(Slide 3: Today's Agenda) "We will explore four key areas, each tackling a different type of problem in hydraulics:"

1. **Root Finding:** Solving implicit equations. **Our Case Study: Finding the normal depth in a trapezoidal channel.**
2. **Numerical Integration:** Calculating total quantities from discrete data. **Our Case Study: Calculating the total volume of a flood hydrograph.**
3. **Solving Ordinary Differential Equations (ODEs):** Computing how a value changes along a single dimension. **Our Case Study: Computing a Gradually Varied Flow water surface profile.**
4. **Solving Partial Differential Equations (PDEs):** Simulating dynamic systems that change in space and time. **Our Case Study: A simplified 1D dam break simulation.**

"For each topic, we'll first cover the theory and the governing equations, and then we'll switch to a Jupyter Notebook to see a live implementation in Python."

Part 1: Root Finding - Solving for Normal Depth (25 minutes)

Theory & Concepts (10-15 minutes)

(Slide 4: The Concept of Normal Depth) "Let's start with a fundamental concept: **Normal Depth (y_n)**. This is the depth of flow in a channel when the flow is **uniform**. Uniform flow means the depth, velocity, and cross-sectional area are constant along the channel. This occurs when the gravitational force driving the flow is perfectly balanced by the frictional resistance from the channel bed and banks."

- **Key Condition:** Bed Slope (S_0) = Water Surface Slope (S_w) = Friction/Energy Slope (S_f).
- The governing equation for this condition is **Manning's Equation**.

(Slide 5: Manning's Equation) "You're all familiar with this equation:" $$ Q = \frac{1}{n} A R_h^{2/3} S_0^{1/2} $$ "Where:

- Q: Discharge (m^3/s) - The volume of water passing a point per second.
- n: Manning's roughness coefficient - Represents the friction of the channel material.
- A: Cross-sectional area (m^2) - A function of depth, $A(y)$.
- P: Wetted perimeter (m) - The length of the channel boundary in contact with water, also a function of depth, $P(y)$.
- R_h: Hydraulic radius (A/P) - A measure of channel efficiency.
- S_0: Bed slope."

(Slide 6: The Implicit Problem) "If we know the discharge Q and the channel properties, our goal is to find the depth, y, that satisfies this equation. This depth is the normal depth, y_n."

"For a simple rectangular channel, we can sometimes rearrange the equation to solve for y directly. But for almost any other shape, like the common **trapezoidal channel**, this is impossible."

- For a trapezoid with bottom width b and side slope z (zH:1V):
  - Area: $A = (b + zy)y$
  - Wetted Perimeter: $P = b + 2y\sqrt{1+z^2}$

"If you substitute these into Manning's equation, you get a very complex equation where y is on both sides and cannot be isolated. This is an **implicit equation**."

(Slide 7: Framing as a Root-Finding Problem) "The standard way to solve this is to rearrange the equation into the form $f(y) = 0$. We are looking for the 'root' of this function."

$$ f(y) = Q_{known} - Q_{calculated}(y) = 0 $$ $$ f(y) = Q - \frac{1}{n} \frac{((b+zy)y)^{5/3}}{(b+2y\sqrt{1+z^2})^{2/3}} S_0^{1/2} = 0 $$ "Our task is now to find the value of y that makes this function equal to zero. Let's look at two algorithms to do this."

(Slide 8: Method 1 - The Bisection Method) "The Bisection Method is the most reliable and intuitive root-finding algorithm."

- **Concept:** Based on the Intermediate Value Theorem. If you have a continuous function and two points, `a` and `b`, where $f(a)$ is positive and $f(b)$ is negative (or vice-versa), then a root *must* exist somewhere between `a` and `b`.
- **Algorithm:**
  1. Start with a bracket `[a, b]` where `f(a)` and `f(b)` have opposite signs.
  2. Calculate the midpoint: `c = (a + b) / 2`.
  3. Evaluate `f(c)`.
  4. If `f(a)` and `f(c)` have opposite signs, the new bracket is `[a, c]`.
  5. Otherwise, the new bracket is `[c, b]`.
  6. Repeat this process, halving the interval each time, until the interval is smaller than your desired tolerance.
- **Pros:** Guaranteed to converge if you can find a valid starting bracket. It's very robust.
- **Cons:** It's relatively slow.

