

Homework Schedule

- Because of where we are in the class, I am cutting one homework assignment
- There will be 6 total
- Homework #5 is assigned today (and fits on one side of the paper!)
- Homework #6 will be assigned in 2 weeks, and will cover both diffusion and computational fluid dynamics
 - It won't be the length of 2 assignments—don't worry

Summary of PDEs (so far...)

- Hyperbolic
 - Think: advection
 - Real, finite speed(s) at which information propagates—carries changes in the solution
 - Second-order explicit methods are robust
- Elliptic
 - Think: Poisson's equation
 - Solution sees the boundaries and source instantaneously
 - Relaxation is the basic method
 - Multigrid accelerates relaxation

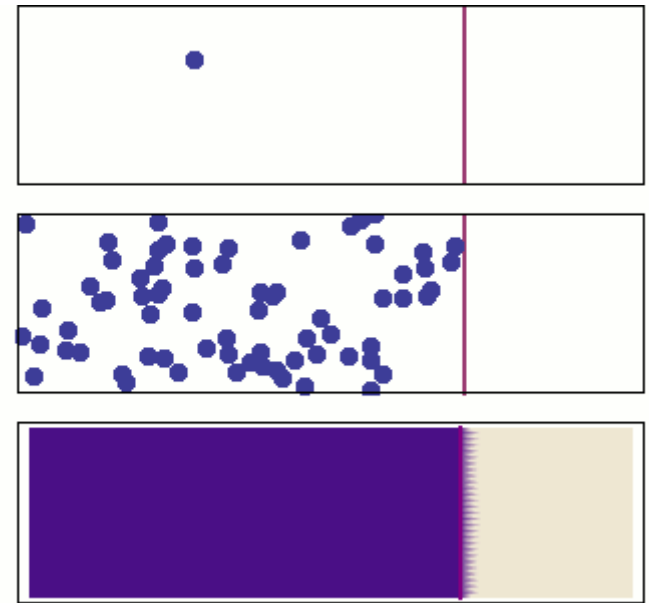
Parabolic Equations

- The prototypical parabolic equation is:
 - $\phi_t = k\phi_{xx}$
 - This represents diffusion
- The solution is time-dependent (unlike elliptic PDEs)

Physical Systems

- **Diffusion:**
 - Fick's law says that the diffusive flux of a quantity is proportional to the negative of its gradient
 - high-concentrations flow toward low concentrations
 - $F = -k\phi_x$
 - Time-evolution is given by the divergence of the flux (think conservation):

$$\phi_t = [-k\phi_x]_x$$

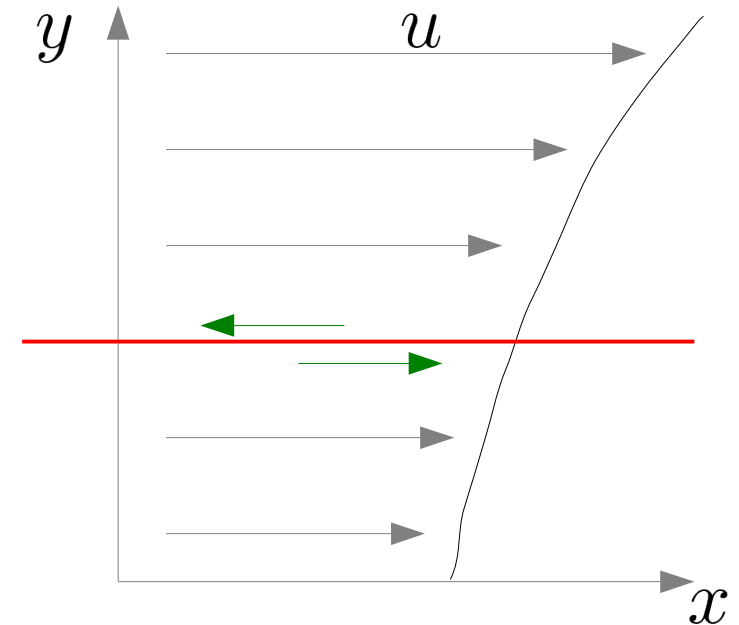


(Sbyrnes321/Wikipedia)

Physical Systems

- **Viscosity** (see Tritton Ch. 1,2,5):

- Viscous stresses oppose relative movements between neighboring fluid particles—redistributes momentum
- Consider the flow on the right
 - Faster fluid above the plane drags slower fluid below ahead
 - Slower fluid below pulls upper fluid behind
 - Forces equal and opposite, and proportional to normal gradient
- Net force on a fluid element is



$$\left(\mu \left(\frac{\partial u}{\partial y} \right)_{y+\delta y} - \mu \left(\frac{\partial u}{\partial y} \right)_y \right) \delta x = \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \delta x \delta y$$

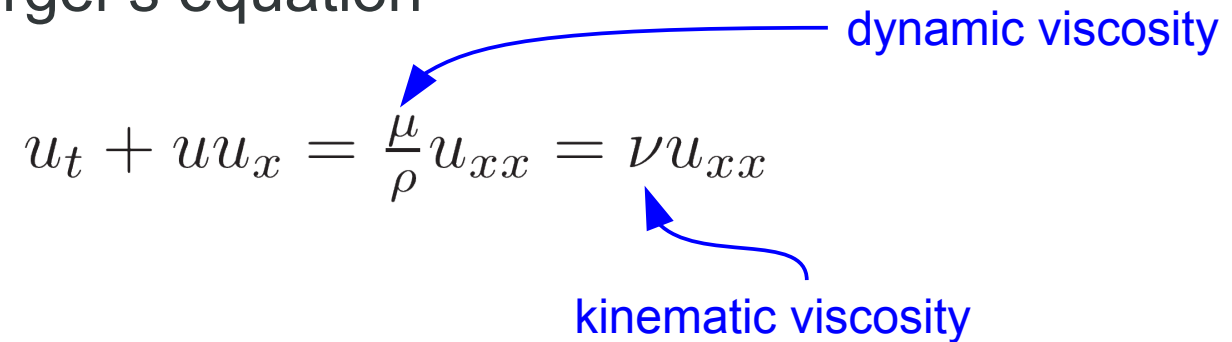
Physical Systems

- This needs to be accounted for in velocity equation
 - e.g. viscous Burger's equation

$$u_t + uu_x = \frac{\mu}{\rho} u_{xx} = \nu u_{xx}$$

dynamic viscosity

kinematic viscosity

A blue arrow points from the text 'dynamic viscosity' to the fraction $\frac{\mu}{\rho}$ in the equation. Another blue arrow points from the text 'kinematic viscosity' to the symbol ν in the equation.

Multiphysics

- So far, we've been looking at each type of equation on its own
- Real systems often mix the various types of PDEs
 - E.g. Hydrodynamics with viscosity combines hyperbolic and diffusive PDEs
 - Incompressible flow with viscosity combines all 3 types
- We can also have non-zero sources on these equations
- We'll look at various examples of this next lecture to see how to piece together everything we looked at so far

Test Problem

- As usual, when developing methods, it is essential to have a test problem with known solution to gauge the accuracy of the method
 - Determining that you are solving the equations correctly is called **verification**
 - A related concept is determining that the equations you are solving actually represent the system you are modeling. This is **validation**.
 - Here, for instance, you may compare to experiments.
- Verification and Validation (V&V) is a very big field in computational science

Test Problem

- Consider a Gaussian profile:

$$\phi^a(x, t) = (\phi_2 - \phi_1) \left(\frac{t_0}{t + t_0} \right)^{1/2} e^{-\frac{1}{4}(x-x_c)^2 / k(t+t_0)} + \phi_1$$

- Inserting this into our model parabolic PDE:

$$\phi_t = k\phi_{xx}$$

demonstrates that this is a solution (blackboard...)

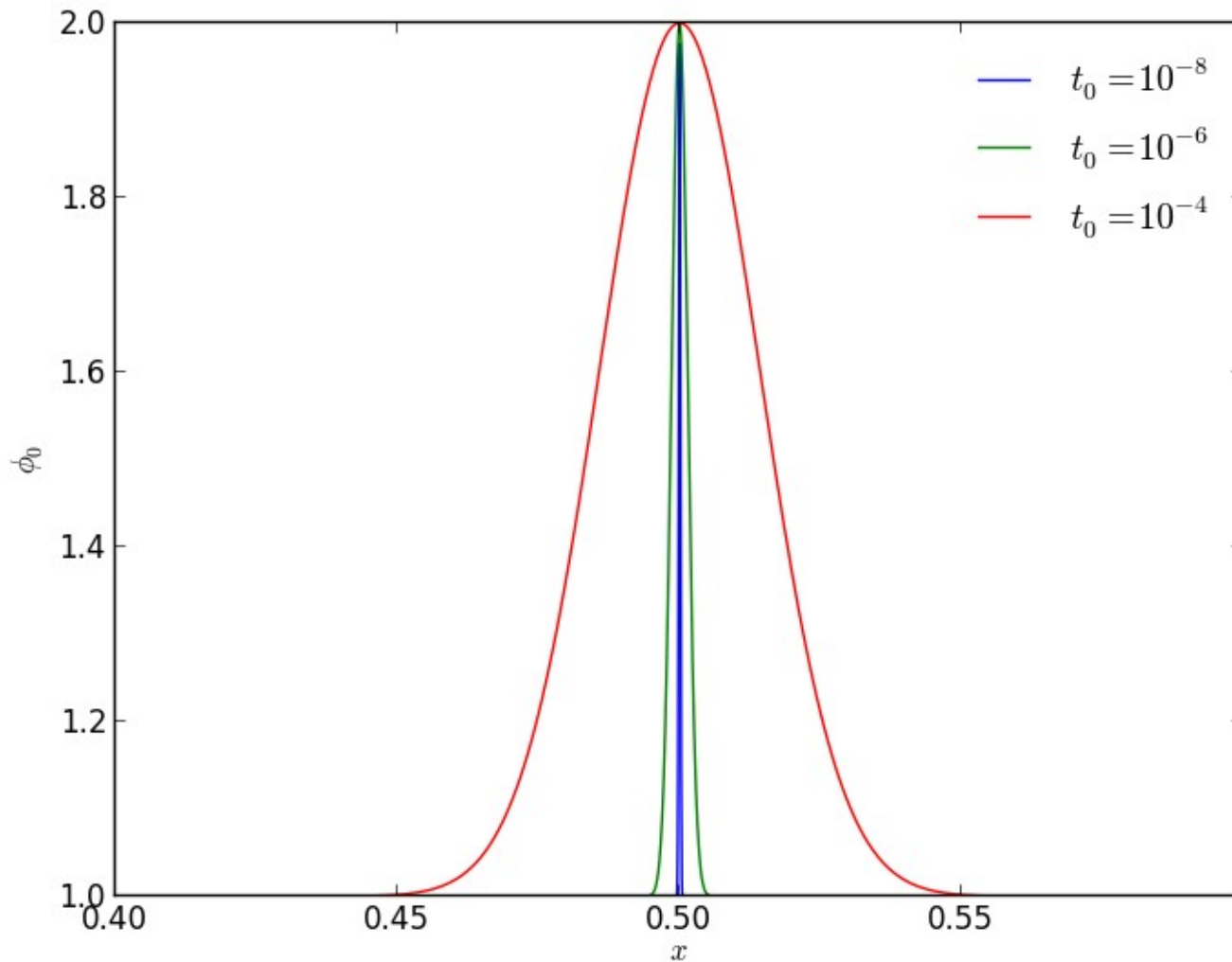
- Here t_0 is a small time—in the limit that $t_0 \rightarrow 0$, we get a delta-function initial condition
- The diffusion of a Gaussian is a Gaussian
 - Amplitude and width change—it spreads out with time

