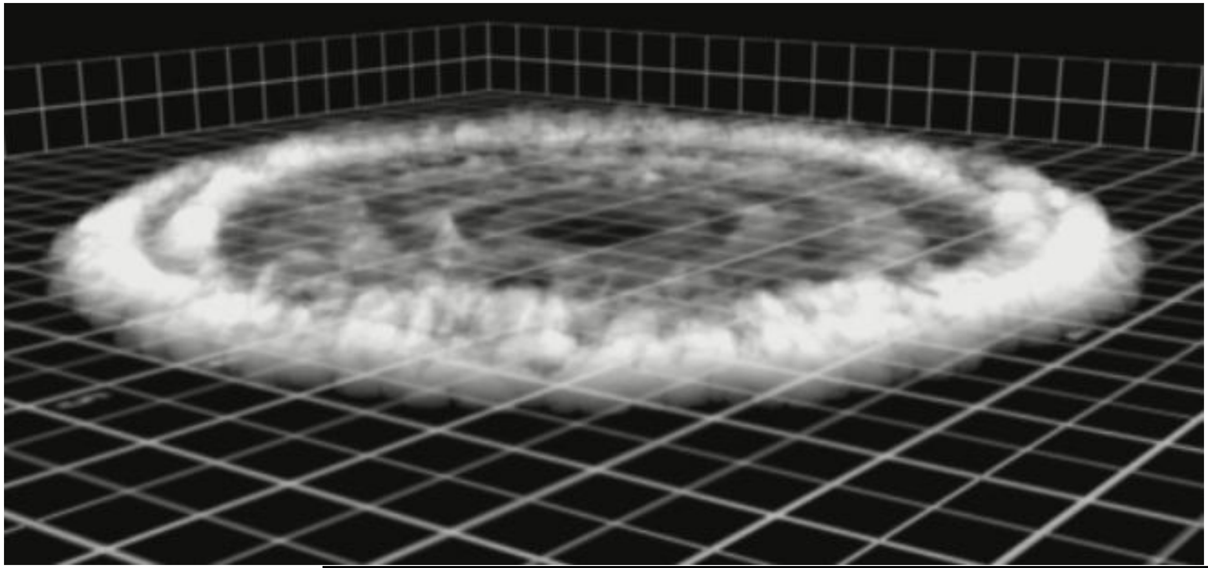
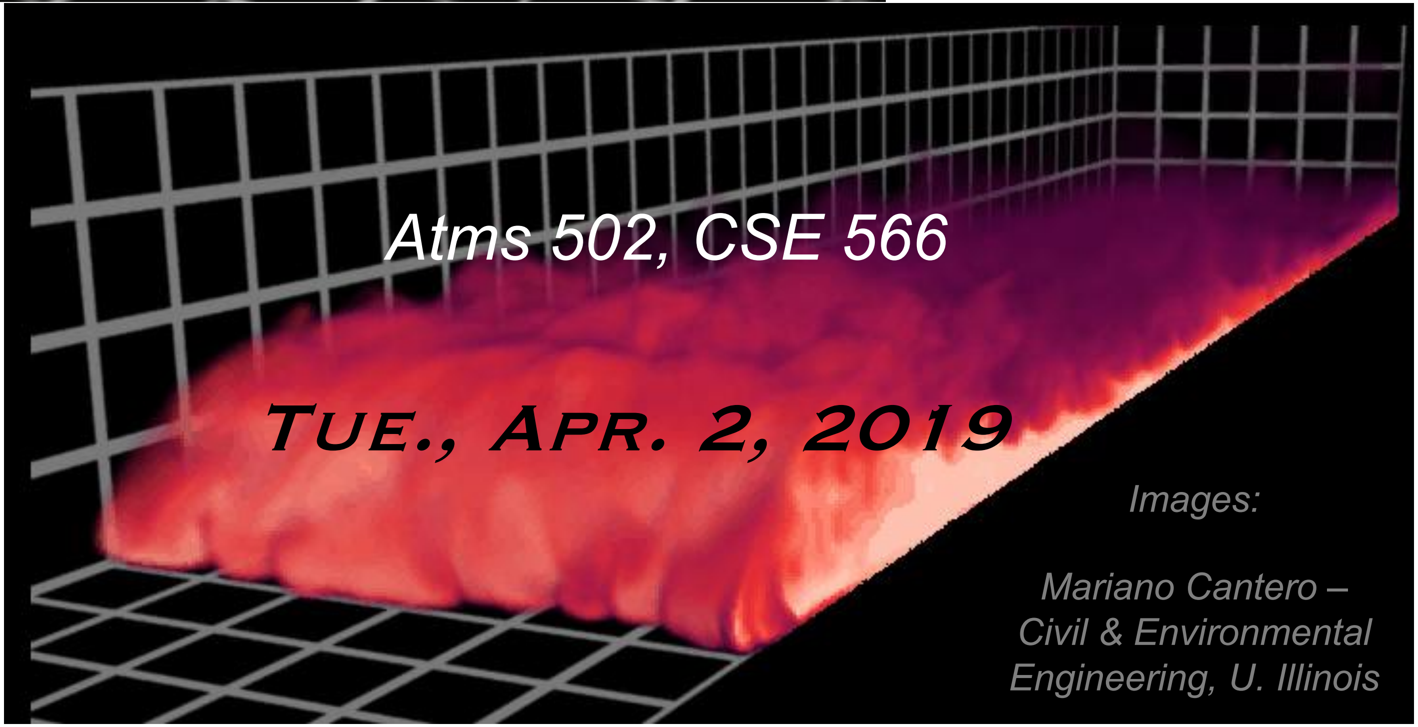