(Slide 9: Method 2 - The Newton-Raphson Method) "The Newton-Raphson method is a much faster, but more temperamental, approach."

- **Concept:** It starts with an initial guess and uses the tangent line of the function at that guess to find the next, better guess.
- **Algorithm (The Formula):** $$ y_{i+1} = y_i - \frac{f(y_i)}{f'(y_i)} $$ "It iteratively refines the guess y_i until the change is negligible or $|f(y_i)|$ is very close to zero."
- **The Derivative, $f'(y)$:** This method requires the function's derivative. The derivative of our Manning function is very messy to calculate by hand. So, we use a practical shortcut: a **numerical derivative**. The central difference formula is a good choice: $$ f'(y) \approx \frac{f(y+h) - f(y-h)}{2h} $$ (where h is a very small number, like 1e-6).
- **Pros:** Extremely fast convergence (quadratic) when it's close to the root.
- **Cons:** Can fail spectacularly! It might diverge if the initial guess is poor, or fail if the derivative is zero or close to zero.

Practical Implementation (10 minutes)

Teacher's Guide: Switch to the Jupyter Notebook. Execute the cells one by one.

1. **Introduce Libraries:** Briefly explain the roles of `numpy` (for math) and `matplotlib` (for plotting). Run the first code cell.
2. **Problem Setup:**

  - Go to the `Problem Setup` code cell under "Root Finding".
  - "Here, we define all our known parameters for the trapezoidal channel: Q, n, slope, width, etc."
  - "Next, we define our function `f_manning`. This is the Python version of the $f(y) = 0$ equation we just discussed. It takes a depth `y` and calculates the difference between the target Q and the calculated Q."
  - Execute this cell.
3. **Bisection Method:**
   - "First, let's try the Bisection Method. We need to give it a starting bracket. Let's guess the depth is between 0.1m and 10m."
   - Walk through the `bisection_method` function line by line, explaining how it matches the algorithm on the slide (checking signs, calculating `c`, updating `a` or `b`).
   - Execute the cell. "You can see it converged in a number of iterations and gives us a normal depth of about 2.93 m."
4. **Newton-Raphson Method:**
   - "Now for the faster Newton-Raphson method. It only needs one initial guess. Let's try 2.0m."
   - Show the `numerical_derivative` function first. "This is our practical way of finding the slope of the function."
   - Then, walk through the `newton_raphson_method` function, highlighting the iterative update formula: `y = y - f_y / fp_y`.
   - Execute the cell. "Look at that! It converged in just a few iterations. This is why it's preferred when it works."
5. **The 'Real-World' Way:**
   - "In a real project, you wouldn't write these from scratch. You'd use a professionally written, highly-optimized library like SciPy."
   - Execute the `scipy.optimize.newton` cell. "As you can see, we get the same result instantly. But now you know the logic behind what this function is doing."

Part 2: Numerical Integration - Flood Volume (15 minutes)

Theory & Concepts (5-7 minutes)

(Slide 10: The Problem - Total Flood Volume) "A common task for a hydrologist is to analyze a flood. We often get data from a river gauge in the form of a **hydrograph** - a plot of discharge (Q) over time (t)." "The question is: what was the total volume of water that passed the gauge during the flood event?"

- **The Mathematical Formulation:** The total volume V is the area under the hydrograph curve. Mathematically, it's the integral of discharge with respect to time: $$ V = \int_{t_{start}}^{t_{end}} Q(t) ,dt $$

(Slide 11: The Challenge & The Solution) "The challenge is that we don't have a neat function $Q(t)$. We have a set of discrete data points (t_i, Q_i) from our measurements." "So, how do we find the area? We approximate it. The simplest and most common method for this is the **Trapezoidal Rule**."

- **Concept:** We connect each pair of adjacent data points with a straight line and calculate the area of the trapezoid formed underneath. The total volume is simply the sum of the areas of all these small trapezoids.
- **Formula:**

    - Area of one trapezoid: $A_i = \frac{Q_i + Q_{i+1}}{2} \times (t_{i+1} - t_i)$
    - Total Volume: $V = \sum_{i=0}^{N-1} A_i$
- **A CRITICAL NOTE ON UNITS!** "Discharge is usually in meters-cubed-per-**second** (m^3/s), but time in a hydrograph is often given in **hours** or **days**. You *must* convert your time units to match your discharge units (i.e., seconds) before calculating, or your answer will be wrong by a factor of 3600 or more!"