Test Problem

- We'll use the parameters:
 - $x \in [0,1]$
 - Neumann Bcs
 - $k = 1$
 - $\phi_1 = 1, \phi_2 = 2$
 - $t_0 = 10^{-4}$
 - Stopping time: $5 \cdot 10^{-3}$ s

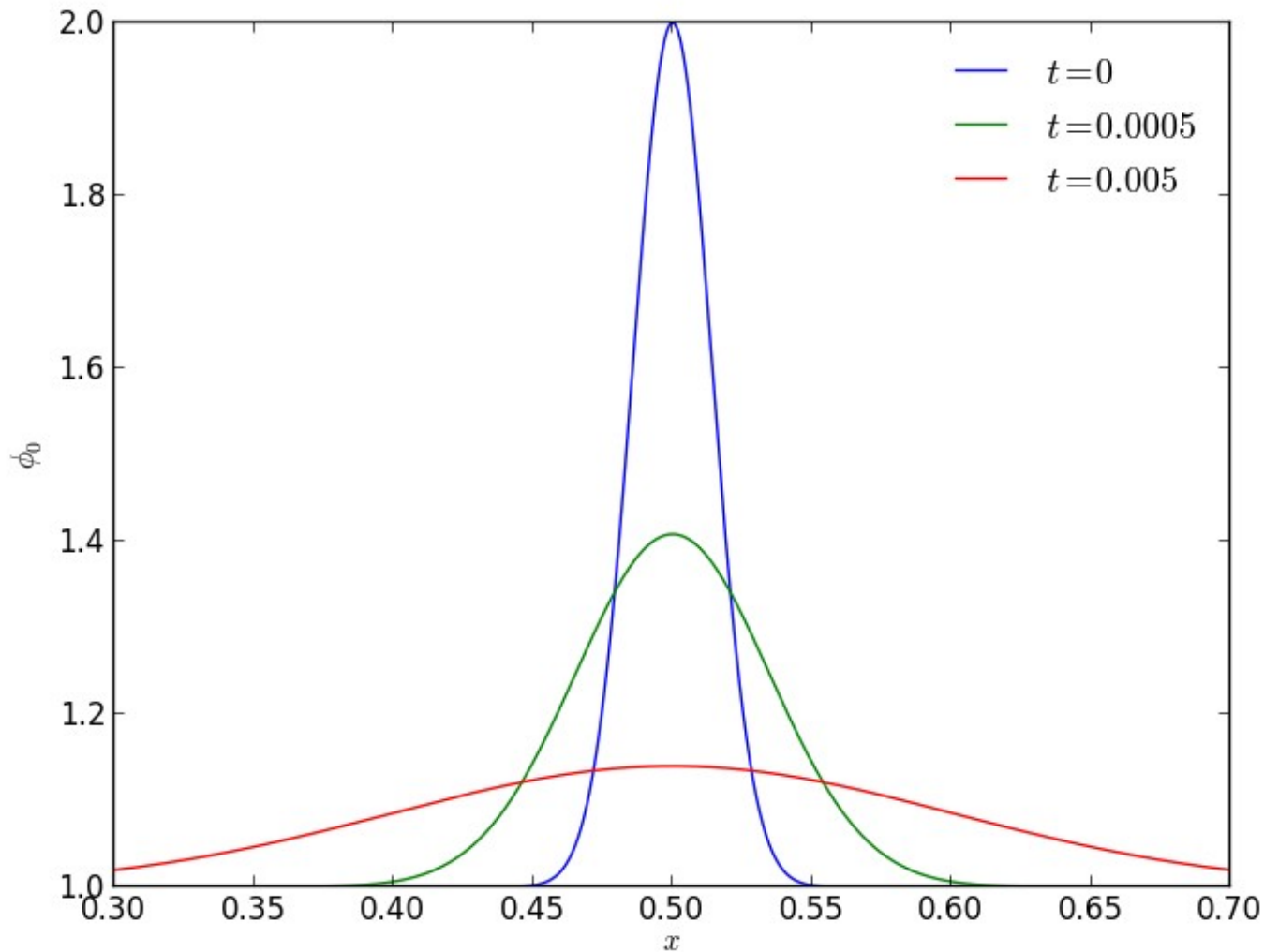
Test Problem

- Note how narrow the initial profile can become—it must be resolved



Test Problem

- Solution is to simply spread out the initial profile with time



Explicit Methods

- Let's consider an explicit finite-difference discretization
 - Replace the time derivative with a forward-difference (first order)
 - Replace the spatial second derivative with our standard centered second-derivative (second order):

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = k \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2}$$

- This is reminiscent of the FTCS method we did for advection (which blew up!)
- Is this stable?

Stability Analysis

- Note that our equation is linear—different Fourier modes will not interact
 - We can assess the stability of this discretization in the same fashion as we did for the advection equations

- Test solution: single Fourier mode:

$$\phi_i^n = A^n e^{i\theta j} \quad j = \sqrt{-1}$$

- Recall, stability requires that:

$$\left| \frac{A^{n+1}}{A^n} \right| \leq 1$$

- Result:

$$\Delta t \leq \frac{1}{2} \frac{\Delta x^2}{k}$$

Blackboard derivation...

Stability Analysis

- Note that this timestep constraint $\propto \Delta x^2$ whereas for advection it was $\propto \Delta x$
 - Very restrictive!
 - As we increase our resolution, the constraint for diffusion will become more restrictive faster
 - For multiphysics problems, this may result in us wanting to treat diffusion implicitly and advection explicitly for the same problem (more on that later...)

Implementation

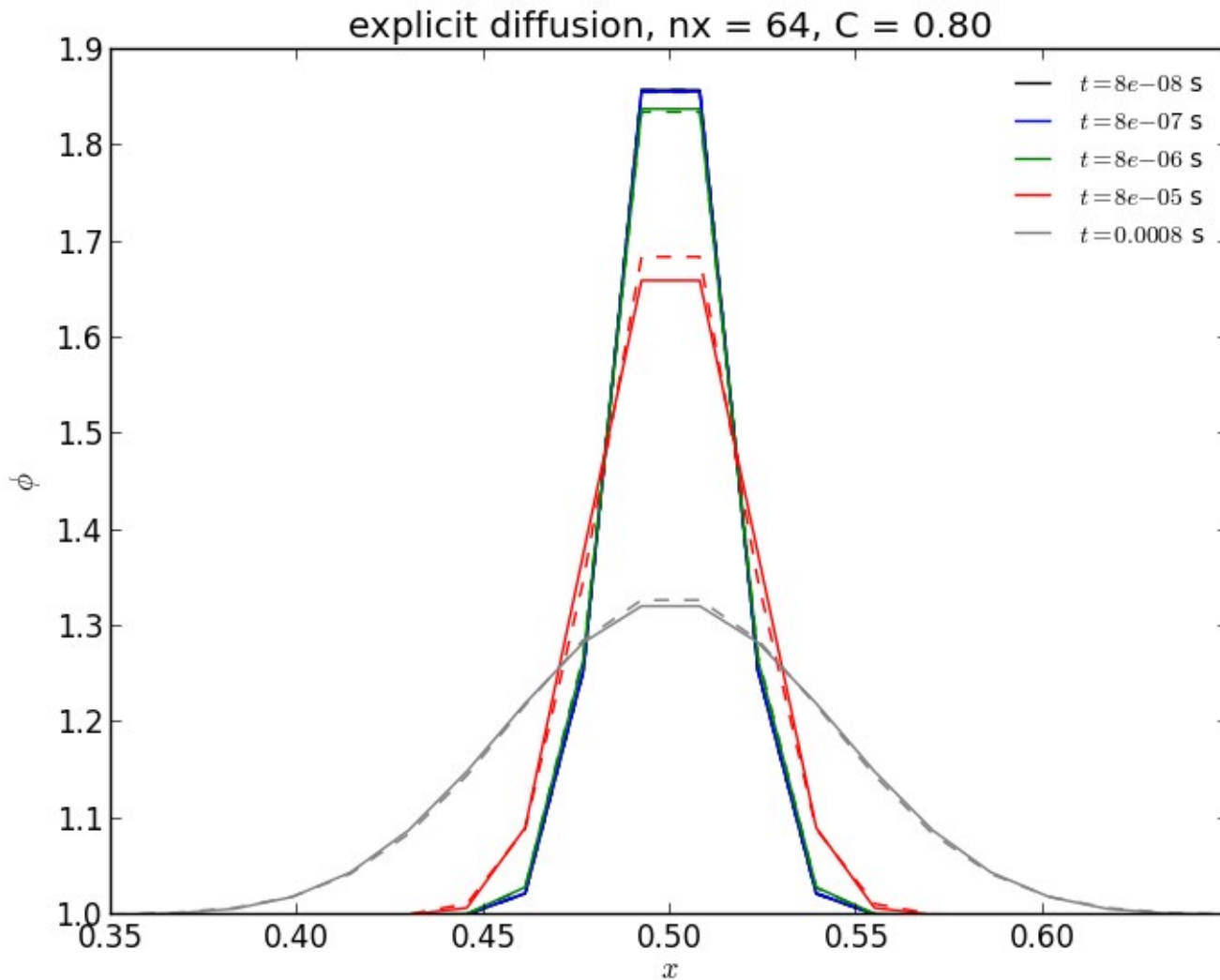
- The diffusion equation is second-order in space—two boundary conditions are needed
 - Note: unlike the Poisson equation, the boundary conditions don't immediately “pollute” the solution everywhere in the domain—there is a timescale associated with it
- Characteristic timescale (dimensional analysis):

