

**Title:** Lecture - Numerical Methods, PHYS 777

**Speakers:** Erik Schnetter, Dustin Lang

**Collection/Series:** Numerical Methods (Core), PHYS 777-, January 6 - February 5, 2025

**Subject:** Other

**Date:** January 23, 2025 - 10:15 AM

**URL:** <https://pirsa.org/25010060>

## CFL - Condition

Advection Equation

$$\partial_t u(t, x) + \underbrace{a \partial_x u(t, x)}_{\text{velocity}} = 0 \quad (\frac{\partial}{\partial t} - a \frac{\partial}{\partial x}) u = 0$$

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



on

f, by

$$\left( \frac{\partial^2}{\partial t^2} u - \alpha^2 \frac{\partial^2}{\partial x^2} u = 0 \right) \quad | \quad \begin{aligned} O &= \nabla_m J^m \\ &= \nabla_m (\rho_s u^n) \\ &= \partial_t (\rho_s \dot{u}) + \partial_x (\rho_s \gamma_V) \end{aligned}$$
$$\left( \frac{\partial^2}{\partial t^2} - \alpha^2 \frac{\partial^2}{\partial x^2} \right) u = 0 \quad | \quad \begin{aligned} u(t, x) &= \bigcirc \left( \frac{\partial}{\partial t} - \alpha \frac{\partial}{\partial x} \right) \left( \frac{\partial}{\partial t} + \alpha \frac{\partial}{\partial x} \right) u = 0 \quad | \quad V = \text{const} \\ &\Rightarrow \partial_t \rho_s + V \partial_x \rho_s \end{aligned}$$

ε)

$$x_j = j \Delta x$$

$$U_j \equiv U(x_j)$$

$$U(x) = U(x_0) + (x - x_0)U'_0 + \frac{1}{2}(x - x_0)^2 U''_0 + \frac{1}{6}(x - x_0)^3 U'''_0 + \frac{1}{24}(x - x_0)^4 U''''_0$$

$$U_{j+1} = U_j + \Delta x \bar{U}'_j + \frac{1}{2} \Delta x^2 \bar{U}''_j + \frac{1}{6} \Delta x^3 \bar{U}'''_j + \frac{1}{24} \Delta x^4 \bar{U}''''_j + \dots$$

$$U_{j-1} = U_j - \bar{U}'_j + \frac{1}{2} \bar{U}''_j - \frac{1}{6} \bar{U}'''_j + \frac{1}{24} \bar{U}''''_j + \dots$$

$$\text{Want } U_j' \approx \frac{U_{j+1} - U_j}{\Delta x} = U_j' + \frac{1}{2} \Delta x U_j'' + \dots = U_j'$$

$$\text{Want } u_j' \approx \frac{u_{j+1} - u_j}{\Delta x} = u_j' + \frac{1}{2}\Delta x u_j'' + \dots = u_j' + O(\Delta x) \quad \begin{matrix} \text{Forewards Diff} \\ \text{Backwards Diff} \end{matrix}$$

$$u_j' \approx \frac{u_j - u_{j-1}}{\Delta x} = u_j' + O(\Delta x)$$

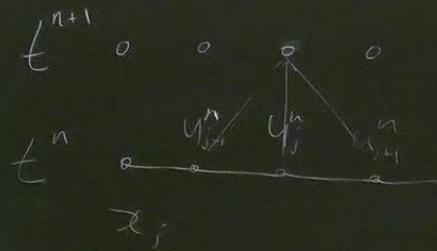
$$u_j' \approx \frac{u_{j+1} - u_{j-1}}{2\Delta x} = u_j' + O(\Delta x^2)$$