Practical Implementation (8 minutes)

Teacher's Guide: Switch back to the notebook.

1. **Problem Setup:**
    - Go to the "Numerical Integration" section and the `Problem Setup` cell.
    - "Here we have some sample hydrograph data: time in hours and the corresponding discharge."
    - Execute the cell to show the plot. "This is our hydrograph. We want to find the total blue area under this curve."
2. **Trapezoidal Rule Implementation:**
    - Show the `trapezoidal_rule` function. "This function implements the exact logic we just discussed. It loops through the data points, calculates the area of each trapezoid (`avg_height * dx`), and adds it to a running total."
    - Move to the next cell. "First, and most importantly, we convert our time array from hours to seconds by multiplying by 3600."
    - "Then, we call our function to calculate the volume."
    - Execute the cell. "It gives us a total volume of about 7.5 million cubic meters."
3. **The NumPy Way:**
    - "Again, in practice, we use libraries. NumPy has a built-in `trapezoid` function that does this for us."
    - Point to the `np.trapezoid` line. "Notice how much simpler it is. We give it the y-values (discharge) and the corresponding x-values (time in seconds)."
    - Execute the line. "The result is identical. Our from-scratch function was correct, but the NumPy version is faster and less prone to coding errors."

Part 3: Ordinary Differential Equations - GVF Profile (25 minutes)

Theory & Concepts (10-15 minutes)

(Slide 12: Gradually Varied Flow (GVF)) "Now we move to a more complex problem. **Gradually Varied Flow** describes flow that is steady (not changing with time) but **non-uniform** (depth changes along the length of the channel). Think of the water surface profile behind a dam or as water approaches a waterfall."

(Slide 13: The GVF Governing Equation) "The change in depth (y) with respect to distance along the channel (x) is described by this first-order Ordinary Differential Equation (ODE):" $$ \frac{dy}{dx} = \frac{S_0 - S_f}{1 - Fr^2} $$ "Let's break this down:

- dy/dx: The slope of the water surface. This is what we want to find.
- S_0: The bed slope (a constant).
- S_f: The **friction slope**. This is the energy loss due to friction at a given depth, and we calculate it using Manning's equation: $S_f = \left(\frac{nQ}{AR_h^{2/3}}\right)^2$. Notice it depends on depth y.

- Fr: The **Froude number**, $Fr = \frac{V}{\sqrt{gD}}$, which represents the ratio of inertial forces to gravitational forces. It also depends on depth.
  - $Fr < 1$: Subcritical flow (deep, slow)
  - $Fr > 1$: Supercritical flow (shallow, fast)
  - $Fr = 1$: Critical flow. The denominator becomes zero, and the equation breaks! This is the location of a hydraulic jump or drop."

(Slide 14: Solving the ODE - An Initial Value Problem) "To solve this, we need a starting point. This is called an **Initial Value Problem** (or in this context, a Boundary Value Problem). We know the depth at one location (e.g., at a dam), and we want to compute the depth at all other locations." "We will do this by 'marching' step-by-step along the channel."

"**Example: The M1 Backwater Curve** Imagine a dam raises the water to 5m. We know that at $x=0$ (the dam), $y=5$m. We want to find the water profile *upstream*. This means we will take steps in the negative x-direction, so our step size, Δx, will be negative."

(Slide 15: Method 1 - Euler's Method) "This is the simplest possible way to solve an ODE."

- **Concept:** If we know the point (x_i, y_i) and the slope at that point (dy/dx), we can just extrapolate along that slope for a short distance, Δx, to find the next point, y_{i+1}.
- **Formula:** $$ y_{i+1} = y_i + \Delta x \cdot f(x_i, y_i) $$ (where $f(x_i, y_i)$ is the GVF equation).
- **Issue:** It's like taking a step in the direction you're currently facing. If the path curves, you'll start to drift off. It's simple but can be inaccurate and accumulate errors quickly.

(Slide 16: Method 2 - 4th Order Runge-Kutta (RK4)) "RK4 is the classic, reliable workhorse for solving ODEs. It's much more accurate than Euler."