$$t_{\text{diffuse}} \sim \frac{L^2}{k}$$

- We want to make sure to avoid the boundaries in our comparison to the analytic solution
- We can define a “Courant number” for this problem, and set the timestep as:

$$\Delta t = C \frac{1}{2} \frac{\Delta x^2}{k}$$

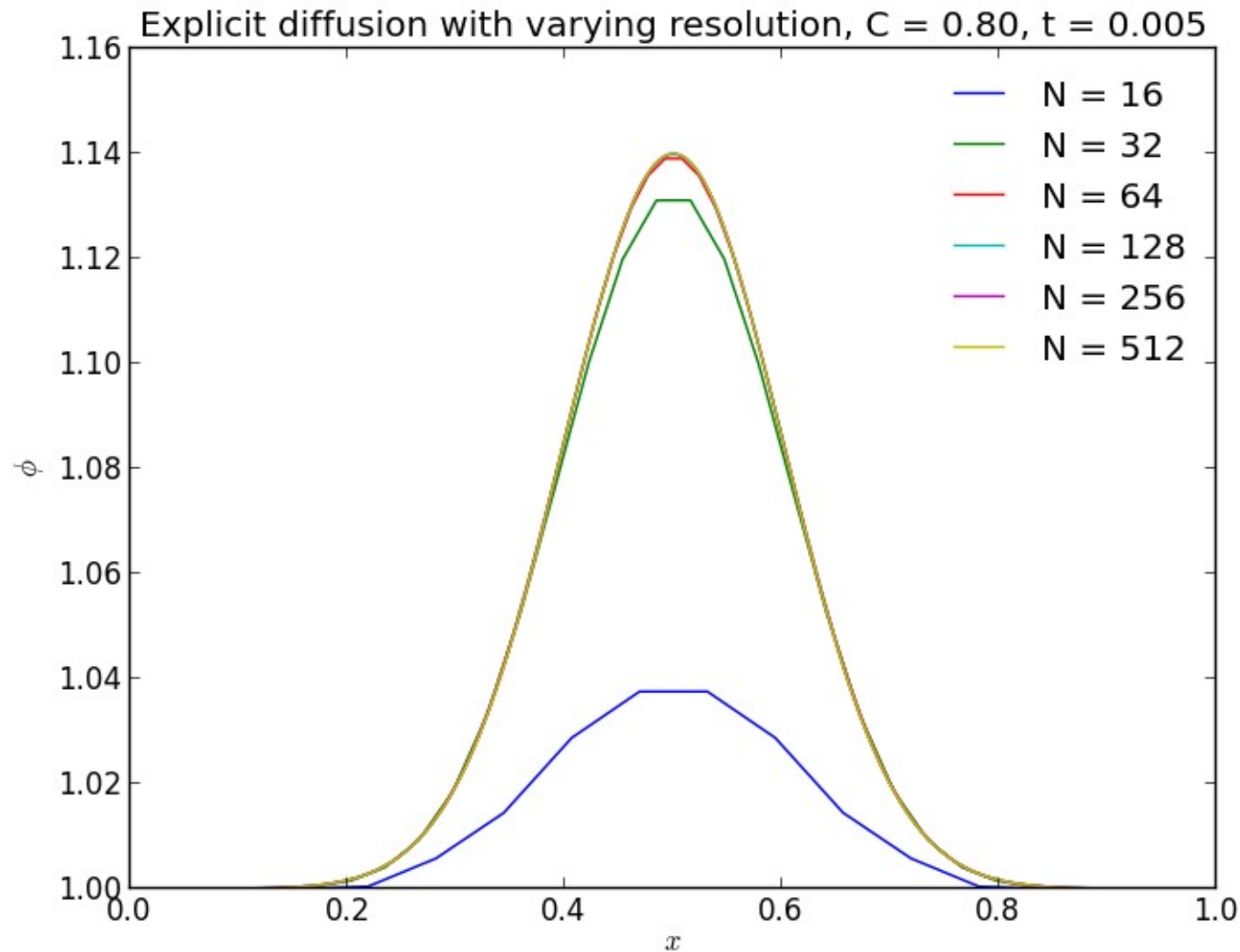
Solution Time-Evolution



(Dashed line is the analytic solution)

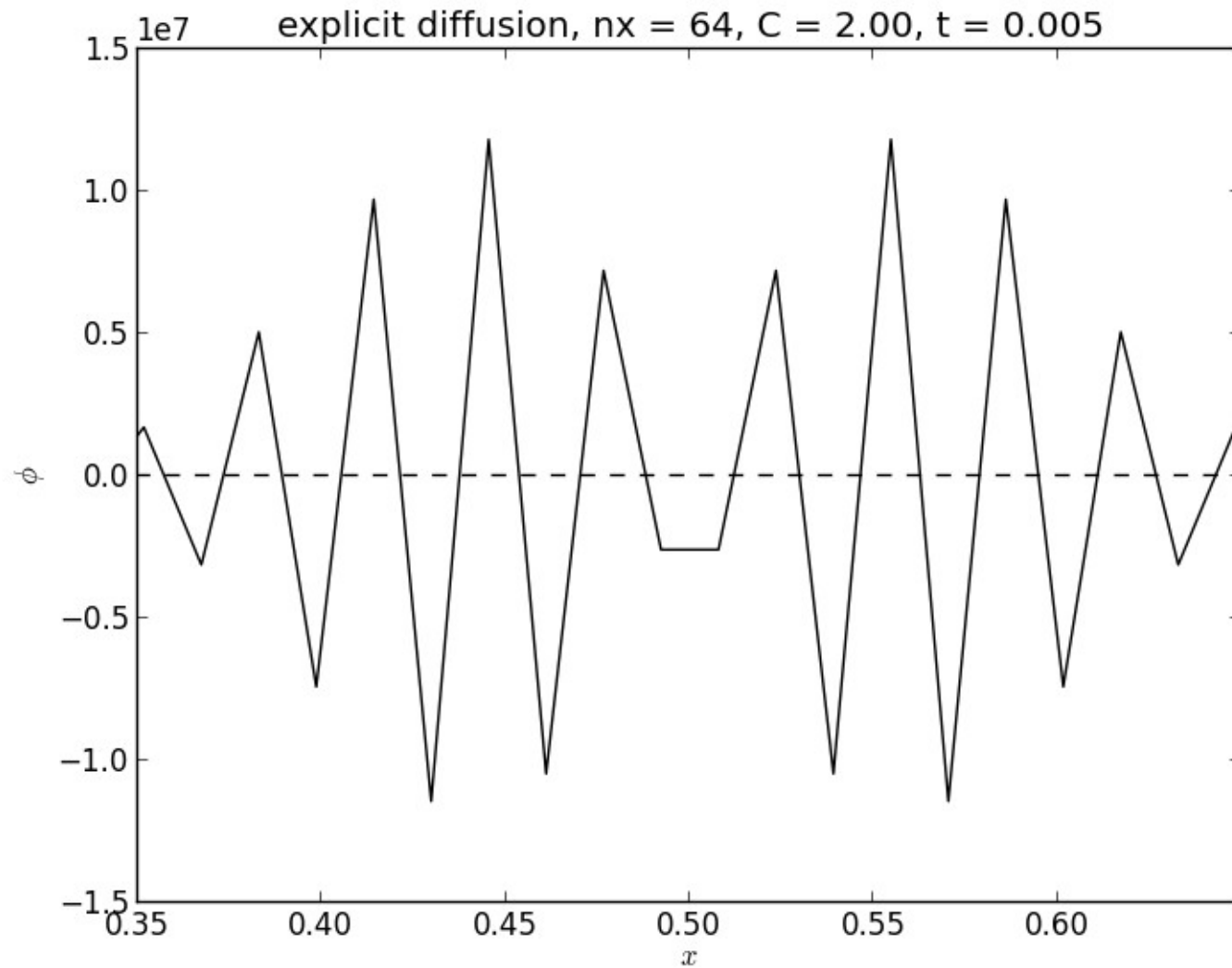
Solution with Resolution

- If the initial conditions are not resolved, the solution is bad



Exceed Timestep Constraint

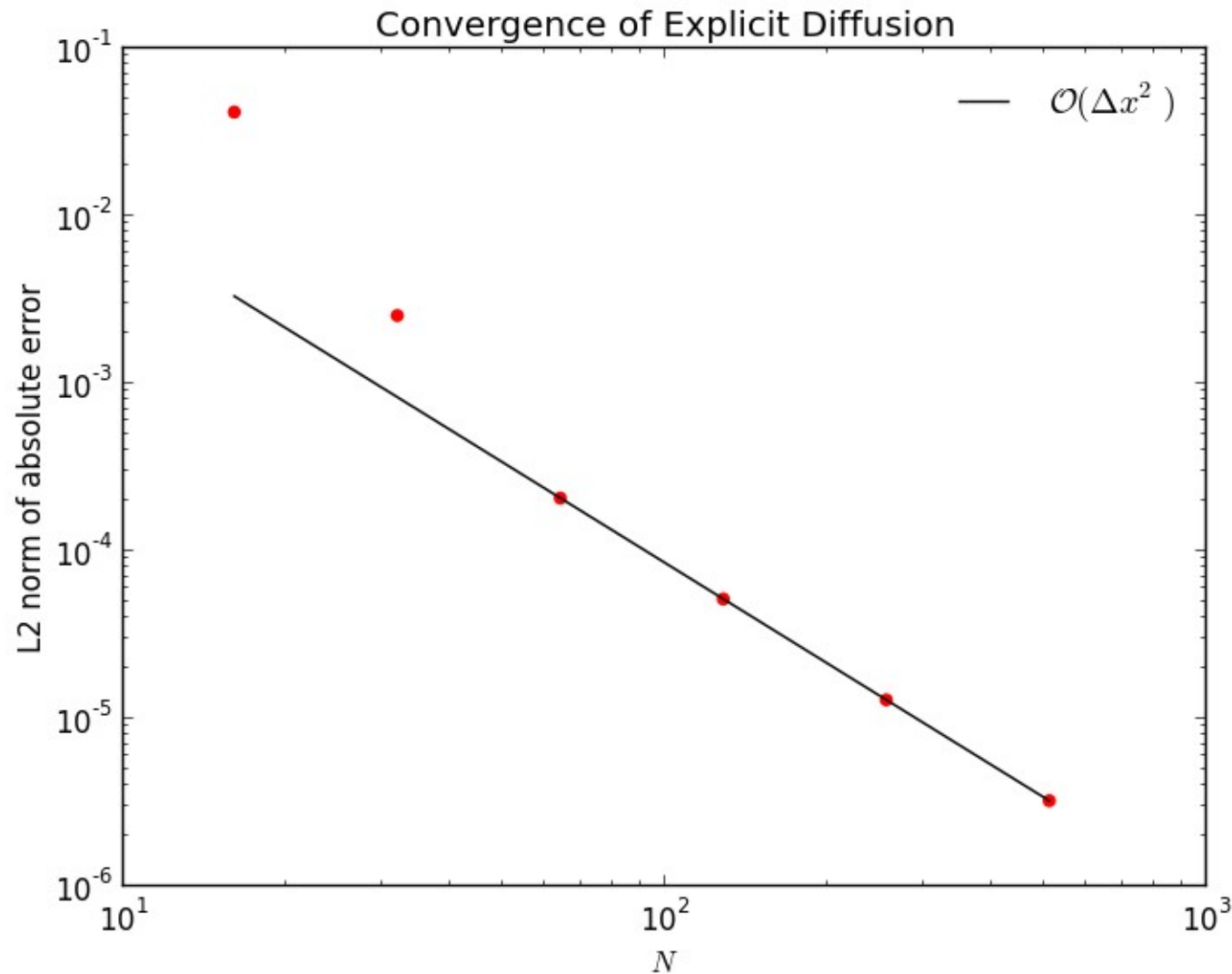
- When you go beyond the timestep constraint, bad things happen...



