion of a Scalar

- **Governing eq.** $$\boxed{\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^2 u(x,t)}{\partial x^2}}$$
  - **Scalar u simply diffuses away with a diffusion constant D**
  - **Assuming D is constant**
  - **u is conserved →** $\int u(x,t)dx = constant$



FTCS Scheme for Diffusion

- **Diffusion eq.** $$\boxed{\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^2 u(x,t)}{\partial x^2}}$$
- **FTCS scheme:**

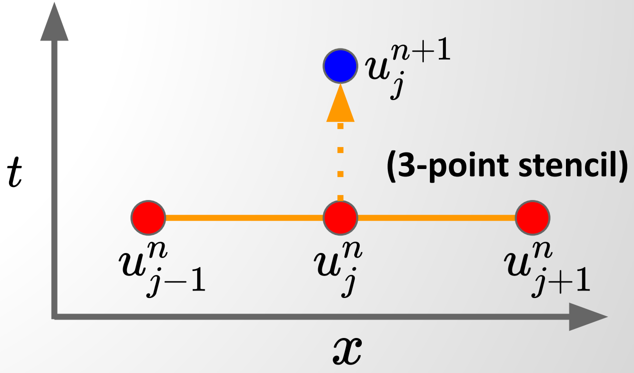
$$\frac{\partial u(x_j,t_n)}{\partial t} \rightarrow \frac{u_j^{n+1} - u_j^n}{\Delta t} + O(\Delta t)$$

↙️ forward-time

$$\frac{\partial^2 u(x_j,t_n)}{\partial x^2} \rightarrow \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} + O(\Delta x^2)$$

↖️ central-space

$$\boxed{u_j^{n+1} = u_j^n + \frac{D\Delta t}{\Delta x^2}\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right)}$$



FTCS Scheme for Diffusion

- Stability criterion

$$|\xi| = \left|1 - \frac{4D\Delta t}{\Delta x^2}\sin^2\left(\frac{k\Delta x}{2}\right)\right| \leq 1$$

$$\rightarrow \boxed{\Delta t \leq \frac{\Delta x^2}{2D} \propto \Delta x^2}$$

- High resolution may require prohibitively small time-steps
  - For all diffusion-like equations (e.g., heat conduction, Schroedinger eq.)
  - Very unpleasant feature!
  - Motivate <u>implicit</u> methods (to be discussed soon)

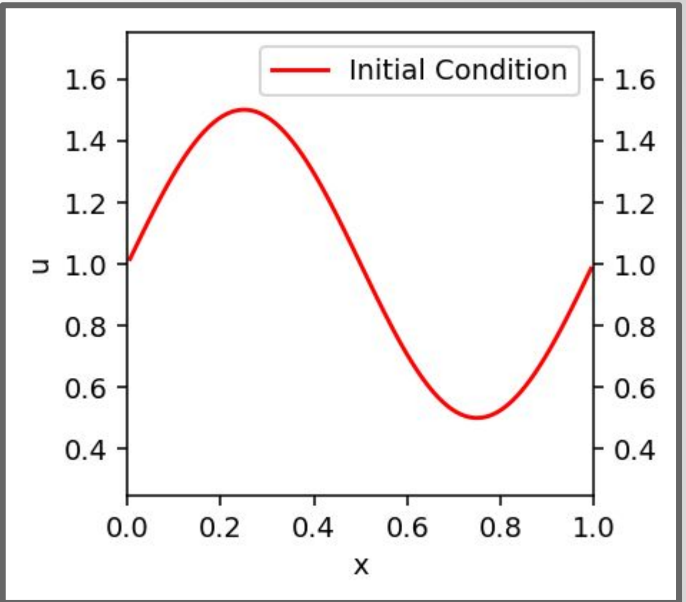
- Catch: diffusion eq. actually propagates information at infinite speed
  - Example: instantaneous point mass $\boxed{u(x,t) = \frac{M}{\sqrt{4\pi D t}}\exp\left(-\frac{x^2}{4Dt}\right)}$

Demo: FTCS Scheme for Diffusion

```
# constants
L   = 1.0   # 1-D computational domain size
N   = 100   # number of sampling points
D   = 1.0   # diffusion coefficient
u0  = 1.0   # background density
amp = 0.5   # sinusoidal amplitude
cfl = 0.8   # Courant condition factor
```