- **Concept:** Instead of using just one slope at the beginning of the step, RK4 cleverly calculates a weighted average of four different slopes at different points within the step (the beginning, two in the middle, and the end). This allows it to "see" the curvature of the function and make a much better prediction.
- **The Formulas:** (Briefly show the k-formulas, but focus on the concept) $k_1 = f(y_i)$ (slope at start) $k_2 = f(y_i + ...)$ (first midpoint slope) $k_3 = f(y_i + ...)$ (second midpoint slope) $k_4 = f(y_i + ...)$ (slope at end) $y_{i+1} = y_i + \frac{\Delta x}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ "The key takeaway is that it's a much smarter way of averaging the slope over the interval, which dramatically improves accuracy."

Practical Implementation (10 minutes)

Teacher's Guide: Back to the notebook.

1. **Problem Setup:**
   - Go to the "GVF Profile" section and its `Problem Setup` cell.
   - "We define the parameters for our rectangular channel and the GVF flow."
   - "Then we create the `dydx_gvf` function. This is the heart of it all - it's the Python implementation of the GVF equation, $\frac{S_0 - S_f}{1 - Fr^2}$."
   - Execute this cell.
2. **Euler's Method:**
   - "We'll start at the downstream end ($x=0$) where we know the depth is 5m. We want to compute the profile 5km upstream, so we'll go to $x=-5000$m using a step size of -50m."

- Show the `euler_solver` function, pointing to the simple update line: `y[i+1] = y[i] + dx * f(y[i])`.
- Execute the cell to run the solver and plot the result. "Here is the backwater profile calculated with Euler's method. We've plotted the water surface elevation (WSE), which is the bed elevation plus the water depth."

3. **RK4 Method:**
    - "Now, let's solve the same problem with the more accurate RK4 method."
    - Walk through the `rk4_solver` function. "You can see the implementation is more complex. Here are the four `k` values being calculated, and here is the final weighted average to get the next `y`."
    - Execute the cell to run the RK4 solver and show the comparison plot.
    - "Look at the comparison. The blue solid line is RK4 and the dashed line is Euler. You can see that Euler's method over-estimates the water surface elevation upstream. For the same step size, RK4 gives a more physically accurate result."
4. **The Pro Tool:** "As always, for professional work, you'd use a function like `scipy.integrate.solve_ivp`, which is even more robust and can automatically adjust its step size."

Part 4: Partial Differential Equations - 1D Dam Break (30 minutes)

Of course. Here is the revised and more detailed version of Part 4 of the lecture notes, focusing on the theory behind the Saint-Venant equations and the Lax-Friedrichs scheme.

Part 4: Partial Differential Equations - 1D Unsteady Flow (30 minutes)

Theory & Concepts (15-20 minutes)

(Slide 17: The Challenge - Unsteady, Non-Uniform Flow) "We now arrive at the most comprehensive topic: modeling flow where conditions change in both **space (x)** and **time (t)**. This is the domain of unsteady, non-uniform flow. The classic, dramatic example is the propagation of a wave after a dam fails."

"To model this, we need equations that describe how mass and momentum change at every point and every instant. These are the **Saint-Venant Equations**."

(Slide 18: The Saint-Venant Equations (Full Form)) "The Saint-Venant equations are a pair of coupled, non-linear, hyperbolic partial differential equations. They are derived by applying the principles of conservation of mass and conservation of momentum to a small control volume of fluid in an open channel."

1. **Conservation of Mass (Continuity Equation):** $$ \frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 $$

    - $\frac{\partial A}{\partial t}$: The rate of change of the cross-sectional area at a point in time. This represents how much water is being stored or released at that location.
    - $\frac{\partial Q}{\partial x}$: The spatial gradient of the discharge. This represents the difference between the flow entering and leaving a small segment of the channel.
    - **In words:** "The rate at which water level rises or falls is equal to the net flow into that section."