Performance

- With advection, when we decreased Δx , we decreased Δt in proportion
 - This let us see the convergence in time and space together
- With explicit diffusion, when we decrease Δx by 2, Δt drops by 4
 - The truncation error in the time-discretization will be hard to pick out from the spatial discretization.

Performance



At small N the narrowness of the initial conditions makes the error large—we are not sufficiently resolving them

First-Order Implicit Discretization

- The timestep constraint for explicit diffusion is very restrictive
 - Most times, we will discretize this implicitly to get around this constraint
- Consider a first-order (in time) discretization:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = k \frac{\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}}{\Delta x^2}$$

- Update becomes:

$$-\alpha\phi_{i-1}^{n+1} + (1 + 2\alpha)\phi_i^{n+1} - \alpha\phi_{i+1}^{n+1} = \phi_i^n$$

- This is a coupled set of algebraic equations—a linear system

First-Order Implicit Discretization

- Boundary conditions—consider the left boundary:

- **Neumann:** $\phi_{\text{lo}-1}^{n+1} = \phi_{\text{lo}}^{n+1}$

- Update for first zone inside boundary:

$$(1 + \alpha)\phi_{\text{lo}}^{n+1} - \alpha\phi_{\text{lo}+1}^{n+1} = \phi_{\text{lo}}^n$$

- **Dirichlet:** $\phi_{\text{lo}-1}^{n+1} = \phi_{\text{lo}}^{n+1}$

- Update for the first zone inside boundary:

$$(1 + 3\alpha)\phi_{\text{lo}}^{n+1} - \alpha\phi_{\text{lo}+1}^{n+1} = \phi_{\text{lo}}^n$$

- Similar constructions at the other boundary

First-Order Implicit Discretization

- Matrix form (Neumann BCs at both ends):

$$\begin{pmatrix} 1 + \alpha & -\alpha & & & \\ -\alpha & 1 + 2\alpha & -\alpha & & \\ & -\alpha & 1 + 2\alpha & -\alpha & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & \ddots & \ddots \\ & & & & -\alpha & 1 + 2\alpha & -\alpha \\ & & & & & -\alpha & 1 + \alpha \end{pmatrix} \begin{pmatrix} \phi_{\text{lo}}^{n+1} \\ \phi_{\text{lo}+1}^{n+1} \\ \phi_{\text{lo}+2}^{n+1} \\ \vdots \\ \vdots \\ \phi_{\text{hi}-1}^{n+1} \\ \phi_{\text{hi}}^{n+1} \end{pmatrix} = \begin{pmatrix} \phi_{\text{lo}}^n \\ \phi_{\text{lo}+1}^n \\ \phi_{\text{lo}+2}^n \\ \vdots \\ \vdots \\ \phi_{\text{hi}-1}^n \\ \phi_{\text{hi}}^n \end{pmatrix}$$

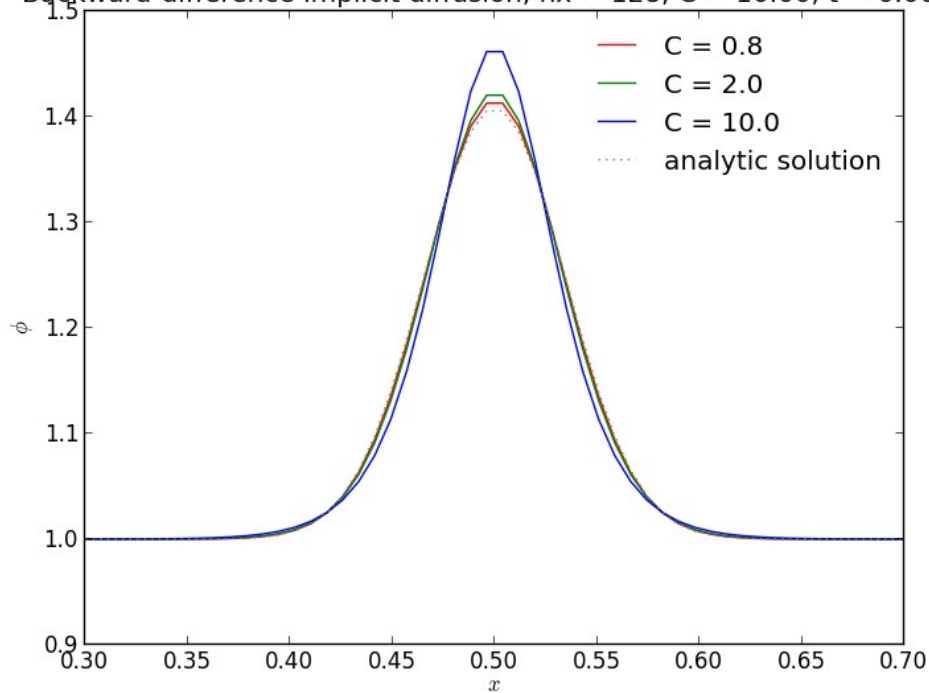
- Tridiagonal matrix—this is pretty efficient to solve
- Note that the ghost cells do not explicitly appear in this system
- Diagonally dominant—iterative methods will work as well

Direct Solve

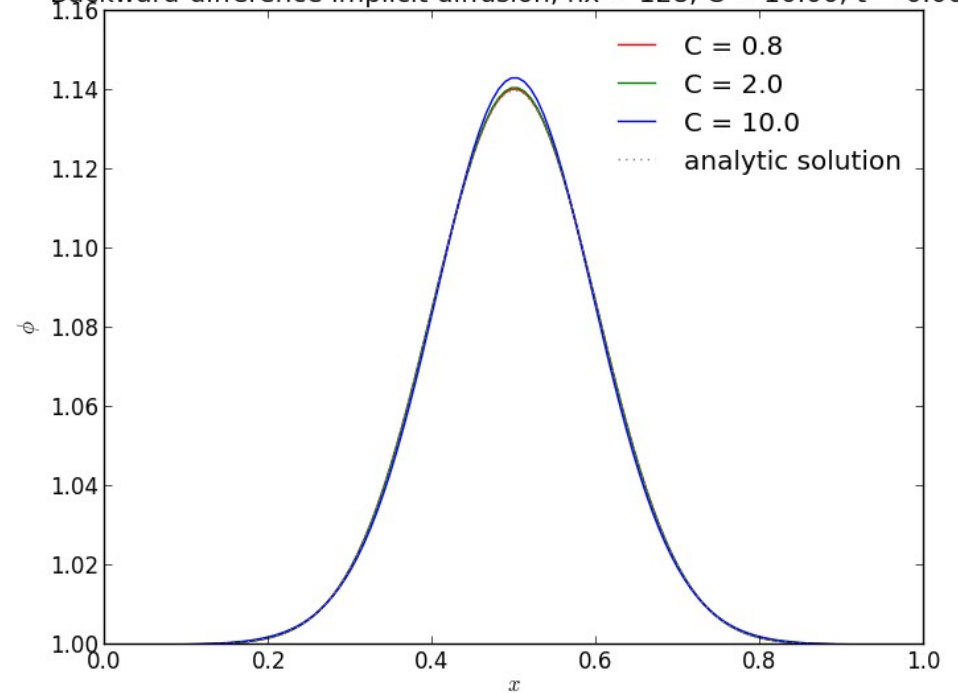
- We can solve this using our linear algebra techniques
 - Let's look at the code...

Direct Solve

Backward-difference implicit diffusion, $n_x = 128$, $C = 10.00$, $t = 0.0005$



Backward-difference implicit diffusion, $n_x = 128$, $C = 10.00$, $t = 0.005$



Notice that we stay stable as we exceed $C = 1$. Also recall stability and accuracy are two separate things...

Crank-Nicolson

- Let's go second-order in space and time
- Recall that a difference approximation to a first-derivative is second-order accurate if we center the difference about the point at which the derivative is taken
- Second order (in space and time):

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{k}{2} ([L\phi]_i^n + [L\phi]_i^{n+1})$$

- Here L is a discrete approximate to the Laplacian
- The RHS is centered in time, making the time-difference second order
- This discretization is called **Crank-Nicolson**

Crank-Nicolson

- We can write this in matrix for (blackboard...)

$$\begin{pmatrix} 1 + \frac{\alpha}{2} & -\frac{\alpha}{2} & & & \\ -\frac{\alpha}{2} & 1 + \alpha & -\frac{\alpha}{2} & & \\ & -\frac{\alpha}{2} & 1 + \alpha & -\frac{\alpha}{2} & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & \ddots & \ddots \\ & & & & -\frac{\alpha}{2} & 1 + \alpha & -\frac{\alpha}{2} \\ & & & & & -\frac{\alpha}{2} & 1 + \frac{\alpha}{2} \end{pmatrix} \begin{pmatrix} \phi_{lo}^{n+1} \\ \phi_{lo+1}^{n+1} \\ \phi_{lo+2}^{n+1} \\ \vdots \\ \vdots \\ \phi_{hi-1}^{n+1} \\ \phi_{hi}^{n+1} \end{pmatrix} = \begin{pmatrix} \phi_{lo}^n + \frac{k\Delta t}{2} [L\phi]_{lo}^n \\ \phi_{lo+1}^n + \frac{k\Delta t}{2} [L\phi]_{lo+1}^n \\ \phi_{lo+2}^n + \frac{k\Delta t}{2} [L\phi]_{lo+2}^n \\ \vdots \\ \vdots \\ \phi_{hi-1}^n + \frac{k\Delta t}{2} [L\phi]_{hi-1}^n \\ \phi_{hi}^n + \frac{k\Delta t}{2} [L\phi]_{hi}^n \end{pmatrix}$$

This can be solved in pretty much the same way.