Finite Differences

$$t^n = n \Delta t$$

$$U_j^n \equiv u(t^n, x_j)$$

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} + \alpha \frac{U_{j+1}^n - U_{j-1}^n}{2 \Delta x} = 0$$



$$\text{For } \partial_t u \approx \frac{U_j^{n+1} - U_j^n}{\Delta t}$$

$$\text{Eq } \partial_x u \approx \frac{U_{j+1}^n - U_{j-1}^n}{2 \Delta x}$$

$$U_j^{n+1} = U_j^n - \frac{\alpha \Delta t}{2 \Delta x} (U_{j+1}^n - U_{j-1}^n)$$

FTCS



The screenshot shows a Jupyter Notebook interface with the title "advection\_howto". The notebook is running on "localhost:8888/notebooks/advection\_howto.ipynb". The top bar includes standard browser controls like back, forward, and search, along with Jupyter-specific icons for file operations and help.

The notebook content starts with a header "Spatial Grid". It discusses the creation of a "cell-centered" grid from interval  $[a, b]$  into  $N_x$  cells of width  $\Delta x = (b - a)/N_x$ . It mentions  $N_x + 1$  faces between cells and  $N_x$  cell centers at  $x_i = a + (i + 1/2)\Delta x$ . The `make_grid` function also supports  $N_g$  ghost zones on each side of the boundary.

Below this, a code cell [17] contains the implementation of the `make_grid` function:

```
[17]: function make_grid(a, b, Nx, Ng)
    # The desired cell-width
    dx = (b - a) / Nx

    # First get the edges of all the cells, adding ghost zones to the left and right
    xe = LinRange(a - Ng*dx, b + Ng*dx, Nx + 2*Ng + 1)

    # Now the cell centers are just the average of the left & right cell faces
    XL = xe[begin:end-1]
    XR = xe[begin+1:end]
    x = 0.5*(XL + XR)

    return dx, x, xe
end
```

Another code cell [17] shows the definition of the `make_grid` function as a generic function with one method:

```
[17]: make_grid (generic function with 1 method)
```

A sidebar section titled "Plotting Utilities" is expanded, stating: "Just a couple useful functions for plotting our data."

The next section, "You may have to change GLMakie to WGLMakie", contains code for initializing GLMakie and defining plotting functions:

```
[2]: using GLMakie
GLMakie.activate!()

function make_figure()
    fig = Figure()
    ax = Axis(fig[1,1])
    return fig, ax
end

function add_to_plot(ax, x, f)
```

advection\_howto +  
localhost:8888/notebooks/advection\_howto.ipynb 170% Trusted

jupyter advection\_howto Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help JupyterLab Julia 1.11.1

THIS IS A CELL-CENTERED GRID THAT BREAKS UP THE INTERVAL  $[a, b]$  INTO  $N_x$  CELLS OF WIDTH  $\Delta x = (b - a) / N_x$ . THERE ARE  $N_x + 1$  FACES IN BETWEEN THE CELLS.

$x_{e,i} = a + i\Delta x$  AND  $N_x$  CELL CENTERS AT  $x_i = a + (i + 1/2)\Delta x$ . THE `make_grid` FUNCTION ALSO ALLOWS FOR  $N_g$  GHOST ZONES ON EACH SIDE OF THE BOUNDARY, WHICH ARE USEFUL FOR APPLYING BOUNDARY CONDITIONS.

```
[17]: function make_grid(a, b, Nx, Ng)
    # The desired cell-width
    dx = (b - a) / Nx

    # First get the edges of all the cells, adding ghost zones to the left and right
    xe = LinRange(a - Ng*dx, b + Ng*dx, Nx + 2*Ng + 1)

    # Now the cell centers are just the average of the left & right cell faces
    xL = xe[begin:end-1]
    xR = xe[begin+1:end]
    x = 0.5*(xL + xR)

    return dx, x, xe
end

[17]: make_grid (generic function with 1 method)
```

Plotting Utilities

Just a couple useful functions for plotting our data.