2. **Conservation of Momentum (Momentum Equation):** $$ \underbrace{\frac{\partial Q}{\partial t}}*{\text{Local Accel.}} + \underbrace{\frac{\partial (\beta Q^2/A)}{\partial x}}*{\text{Convective Accel.}} +

\underbrace{gA \frac{\partial y}{\partial x}}*{\text{Pressure Force}} = \underbrace{gA (S_0 - S_f)}*
{\text{Gravity & Friction}} $$

- **This is Newton's Second Law (F=ma) for a fluid element.**
- **Left Side (The "ma" part):** Represents the total acceleration of the fluid. It has two components:
  - *Local Acceleration*: How velocity changes with time at a fixed point.
  - *Convective Acceleration*: How velocity changes because the fluid is moving to a new location with a different velocity.
- **Right Side (The "F" part):** Represents the sum of forces acting on the fluid.
  - *Pressure Force*: Caused by differences in water depth upstream and downstream.
  - *Gravity Force*: The component of the fluid's weight acting along the channel slope (S_0).
  - *Friction Force*: The resistive force from the channel bed and banks, represented by the friction slope (S_f).

(Slide 19: Simplifying for our Dam Break Problem) "To make the problem tractable for this tutorial, we will make several simplifying assumptions, common for the classic dam-break scenario:"

1. The channel is a wide rectangle, so we can analyze flow *per unit width*. We replace Area A with depth h and Discharge Q with discharge-per-unit-width $q=hu$.
2. The channel is horizontal ($S_0=0$).
3. The channel is frictionless ($S_f=0$).
4. The velocity profile is uniform ($\beta=1$).

"With these assumptions, our equations simplify significantly."

(Slide 20: The Conservative Vector Form - The Key Idea) "The simplified equations can be elegantly written in a single **conservative vector form**: $$ \frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} = \mathbf{S} $$ "This form is fundamental to modern numerical methods. Let's define each part:"

- $\mathbf{U}$: The vector of **Conserved Variables**. These are the quantities whose total amount in a closed system does not change. $$ \mathbf{U} = \begin{bmatrix} h \ hu \end{bmatrix} = \begin{bmatrix} \text{mass per unit length (volume)} \ \text{momentum per unit length} \end{bmatrix} $$

- $\mathbf{F}(\mathbf{U})$: The vector of **Fluxes**. A flux is the rate of transport of a conserved quantity across a boundary. It tells us how much of $\mathbf{U}$ is moving past a point per unit time. $$ \mathbf{F}(\mathbf{U}) = \begin{bmatrix} hu \ hu^2 + \frac{1}{2}gh^2 \end{bmatrix} = \begin{bmatrix} \text{flux of mass (discharge)} \ \text{flux of momentum} \end{bmatrix} $$

  - *Momentum Flux Breakdown:* The momentum flux, $hu^2 + \frac{1}{2}gh^2$, has two parts:
    1. hu^2: The momentum being physically carried (advected) by the moving water.
    2. $\frac{1}{2}gh^2$: The momentum transferred by the pressure force exerted by the water.

- $\mathbf{S}$: The vector of **Source Terms**. These are terms that add or remove the conserved quantity from the system. In our simplified case, we have no gravity slope or friction, so this term is zero. $$ \mathbf{S} = \begin{bmatrix} 0 \ 0 \end{bmatrix} $$ (In the full equation, this would be $\mathbf{S} = \begin{bmatrix} 0 \ gh(S_0 - S_f) \end{bmatrix}$).

(Slide 21: The Finite Difference Method - Discretizing the World) "To solve this on a computer, we must discretize it. We create a grid in space and time."

- **Space:** The channel is divided into `Nx` points, with index `j`, separated by Δx.
- **Time:** The simulation proceeds in steps of Δt, with index `n`.
- **Our Goal:** Find the state $\mathbf{U}_j^{n+1}$ at every point `j` for the next time level, given we know all the states $\mathbf{U}_j^{n}$ at the current time level.

(Slide 22: The Lax-Friedrichs Scheme - A Detailed Look) "The Lax-Friedrichs scheme is an explicit method for updating our solution. Let's build it from the governing equation:" $$ \frac{\partial \mathbf{U}}{\partial t} = - \frac{\partial \mathbf{F}}{\partial x} $$ "We approximate the derivatives:"

1. **Time Derivative (Forward Difference):** $\frac{\partial \mathbf{U}}{\partial t} \approx \frac{\mathbf{U}_j^{n+1} - \mathbf{U}_j^{n}}{\Delta t}$
2. **Spatial Derivative (Central Difference):** $\frac{\partial \mathbf{F}}{\partial x} \approx \frac{\mathbf{F}*{j+1}^{n} - \mathbf{F}*{j-1}^{n}}{2 \Delta x}$

"If we combine these naively, we get: $\mathbf{U}*j^{n+1} = \mathbf{U}j^{n} - \frac{\Delta t}{2 \Delta x} (\mathbf{F}{j+1}^{n} - \mathbf{F}*{j-1}^{n})$. This scheme is famously **unconditionally unstable** and will blow up!"