Note: for multidimensions, this array can become very large ($N_x \times N_y$ squared).

MG Solution

- We can also solve our C-N discretized diffusion equation with multigrid
- Time-discretized PDE is:

$$\phi_i^{n+1} - \frac{\Delta t}{2} k [L\phi]_i^{n+1} = \phi_i^n + \frac{\Delta t}{2} k [L\phi]_i^n$$

- This is an elliptic Helmholtz equation:

$$(A - B\nabla^2)\phi = f$$

– with

$$A = 1 \quad B = \frac{\Delta t}{2} k \quad f = \phi^n + \frac{\Delta t}{2} k [L\phi]^n$$

MG Solution

- Recall for multigrid we need a smoothing operation and a method to compute the residual.

- Smoothing:

$$\phi_i \leftarrow \left(f_i + \frac{B}{\Delta x^2} [\phi_{i+1} + \phi_{i-1}] \right) / \left(A + \frac{2B}{\Delta x^2} \right)$$

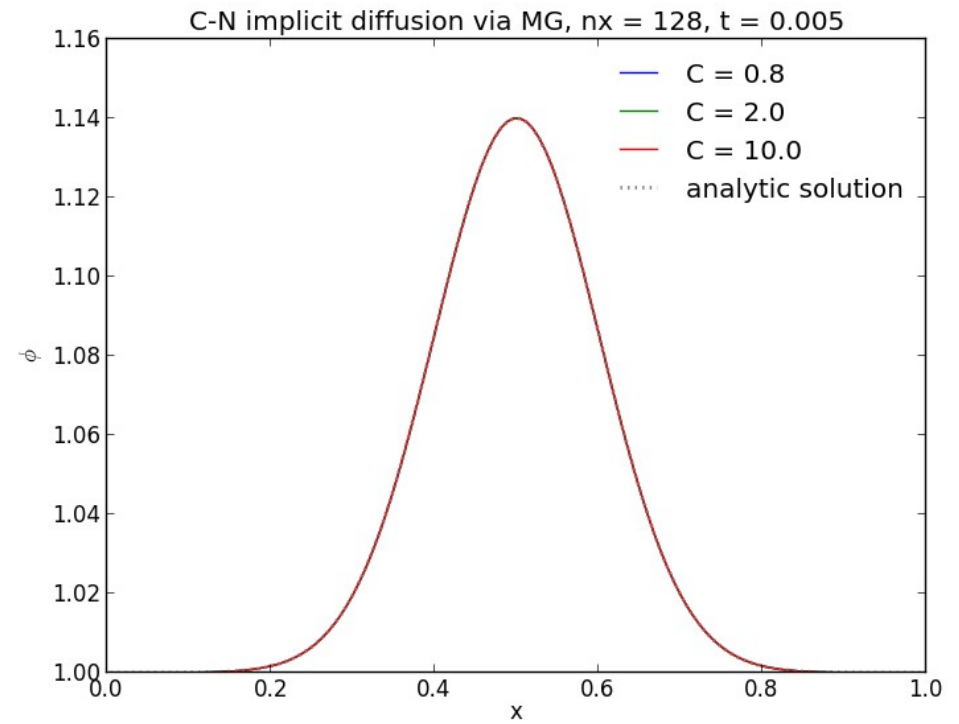
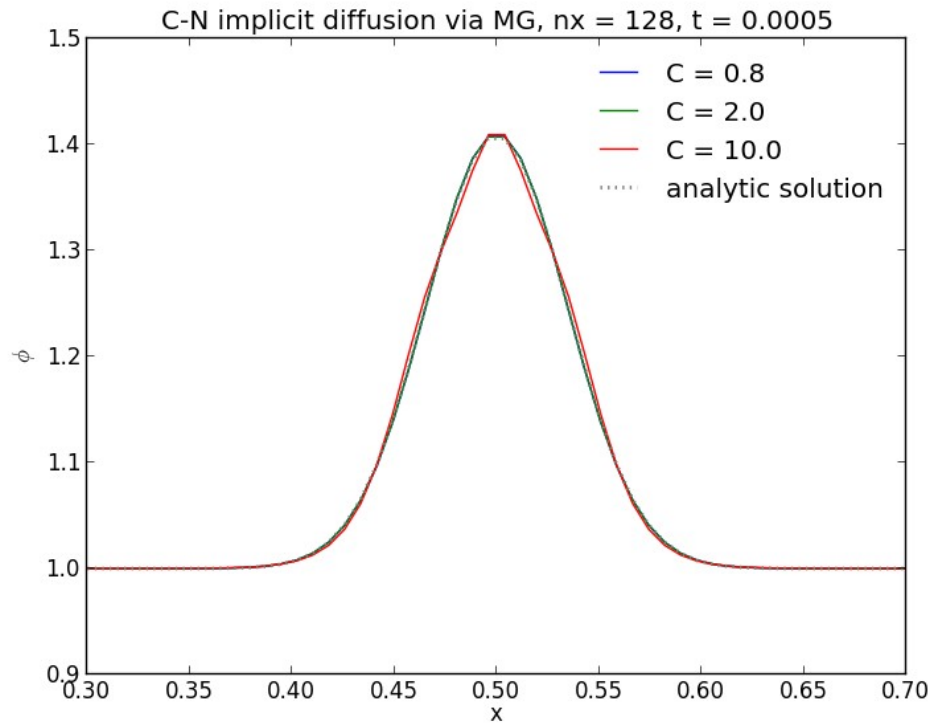
- Only slightly more complicated than the Poisson problem
- Still use Gauss-Seidel

- Residual:

$$r_i = f_i - A\phi_i + B[L\phi]_i$$

- Let's look at the code...

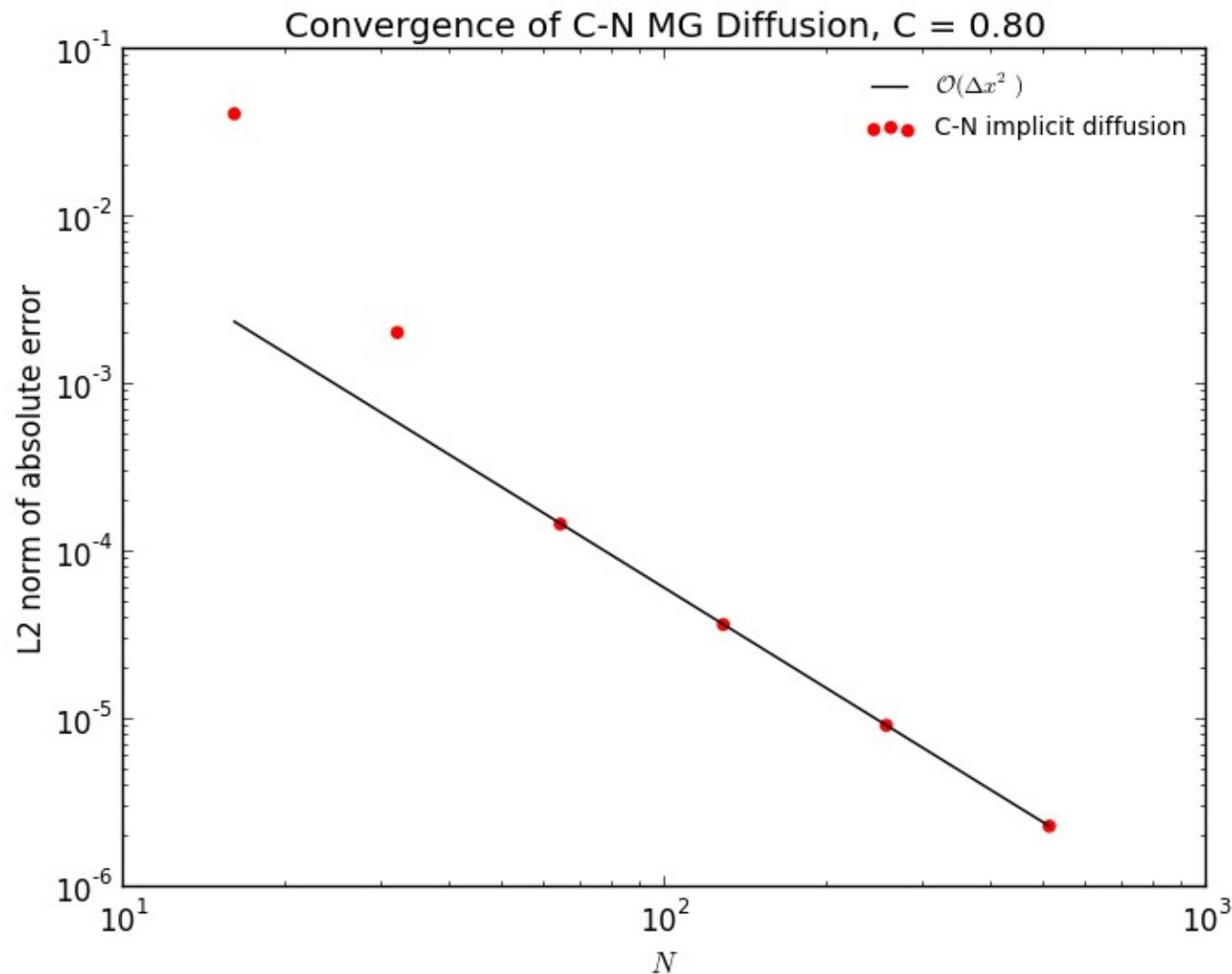
C-N MG Solution



Note: since the discretization is the same as the C-N direct solve, they should give the same answers (to tolerance)