The screenshot shows a Jupyter Notebook interface with the title "advection\_howto". The notebook has a "Last Checkpoint: 1 hour ago" message. The menu bar includes File, Edit, View, Run, Kernel, Settings, Help, and Trusted. The toolbar includes icons for file operations like new, open, save, and run. The status bar shows "JupyterLab" and "Julia 1.11.1". A cell output [5] displays the text: "[5]: evolve (generic function with 1 method)".

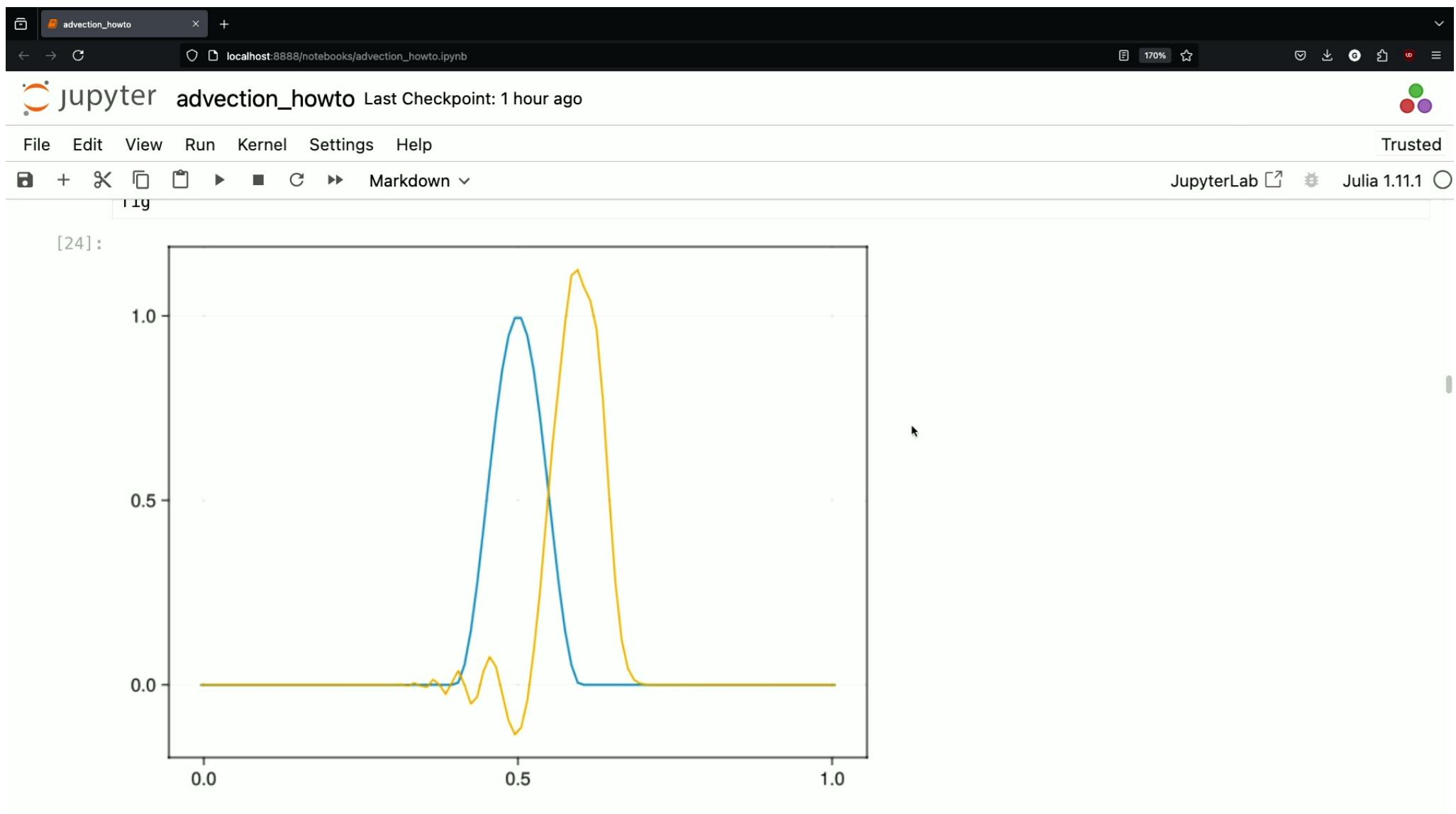
## Per-Cell update function -- where the magic happens!