↖️ proportional to time-step

```
# derived constants
dx      = L/(N-1)               # spatial resolution
dt      = cfl*0.5*dx**2.0/D     # time-step
t_scale = (0.5*L/np.pi)**2.0/D # diffusion time scale across L
```



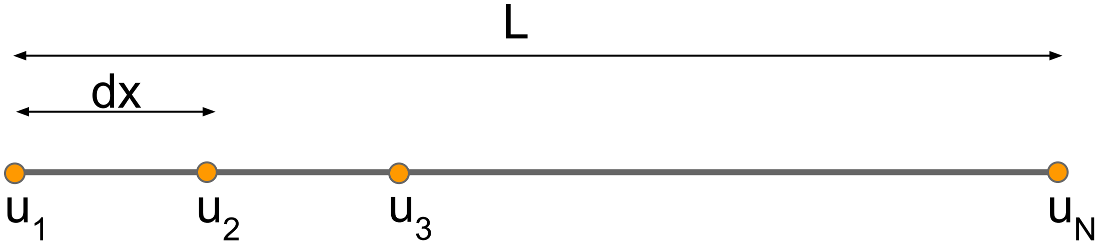
lec02-demo03.py

Demo: FTCS Scheme for Diffusion

```
# define a reference analytical solution
def ref_func( x, t ):
    k = 2.0*np.pi/L   # wavenumber
    return u0 + amp*np.sin( k*x )*np.exp( -k**2.0*D*t )

# initial condition
t = 0.0
x = np.linspace( 0.0, L, N )  # coordinates including both ends
u = ref_func( x, t )          # initial density distribution
```

$$\boxed{u(x,t) = u_0 + A\sin(kx)e^{-k^2 Dt}}$$

periodic or Dirichlet
boundary condition



lec02-demo03.py

Demo: FTCS Scheme for Diffusion

```
def update( frame ):
   global t, u

   for step in range( nstep_per_image ):
#     back up the input data
      u_in = u.copy()

#     update all **interior** cells with the FTCS scheme
      for i in range( 1, N-1 ):
         u[i] = u_in[i] + dt*D*( u_in[i+1] - 2*u_in[i] + u_in[i-1] )/dx**2.0

#     update time
      t = t + dt
      if ( t >= end_time ):   break
```

skip u[0] and u[N-1], which are fixed for the Dirichlet boundary condition

↑central-difference

lec02-demo03.py

Run **lec02-demo03.py**

Backward-Time Central-Space Scheme

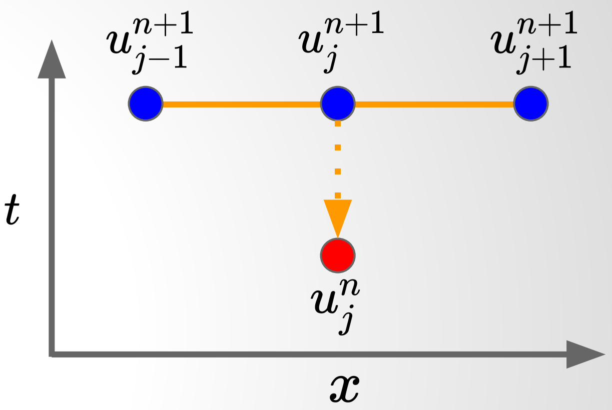
- $$\boxed{u_j^{n+1} = u_j^n + \frac{D\Delta t}{\Delta x^2}\left(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\right)}$$

- **Stability analysis:**

  Insert $u_j^n = \xi^n e^{i(kj\Delta x)}$ **and let** $\alpha \equiv \dfrac{D\Delta t}{\Delta x^2}$

$$\begin{aligned}
\rightarrow \xi &= 1 + \alpha\xi\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right) \\
&= 1 + 2\alpha\xi\left(\cos(k\Delta x) - 1\right) \\
&= 1 - 4\alpha\xi\sin^2(k\Delta x/2)
\end{aligned}$$

$$\rightarrow \boxed{\xi = \frac{1}{1 + 4\alpha\sin^2(k\Delta x/2)} \leq 1}$$



unconditionally stable!