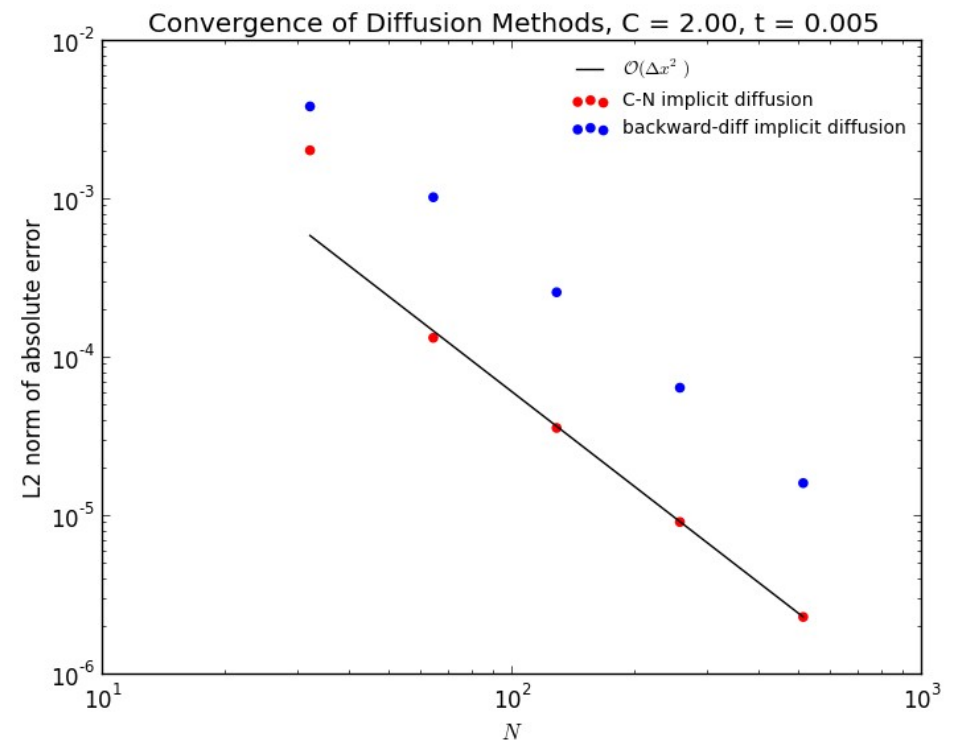
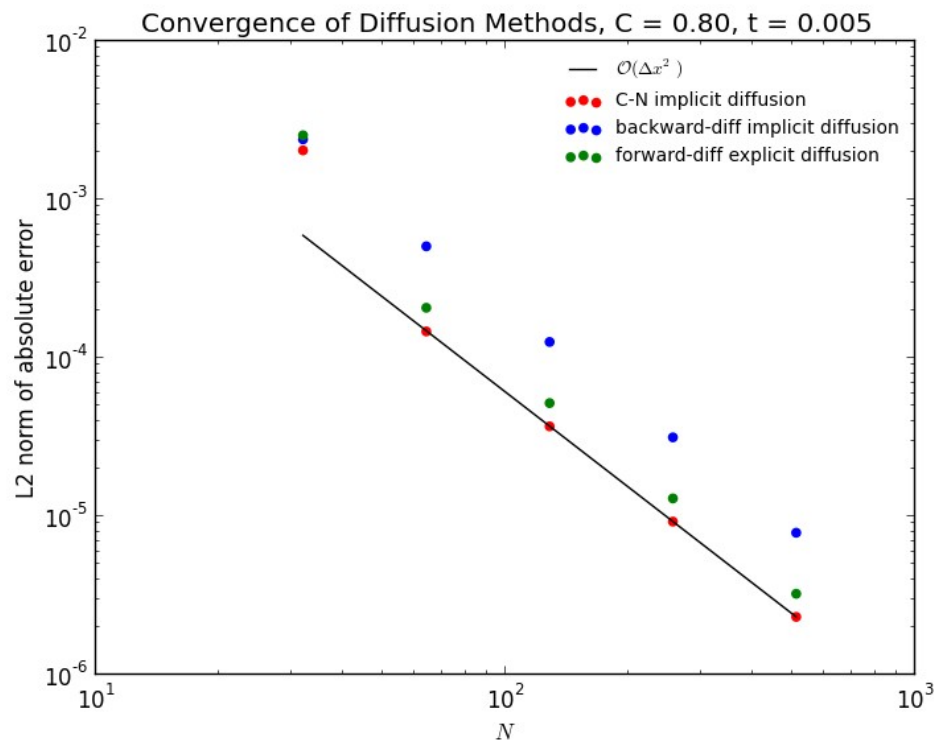
Convergence

- Again, once we are resolved, we converge as 2nd order



Comparison of Accuracy

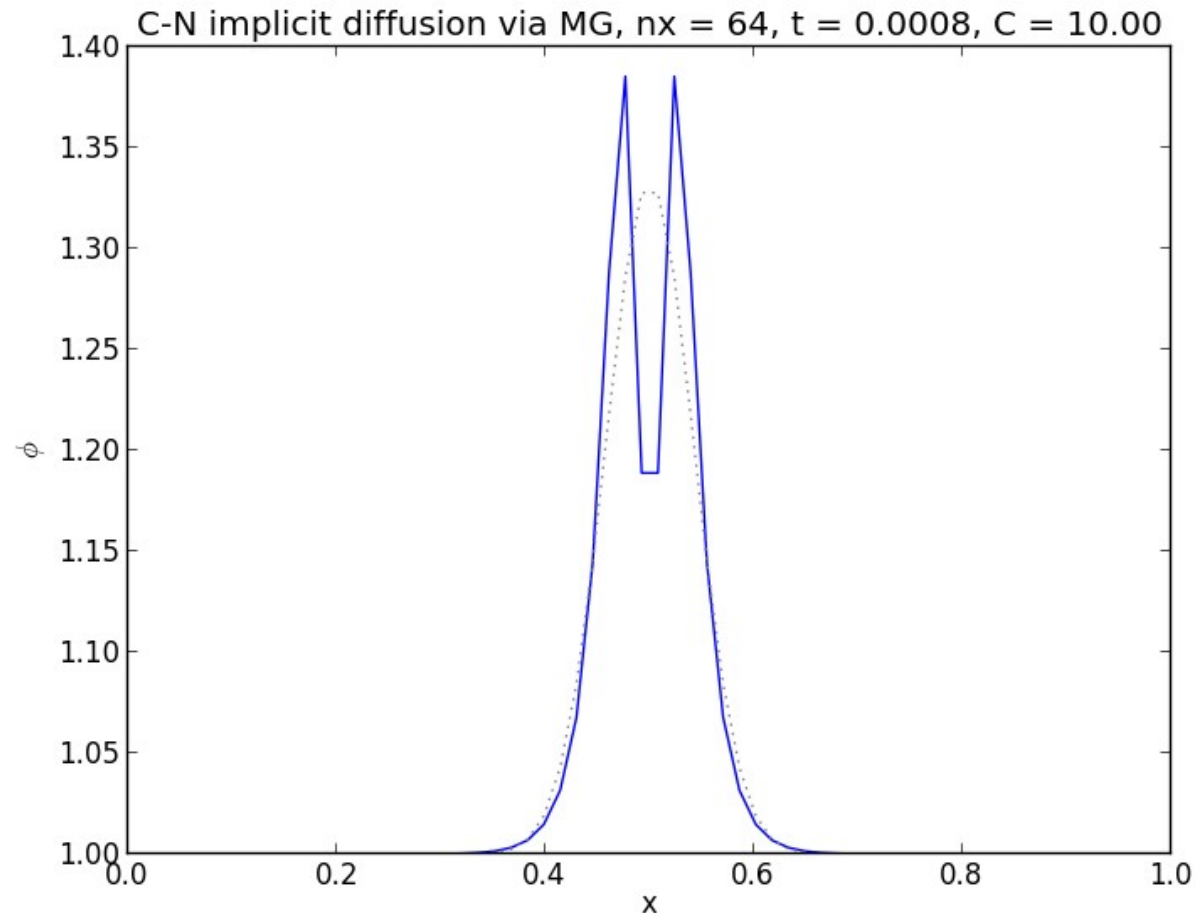
- Compare to our explicit (first-order in time) method to the direct backward-difference and C-N implicit methods:



Difference in accuracy between the backward-difference implicit and C-N implicit method grows with CFL

Important Observation

- If you make the CFL really large, then for underresolved resolved data, C-N can become... wierd
 - This is a known limitation of the type of stability it satisfies (beyond the scope of this course...)



Homework #3 Comments

- Problem 2 (Hilbert matrix and condition number):
 - If you do this in double precision, you will see that you get $O(1)$ error at $N = 12$
 - In single precision, it happens at $N = 7$
 - In quad precision, it happens at $N \sim 19$
- Problem 3 (Time-series analysis)
 - A big issue seems to be restricting the angle to fall between $[-\pi, \pi]$
 - This works regardless of the angle:

$$\theta' = \theta + \pi$$

$$\theta'' = \theta' - 2\pi \cdot \text{floor}(\theta'/2\pi)$$

$$\theta = \theta'' - \pi$$

Python Comment

- Several of you use `np.append(array, x)` to add element `x` to the array `array`
 - This creates allocates a new array with 1 more element, copies the original data over, and adds the new element at the end—very slow
 - Instead, just use a python list, and once you have all the elements, convert it to an array via: `array = np.array(list)`

Extensions

- So far, we've considered the simplest case of a constant coefficient in the diffusion equation
 - Nature doesn't always play this way
- There are two types of non-constant coefficients we might worry about:
 - $k = k(x)$
 - This can apply when k is a function of some other grid variable (i.e. not what we are diffusing)
 - $k = k(\phi)$ —this makes our equation nonlinear

Extensions

- Non-constant conductivity

- Discretize as:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{\{k \nabla \phi\}_{i+1/2} - \{k \nabla \phi\}_{i-1/2}}{\Delta x}$$

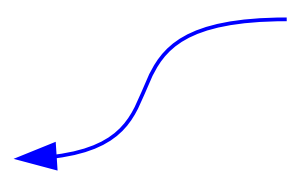
- Now we need k at the interfaces

- Averaging (several choices):

$$k_{i+1/2} = \frac{1}{2}(k_i + k_{i+1})$$

$$\frac{1}{k_{i+1/2}} = \frac{1}{2} \left(\frac{1}{k_i} + \frac{1}{k_{i+1}} \right)$$

This may be the
right thing for
conductivities



Extensions

- State-dependent transport coefficients

- We want:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2} \left\{ \nabla \cdot [k(\phi^n) \nabla \phi^n]_i + \nabla \cdot [k(\phi^{n+1}) \nabla \phi^{n+1}]_i \right\}$$

- Predictor-corrector:

$$\frac{\phi_i^* - \phi_i^n}{\Delta t} = \frac{1}{2} \left\{ \nabla \cdot [k(\phi^n) \nabla \phi^n]_i + \nabla \cdot [k(\phi^n) \nabla \phi^*]_i \right\}$$

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2} \left\{ \nabla \cdot [k(\phi^n) \nabla \phi^n]_i + \nabla \cdot [k(\phi^*) \nabla \phi^{n+1}]_i \right\}$$

- It may be necessary to iterate
 - With a good initial guess, this is second-order in time

Multi-physics Solvers

- In the real world, we often want to solve systems that span the different PDE classifications

Operator Splitting

- Consider the following system with advection, diffusion, and a source (e.g. Reactions):

$$\phi_t + [\phi u]_x = \epsilon \phi_{xx} + R(\phi)$$

- Consider each process separately
 - Advection: $\dot{\phi}^{(1)} = A(\phi^n)$
 - Diffusion: $\dot{\phi}^{(2)} = D(\phi^{(1)})$
 - Reactions: $\dot{\phi}^{n+1} = R(\phi^{(2)})$
- Each process operates on the state left behind by the previous operation
- Coupling between processes is weak—you may push off of the true solution
 - Here, reactions “saw” the advection and diffusion, but advection and diffusion did not see the reactions
- 2nd order-in-time requires some symmetry

Strang Splitting

- **Strang (1968)**: achieves second order accuracy when doing operator splitting

- Consider two-dimensional problem:

$$\phi_t = A\phi_x + B\phi_y$$

- Define the operators:

- Update in x through $\Delta t : L_{\Delta t}^x$
- Update in y through $\Delta t : L_{\Delta t}^y$

- A second-order update in time is then:

$$\phi^{n+1} = L_{\Delta t/2}^x L_{\Delta t}^y L_{\Delta t/2}^x \phi^n$$

- Notice that each operation sees the effects of the previous operation

Method of Lines

- Discretize everything in space
- ODEs in time—solve via our standard ODE methods
- Pro:
 - All processes are coupled to one another
- Con:
 - Must use the most restrictive timestep for all processes (imagine a stiff source, diffusion, and advection...)