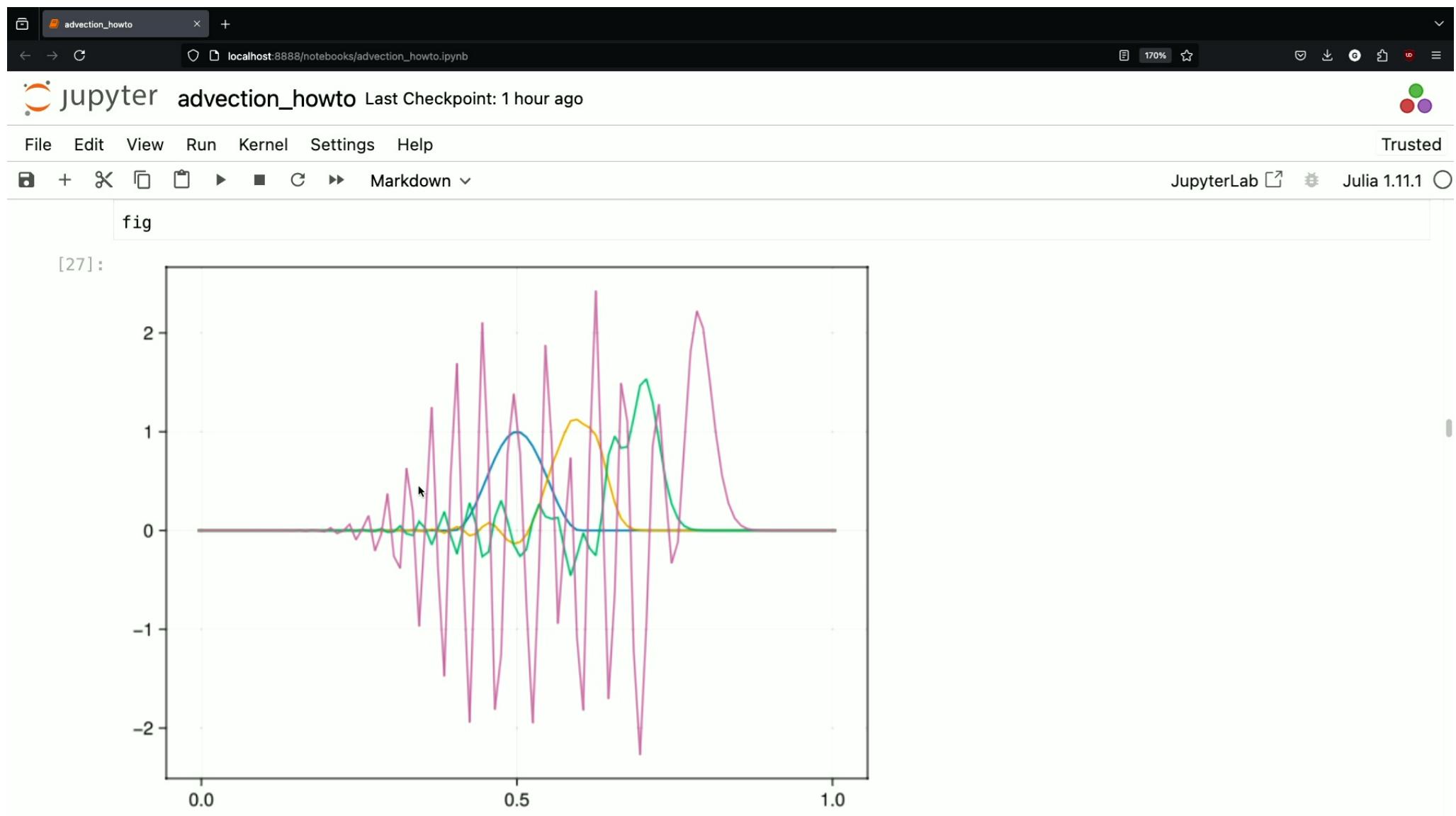
This function gets called on each cell to compute the time-derivative:  $\partial_t u = -a \partial_x u$