Numerical Fluid Dynamics



Atms 502, CSE 566

TUE., APR. 2, 2019

Images:

Mariano Cantero – Civil & Environmental Engineering, U. Illinois

Plan for Today

ATMS 502
CSE 566

Tuesday,
2 April 2019

Class #21

Program #5 is due Tuesday, April 16

- 1) Review
  - Nonlinear instability and aliasing
  - Quasi-compressible system
- 2) Semi-Lagrangian methods
  - Advantages, questions, choices
- 3) Program 5
  - 2-D quasi-compressible nonlinear flow

Semi-Lagrangian methods

Semi-Lagrangian Methods

Ritchie et al. 1995

- "The main motivation for using a semi-Lagrangian formulation is to permit the use of time steps that far exceed the CFL stability criterion for the corresponding Eulerian model … provided that the additional time truncation error does not significantly decrease the accuracy"
- Their case: 4x improvement in efficiency

Semi-Lagrangian Methods

- Generally:
  - Eulerian view - evolution at a point
  - Lagrangian view - following fluid motion
  - *Semi-Lagrangian* viewpoint …
- Semi-Lagrangian methods: find source of tracer arriving at fixed grid locations

$$\boxed{\frac{dF}{dt} = \frac{\partial F}{\partial t} + \frac{dx}{dt}\frac{\partial F}{\partial x} = 0 \qquad \frac{dx}{dt} = U(x,t)}$$

Semi-Lagrangian Properties-1

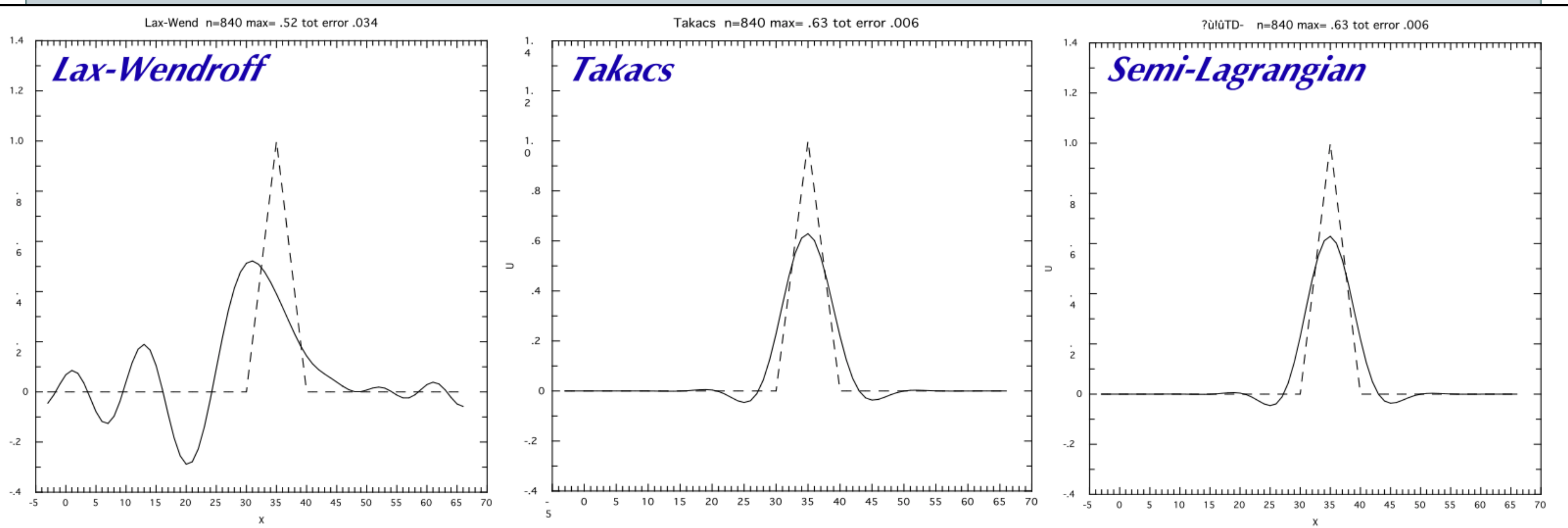
- Maximum Δt not limited by maximum wind speed
- *Can stably integrate with Courant numbers > 1*
- Can handle sharp gradients / discontinuities well
- But … does not have conservation properties like finite volume methods
- Can be somewhat more expensive per time step