Diffusion-Reaction

- With a reaction source term, we can have a propagating front:

$$\phi_t = \kappa \phi_{xx} + \frac{1}{\tau} R(\phi)$$

- This can be thought of as a simple model for a combustion flame
- **Imagine that the reactions are stiff**
 - We need to do this implicitly
 - Will may want a different timestep than the other physical processes
 - We likely want a high-order ODE method

Diffusion-Reaction

- Operator splitting:

- Integrate reaction part with high-order ODE solver with error estimation.
 - It may take many small timesteps to make up our desired Δt (subcycling)
- Use Strang splitting to make the time-integration overall second-order accurate:

$$\phi^{n+1} = R_{\Delta t/2} D_{\Delta t} R_{\Delta t/2} \phi^n$$

- Reactions are implicitly seen by diffusion because you've already integrated the reaction system to the midpoint in time
- No explicitly source terms describing one process appear in the other process' update

Diffusion-Reaction

- Solution procedure

- Evolve reaction system for $\Delta t/2$

$$\frac{d\phi^*}{dt} = \frac{1}{\tau} R(\phi^*), \quad \phi^*(0) = \phi^n$$

- Solve the diffusion equation for Δt

$$\frac{\phi^{**} - \phi^*}{\Delta t} = \frac{1}{2} (D(\phi^*) + D(\phi^{**}))$$

- Evolve reaction system for $\Delta t/2$

$$\frac{d\phi^{n+1}}{dt} = \frac{1}{\tau} R(\phi^{n+1}), \quad \phi^{n+1}(0) = \phi^{**}$$

Diffusion-Reaction

- Consider the simple reaction source:

$$R(\phi) = \frac{1}{4}\phi(1 - \phi)$$

- This is the KPP reaction source (see e.g, Vladimirova 2006)
- Consider ϕ to be a progress variable that lies between 0 (pure reactant) and 1 (pure fuel)
 - Solution is a traveling wave with speed and thickness:

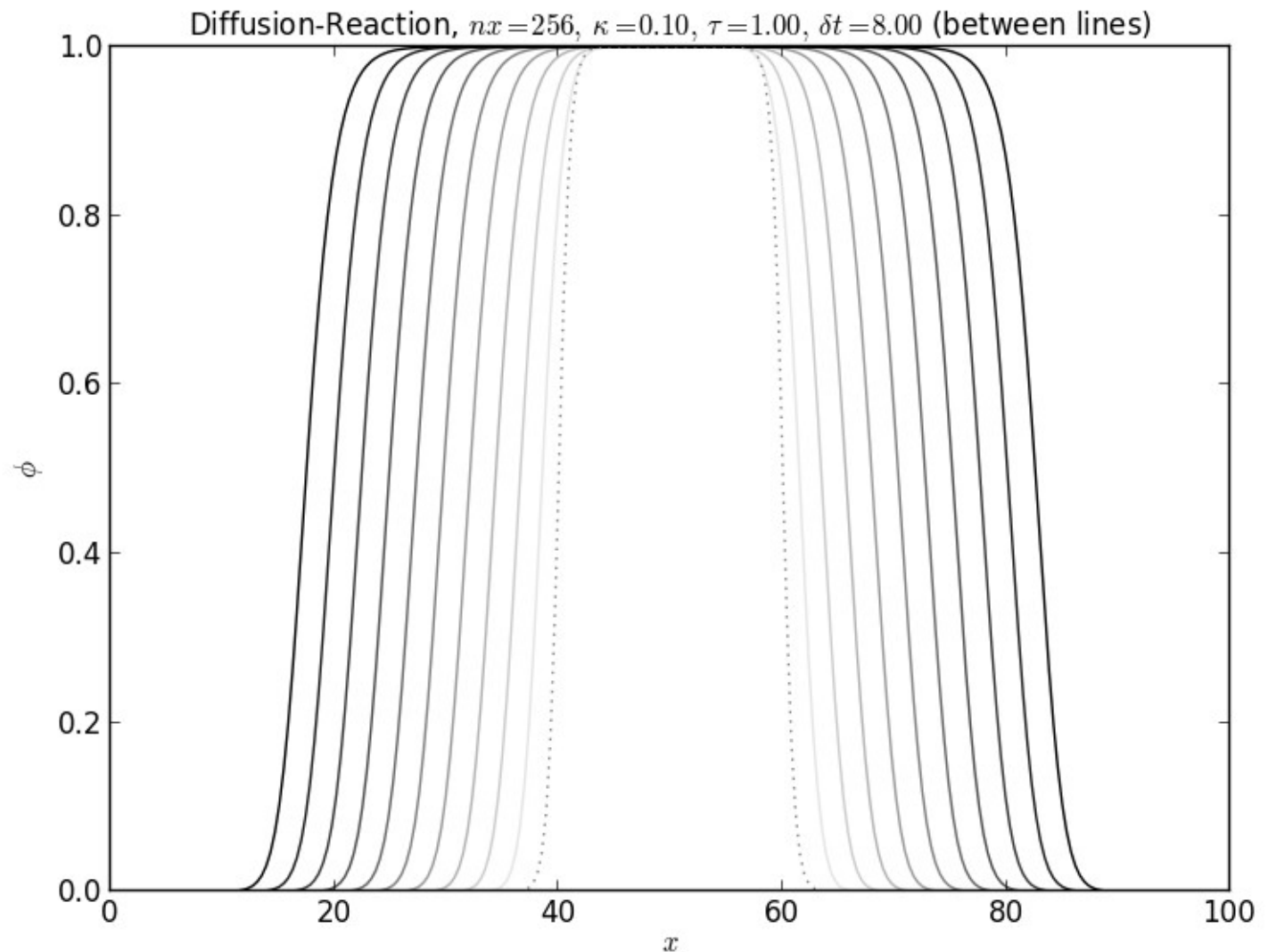
$$S = \sqrt{\kappa/\tau} \quad \delta = \sqrt{\kappa\tau}$$

Diffusion-Reaction

- Starting with a smoothed tophat (tanh smoothing).
 - If the initial conditions are too sharp, then the diffusion acts funny initially

Let's play with the code and slow down reactions...

...and check out what happens if our initial profile is a tophat



Viscous Burger's Equation

- Burgers equation with viscosity

$$u_t + uu_x = \epsilon u_{xx}$$

- The viscosity can act to smear out a shock a little—provides a more physical description
- We will treat the advection and diffusion parts separately
 - We've seen explicit methods work well for advection
 - Diffusion has a very restrictive timestep constraint, so we want to use an implicit method for it

Viscous Burger's Equation

- Let's write our Burger's equation as:

$$u_t + A(u) = D(u)$$

- Where $A(u)$ is a discrete approximation to the advective term:

$$A(u) = \left[\frac{1}{2} u^2 \right]_x$$

- $D(u)$ is a discrete approximation to the diffusion term:

$$D(u) = \epsilon u_{xx}$$

Viscous Burger's Equation

- I. Find the advective update over the timestep:
 - Construct the second-order interface states with $D(u^n)$ as a source term
 - Solve the Riemann problem for Burger's equation
 - Construct the advective update:

$$A_i^{n+1/2} = \frac{\left[\frac{1}{2} \left(u_{i+1/2}^{n+1/2} \right)^2 \right] - \left[\frac{1}{2} \left(u_{i-1/2}^{n+1/2} \right)^2 \right]}{\Delta x}$$

Viscous Burger's Equation

- II. Solve the diffusion equation with the advective source
 - Use a Crank-Nicolson discretization of the diffusion part of our equation:

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}D(u^n) + \frac{1}{2}D(u^{n+1}) - A^{n+1/2}$$

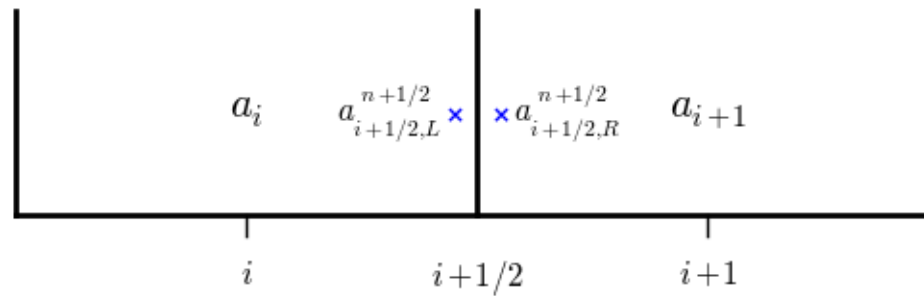
- This is a linear system that can be solved as a tridiagonal matrix or with multigrid

$$u^{n+1} - \frac{\Delta t}{2}D(u^{n+1}) = u^n + \frac{\Delta t}{2}D(u^n) - \Delta t A^{n+1/2}$$

- The resulting u^{n+1} is now updated with the advection (treated explicitly) and the diffusion (treated implicitly)
 - The only timestep constraint (for stability) comes from the advective portion

Viscous Burger's Advection

- We didn't consider source terms in our advection lecture
- Recall that we use a finite-volume grid:



- Construct left/ interface states by Taylor expanding in space and time

$$\begin{aligned} u_{i+1/2,L}^{n+1/2} &= u_i^n + \frac{\Delta x}{2} \frac{\partial u}{\partial x} + \frac{\Delta t}{2} \frac{\partial u}{\partial t} + \dots \\ &= u_i^n + \frac{\Delta x}{2} \frac{\partial u}{\partial x} + \frac{\Delta t}{2} \left(-u \frac{\partial u}{\partial x} + D(u_i^n) \right) \\ &= u_i^n + \frac{1}{2} \left(1 - \frac{\Delta t}{\Delta x} u_i \right) \overline{\Delta u}_i + \frac{\Delta t}{2} D(u^n) \end{aligned}$$

Viscous Burger's Advection

- The Riemann problem remains unchanged

- If $u_l > u_r$ then we are compressing (shock):

$$S = \frac{1}{2}(u_l + u_r)$$

$$u_{i+1/2} = \begin{cases} u_l & S > 0 \\ u_r & S < 0 \end{cases}$$

- Otherwise rarefaction:

$$u_{i+1/2} = \begin{cases} u_l & u_l > 0 \\ u_r & u_r < 0 \\ 0 & \text{otherwise} \end{cases}$$

- The characteristic structure is from the hyperbolic portion only
- The diffusion enters into the Riemann problem only through the addition to the interface states

Viscous Burger's Diffusion

- Updating the system now becomes solving the Parabolic diffusion equation

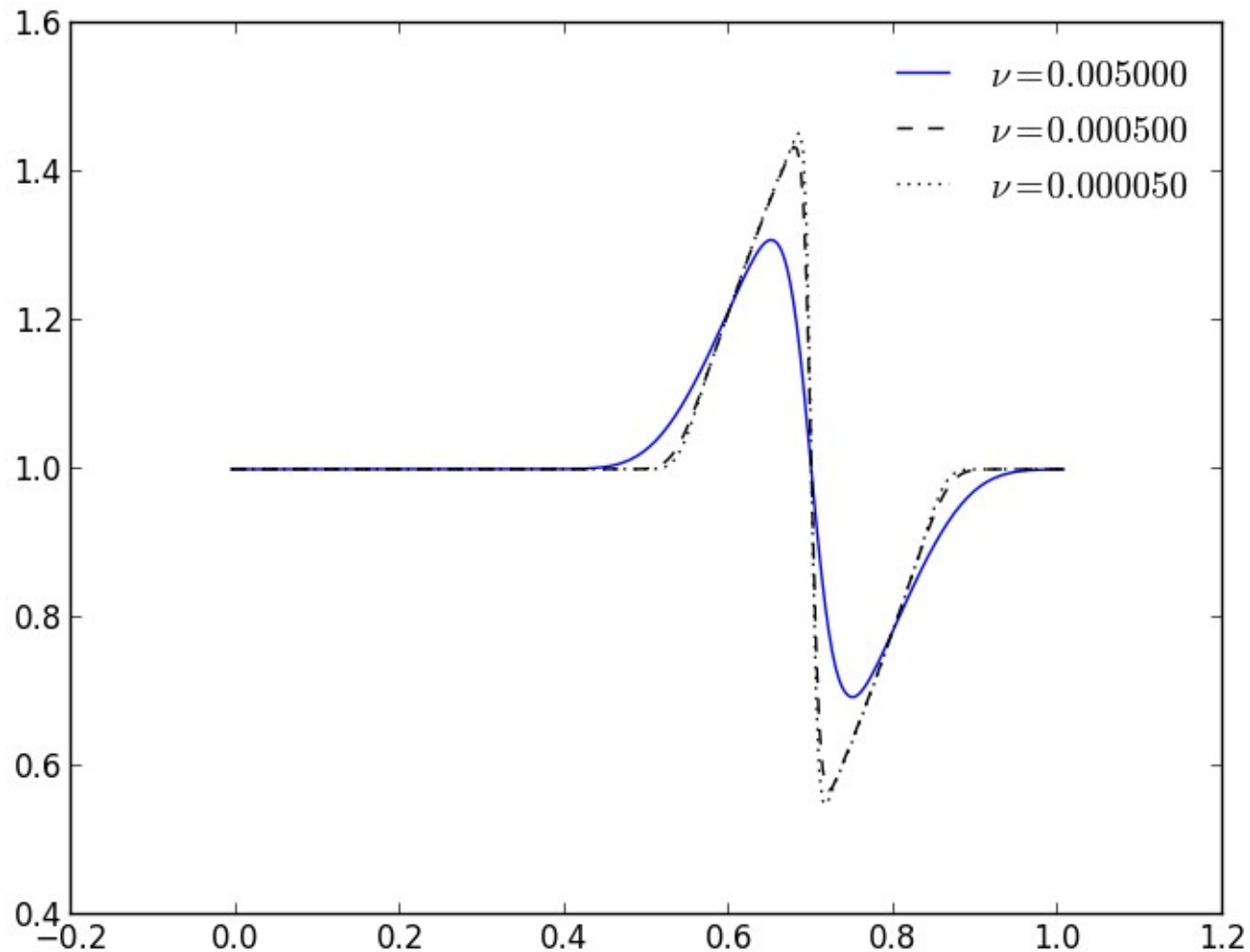
- The change due to advection just appears as a source:

$$u^{n+1} - \frac{\Delta t}{2} D(u^{n+1}) = u^n + \frac{\Delta t}{2} D(u^n) - \Delta t A^{n+1/2}$$

- The only change to our previous diffusion term is the addition of the source to the RHS of the linear system

Viscous Burger's Equation

- Viscosity acts to smear out the shock profile



(let's look at the code...)

Other Advection-Diffusion Systems

- A similar type of method can be used for the Navier-Stokes equations
 - For compressible flow, the viscous flux is a tensor, so in multi-dimensions, the viscous solve is a set of coupled parabolic equations.

Incompressible Flow

- Consider the equations of incompressible flow:

$$U_t + U \cdot \nabla U + \nabla p = 0$$

$$\nabla \cdot U = 0$$

- The momentum equation is hyperbolic (looks like Burger's equation)
- The velocity constraint is elliptic
- Pressure here is not physical—it takes the value that is needed to satisfy the constraint

Incompressible Flow

- A popular method for this is the **projection method**:
 - First update the velocity using our 2nd order finite-volume stuff, ignoring the constraint:

$$U^* = U^n - \Delta t A^{n+1/2} - \Delta t \nabla p^{n-1/2}$$

- Now recall that any vector field can be written as the sum of a divergence free term + gradient of a scalar (see your homework!):

$$U^* = U^d + \nabla \phi$$

- This gives the elliptic equation:

$$\nabla^2 \phi = \nabla \cdot U^*$$

- Solve for the scalar and correct the velocity field:

$$U^{n+1} = U^* - \nabla \phi \quad p^{n+1/2} = \phi$$

- This is the basis for low Mach number methods is combustion, astrophysics, and atmospheric flow

Other Coupling Methods

- Sometimes operator splitting breaks down
 - Reduce the timestep—if the solution behaves better at smaller Δt , then the coupling between the different processes may not be strong enough
- One fix is to use a method-of-lines integration strategy
 - This means that you must step everything at the most restrictive timestep
- Alternate approach involves iteration to ensure coupling

Other Coupling Methods

- Spectral Deferred Corrections

- Consider advection and reactions:

$$\dot{\phi} + A(\phi) = R(\phi)$$

- Iteration loop:

- Advect $\dot{\phi} + A(\phi) = R^{(0)}$ where $R^{(0)}$ is an estimate of the reactions over a timestep. This gives the state ϕ^*

- React:

$$\dot{\phi} = R(\phi) - A^*$$

with the advection source taken as constant in time. This gives $\phi^{(1)}$

- This methodology has been shown to improve the coupling of physical processes in flames.

Some Multi-d Fun...

- We mostly focused on one-dimensional problems, but all of these methods can be extended to multi-dimensions

Multi-Dimensional Advection

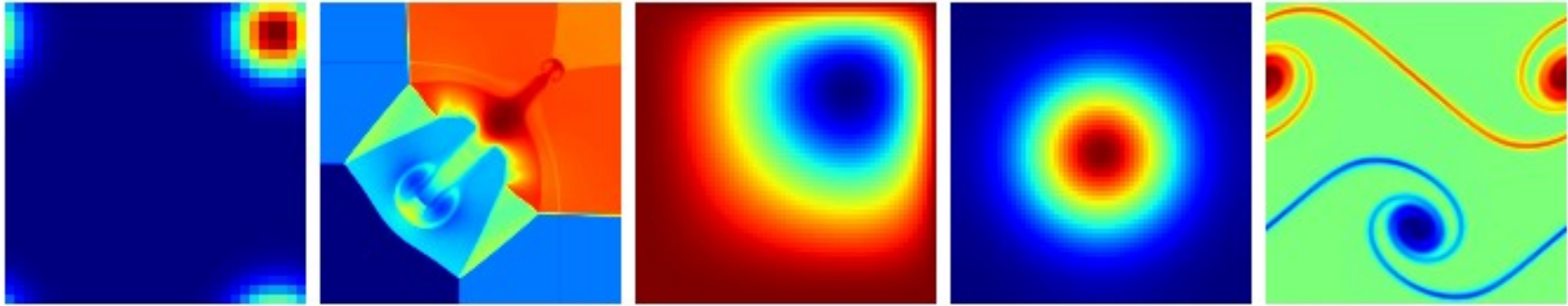
- Simplest method: **dimensional splitting**
 - Perform the update one dimension at a time—this allows you to reuse your existing 1-d solver

$$\frac{a_{i,j}^* - a_{i,j}^n}{\Delta t} = - \frac{u a_{i+1/2,j}^{n+1/2} - u a_{i-1/2,j}^{n+1/2}}{\Delta x}$$

$$\frac{a_{i,j}^{n+1} - a_{i,j}^*}{\Delta t} = - \frac{v a_{i,j+1/2}^{\star,n+1/2} - v a_{i,j-1/2}^{\star,n+1/2}}{\Delta y}$$

- More complicated: **unsplit prediction** of interface states
 - In Taylor expansion, we now keep the transverse terms at each interface
 - Difference of transverse flux terms is added to normal states
 - Better preserves symmetries
 - See Colella (1990) and PDF notes on course page for overview

Pyro: Hydro by Example



- “python hydro”
 - Python-based teaching code
 - Implements 2-d advection, multigrid, implicit diffusion, compressible and incompressible hydrodynamics
 - Written for clarity
 - We'll use this for our fun with compressible hydro later
 - Linked to from our class page (distributed via git)