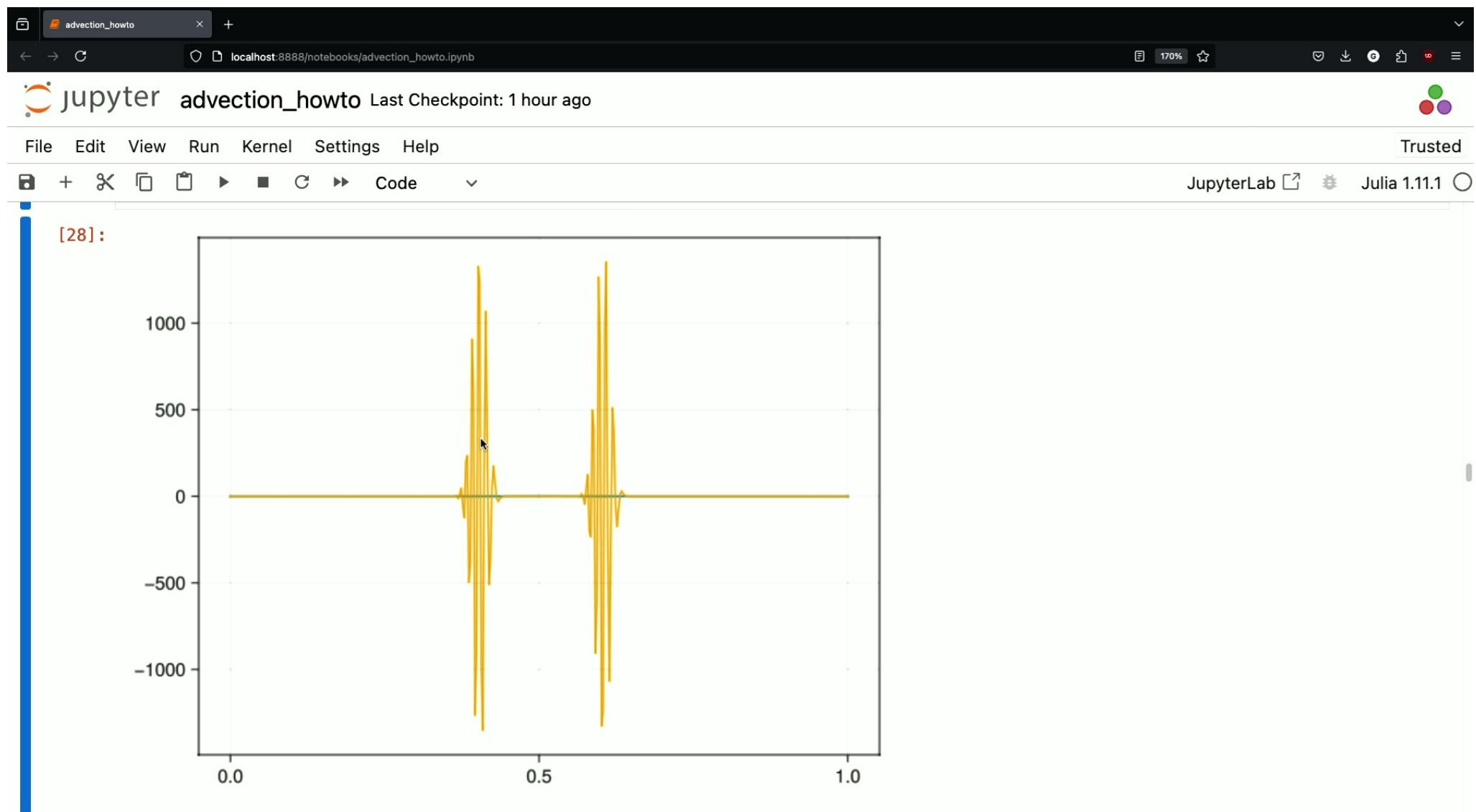
```
[21]: function udot_ftcs(uL, uC, uR, a, dx)
    # Write a function that computes du/dt for the advection equation.
    # uR = u_{j+1}
    # uC = u_j
    # uL = u_{j-1}
    return -a * (uR - uL) / (2*dx)
end

function udot_ftls(uL, uC, uR, a, dx)
    # Write a function that computes du/dt for the advection equation.
    # uR = u_{j+1}
    # uC = u_j
    # uL = u_{j-1}
    return -a * (uC - uL) / (dx)
end
```