"**The Lax-Friedrichs 'Trick'**: The instability arises from not properly accounting for wave propagation in both directions. The solution is to replace the term $\mathbf{U}*j^{n}$ with the average of its neighbors: $$ \mathbf{U}j^{n} \rightarrow \frac{1}{2}(\mathbf{U}{j+1}^{n}* + *\mathbf{U}*{j-1}^{n}) $$ "This averaging introduces what is called **numerical diffusion** or **numerical viscosity**. It has the effect of smearing or smoothing the solution, which is what stabilizes the scheme."

"Substituting this 'trick' into our unstable scheme gives us the **Lax-Friedrichs Scheme**:" $$ \frac{\mathbf{U}*j^{n+1} - \frac{1}{2}(\mathbf{U}*{j+1}^{n} + \mathbf{U}*{j-1}^{n}))}{\Delta t} = - \frac{\mathbf{F}*{j+1}^{n} - \mathbf{F}*{j-1}^{n}}{2 \Delta x} $$ "Rearranging to solve for our unknown, $\mathbf{U}j^{n+1}$, gives the final update formula you see in the code:" $$ \mathbf{U}j^{n+1} = \underbrace{\frac{1}{2}(\mathbf{U}{j+1}^{n} + \mathbf{U}{j-1}^{n})}*{\text{Averaging/Diffusion Term}} - \underbrace{\frac{\Delta t}{2 \Delta x}(\mathbf{F}*{j+1}^{n} - \mathbf{F}*{j-1}^{n})}_{\text{Flux Difference Term}} $$

(Slide 23: The CRITICAL Stability Rule - The CFL Condition) "This scheme is now stable, but only under one critical condition: The **Courant-Friedrichs-Lewy (CFL) Condition**."

- **Physical Meaning:** The numerical grid must be fine enough to 'capture' the physical process. Information in the fluid (a wave) cannot travel more than one grid cell (Δx) in a single time step (Δt). If it 'jumps' over a grid cell, the numerical solution can't see it, leading to instability.

- **Wave Speed:** The fastest speed at which information travels in the shallow water system is the absolute fluid velocity plus the wave celerity: $|u| + c$, where $c=\sqrt{gh}$.

- **The Formula:** We must ensure the Courant number, C, is less than or equal to 1. $$ C = \frac{(|u|*{max} + c*{max}) \Delta t}{\Delta x} \le 1 $$

- **The Practical Implication: Adaptive Time-Stepping.** This rule is paramount. It means Δt is not a constant. In every single step of our simulation loop, we must:

  1. Scan the entire domain to find the current maximum values of $|u|$ and c.

    2. Use these to calculate the maximum allowable Δt for this step to satisfy the CFL condition (usually with a safety factor, e.g., $C \le 0.5$). This ensures our simulation remains stable as flow conditions change.

Practical Implementation (10-15 minutes)

Teacher's Guide: Switch back to the notebook and proceed with the Part 4 implementation as planned. You can now connect the code much more directly to the detailed theory.

1. **Problem Setup:** Connect the initial `h` and `u` arrays directly to the `U` vector concept.
2. **Main Loop:**
    - Explicitly point out the **CFL block** at the top of the loop. "This is us calculating the maximum allowable `dt` for this step, just like we discussed."
    - Point to the **Flux block**. "Here, we are calculating the `F` vector—the flux of mass `hu` and the flux of momentum `hu²/h + 0.5gh²`."
    - Point to the **Update block**. "And this is the heart of it: the direct implementation of the Lax-Friedrichs formula. You can see the averaging term `0.5 * (h_old[j+1] + h_old[j-1])` and the flux difference term `(dt / (2 * dx)) * (F1[j+1] - F1[j-1])`."
3. **Visualization:** When showing the animation, re-emphasize the effect of the numerical diffusion. "See how the wave front is smooth and not a sharp vertical line? That's the effect of the averaging term in the Lax-Friedrichs scheme, which was necessary for stability but comes at the cost of some physical accuracy at sharp gradients."