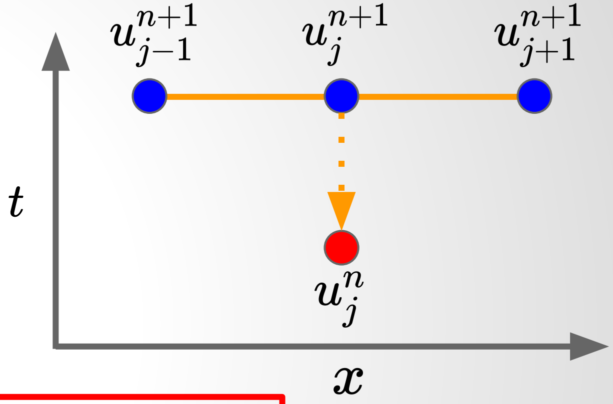
Backward-Time Central-Space Scheme

- $\boxed{u_j^{n+1} = u_j^n + \alpha(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1})}$

- **BTCS scheme is fully <u>implicit</u>**
  - u_j^{n+1} **with different j are coupled together**
  - **Must solve N coupled linear equations**

$$\boxed{-\alpha u_{j-1}^{n+1} + (1+2\alpha)u_j^{n+1} - \alpha u_{j+1}^{n+1} = u_j^n, \quad j = 1 \ldots N}$$

  - **Can be put into a matrix form $AU^{n+1} = U^n$, where A is a tridiagonal coefficient matrix and U^n is the column vector of u_j^n**



Backward-Time Central-Space Scheme

$$\underset{A}{\begin{bmatrix} (1+2\alpha) & -\alpha & 0 & \cdot & \cdot \\ -\alpha & (1+2\alpha) & -\alpha & 0 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 0 & -\alpha & (1+2\alpha) & -\alpha \\ \cdot & \cdot & 0 & -\alpha & (1+2\alpha) \end{bmatrix}} \underset{U^{n+1}}{\begin{bmatrix} u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ u_{N-2}^{n+1} \\ u_{N-1}^{n+1} \end{bmatrix}} = \underset{U^n}{\begin{bmatrix} u_2^n \\ u_3^n \\ \vdots \\ u_{N-2}^n \\ u_{N-1}^n \end{bmatrix}} + \underset{\text{boundary condition}}{\begin{bmatrix} \alpha u_1^{n+1} \\ 0 \\ \vdots \\ 0 \\ \alpha u_N^{n+1} \end{bmatrix}}$$

- **Last term (denoted as B) is associated with the boundary conditions**
  - **Here we apply the Dirichlet boundary condition by fixing u_1 and u_N**
- **Solve $U^{n+1} = A^{-1}(U^n + B)$ with a linear algebra library**

Demo: BTCS Scheme for Diffusion

```
# set the coefficient matrices A with A*u(t+dt)=u(t)
r = D*dt/dx**2
A = np.diagflat( np.ones(N-3)*(-r),         -1 ) + \
    np.diagflat( np.ones(N-2)*(1.0+2.0*r), 0 ) + \
    np.diagflat( np.ones(N-3)*(-r),         +1 );

def update( frame ):
   for step in range( nstep_per_image ):
#      (1) copy u(t) for adding boundary conditions
       u_bk = np.copy( u[1:-1] )
#      (2) apply the Dirichlet boundary condition: u[0]=u[N-1]=u0
       u_bk[ 0] += r*u0
       u_bk[-1] += r*u0
#      (3) compute u(t+dt)
       u[1:-1] = np.linalg.solve( A, u_bk )
```

← set tridiagonal matrix with NumPy diagflat(); note that it only needs to be set once

← NumPy solve() returns u by solving $Au = u\_bk$

↑do not update the boundary values

lec02-demo04.py

Run lec02-demo04.py

Compare with lec02-demo03.py
Increase cfl in both cases

Crank Nicolson scheme

- $$u_j^{n+1} = u_j^n + \frac{D\Delta t}{2\Delta x^2}\boxed{\left[(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}) + (u_{j+1}^n - 2u_j^n + u_{j-1}^n)\right]}$$

- **This is your homework!**

Homework

- **Derive the stability criterion of the Lax-Wendroff scheme for solving the advection equation**
  - **Demonstrate (numerically) that it is second-order accurate**
  - **Compare (numerically) the order of accuracy with the upwind scheme**
- **Demonstrate that the Crank-Nicolson scheme is unconditionally stable for solving the diffusion equation**
- **Implement the Crank-Nicolson scheme by modifying lec02-demo04.py.**
- **Deadline: March 23 at 11 PM**

References

- Numerical Recipes 3rd Edition: The Art of Scientific Computing
  - https://dl.acm.org/citation.cfm?id=1403886
- Numpy
  - http://www.numpy.org