[21]: udot\_ftls (generic function with 1 method)







## Son Neumann - Stability Analysis

$$u \rightarrow u + \varepsilon$$

Ask how error  $\varepsilon$  evolves:  $\varepsilon_j^n = A^n e^{ikx_j}$   $|A| \rightarrow ?$

$$A^{n+1} e^{ikx_j} = A^n e^{ikx_j} - \frac{1}{2} C (A^n e^{ik(x_{j+1}-\Delta x)} - A^n e^{ik(x_{j-1}+\Delta x)})$$

$$C \equiv \frac{\partial u}{\partial x} \quad A^{n+1} = A^n (1 - iC \sin(k\Delta x))$$

$$\|A^{n+1}\|^2 = \|A^n\|^2 (1 + C^2 \sin^2(k\Delta x))$$

## Son Neumann Stability Analysis

$$u \rightarrow u + \varepsilon$$

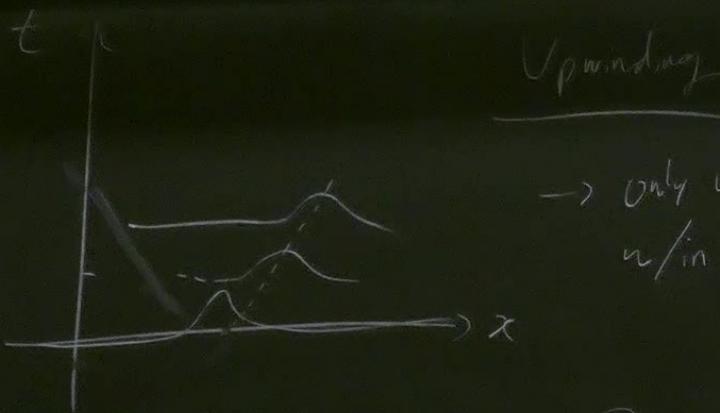
Ask how error  $\varepsilon$  evolves :  $\varepsilon_j^n = A^n e^{ikx_j}$   $|A| \rightarrow ?$

$$A^{n+1} e^{ikx_j} = A^n e^{ikx_j} - \frac{1}{2} C (A^n e^{ik(x_j+\Delta x)} - A^n e^{ik(x_j-\Delta x)})$$

$$A^{n+1} = A^n (1 - iC \sin(k\Delta x))$$

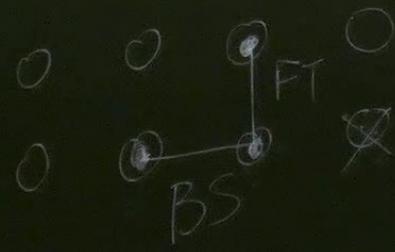
$$|A^{n+1}|^2 = |A^n|^2 (1 + C^2 \sin^2(k\Delta x))$$

$\geq 1$       Unconditionally      Unstable



Upwinding

→ only use information  
w/in light cone



$$U_j^{n+1} = U_j^n - \frac{\alpha \Delta t}{\Delta x} (U_j^n - U_{j-1}^n) \quad FTBS$$

$$c = \frac{\alpha \Delta t}{\Delta x}$$

$$E_j^n = A^n e^{ikx_j}$$

$$\rightarrow |A^n|^2 = |A^n|^2 \left[ 1 - 2c(1-c)(1 - \cos(k\alpha x)) \right] \quad \begin{matrix} \text{CFL Condition} \\ c = \frac{\alpha \Delta t}{\Delta x} < 1 \end{matrix}$$

?>  
?>

advection\_howto × +

localhost:8888/notebooks/advection\_howto.ipynb 170% ☆

jupyter advection\_howto Last Checkpoint: 1 hour ago Trusted

File Edit View Run Kernel Settings Help JupyterLab Julia 1.11.1

We can *upwind* our solution by using a biased estimate for  $\partial_x u$ , only using  $u_{j-1}^n$  and  $u_j^n$ . This gives us the Forward-Time-Left-Space algorithm or FTLS.

FTLS is stable if  $\Delta t < \Delta x/a$ , the Courant-Freidrichs-Lowy (CFL) condition. We can enforce this by setting  $\Delta t$  with the equation:

$$\Delta t = \text{CFL} \times \Delta x/a$$

Where  $\text{CFL} < 1.0$  is a constant.

Let's try FTLS with  $\text{CFL} = 0.9$ ,  $N_x = 100$ , and  $T = 0.3$

```
[10]: dx, x, xe = make_grid(0, 1, 100, 1)
u = init_bump(x, 0.5, 0.1, 1)
fig, ax = make_figure()
add_to_plot(ax, x, u)

a = 1.0

CFL = 0.9
dt = CFL * dx / a

t_run = 0.1

# run to 0.1 & save
evolve(u, t_run, dx, dt, a, udot_ftls, periodic_BC)
```

advection\_howto

localhost:8888/notebooks/advection\_howto.ipynb

jupyter advection\_howto Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Code

# run for 0.25 more (1.0 total) & save  
evolve(u, t\_run, dx, dt, a, udot\_ftls, periodic\_BC)  
add\_to\_plot(ax, x, u)

fig

[30]:

The figure displays four distinct bell-shaped curves, each representing a Gaussian distribution. The curves are colored green, pink, blue, and yellow, and they are positioned side-by-side along a horizontal axis. The vertical axis is labeled 'y' and has tick marks at 0.5 and 1.0. The horizontal axis is labeled 'x'. The curves are perfectly overlapping at their peaks, which are located at the same position on the x-axis.

advection\_howto

localhost:8888/notebooks/advection\_howto.ipynb

jupyter advection\_howto Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Julia 1.11.1

```
evolve(u, t_run, dx, dt, a, udot_ftls, periodic_BC)
add_to_plot(ax, x, u)

# run for 0.25 more (0.75 total) & save
evolve(u, t_run, dx, dt, a, udot_ftls, periodic_BC)
add_to_plot(ax, x, u)

# run for 0.25 more (1.0 total) & save
evolve(u, t_run, dx, dt, a, udot_ftls, periodic_BC)
add_to_plot(ax, x, u)

fig
```

[32]:

The figure displays a time series plot with a horizontal axis representing time and a vertical axis ranging from 0 to 25. The data consists of a series of sharp, vertical blue spikes that occur at regular intervals. Between these spikes, the value is near zero. A small red dot is positioned on the x-axis, just before the second spike. The background of the plot shows a wavy, light-colored line that starts at 0, dips slightly, and then rises towards the end of the plot. A yellow segment is also visible near the right end of the plot.