Semi-Lagrangian Properties-2

- Low dispersion
- Generally accurate but there is damping due to interpolation, though it is scale-selective.
- Important to limit truncation errors in
  1. discretized governing equations and
  2. in trajectory computations

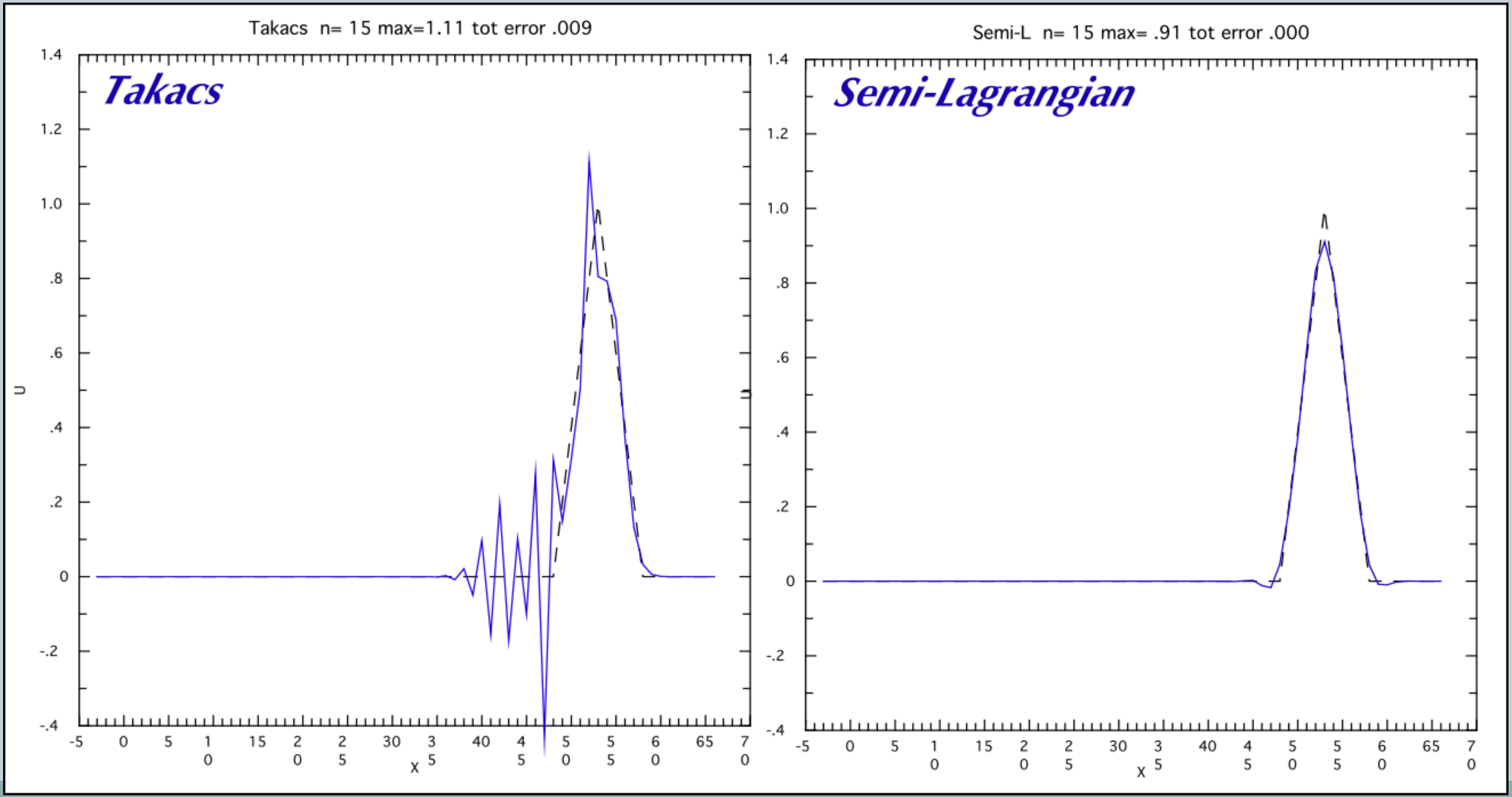
Semi-Lagrangian Methods

- Linear advection example
- 3 revolutions, courant number 0.25



Semi-Lagrangian Methods

- Different case… with Courant # 1.2



Semi-Lagrangian Methods

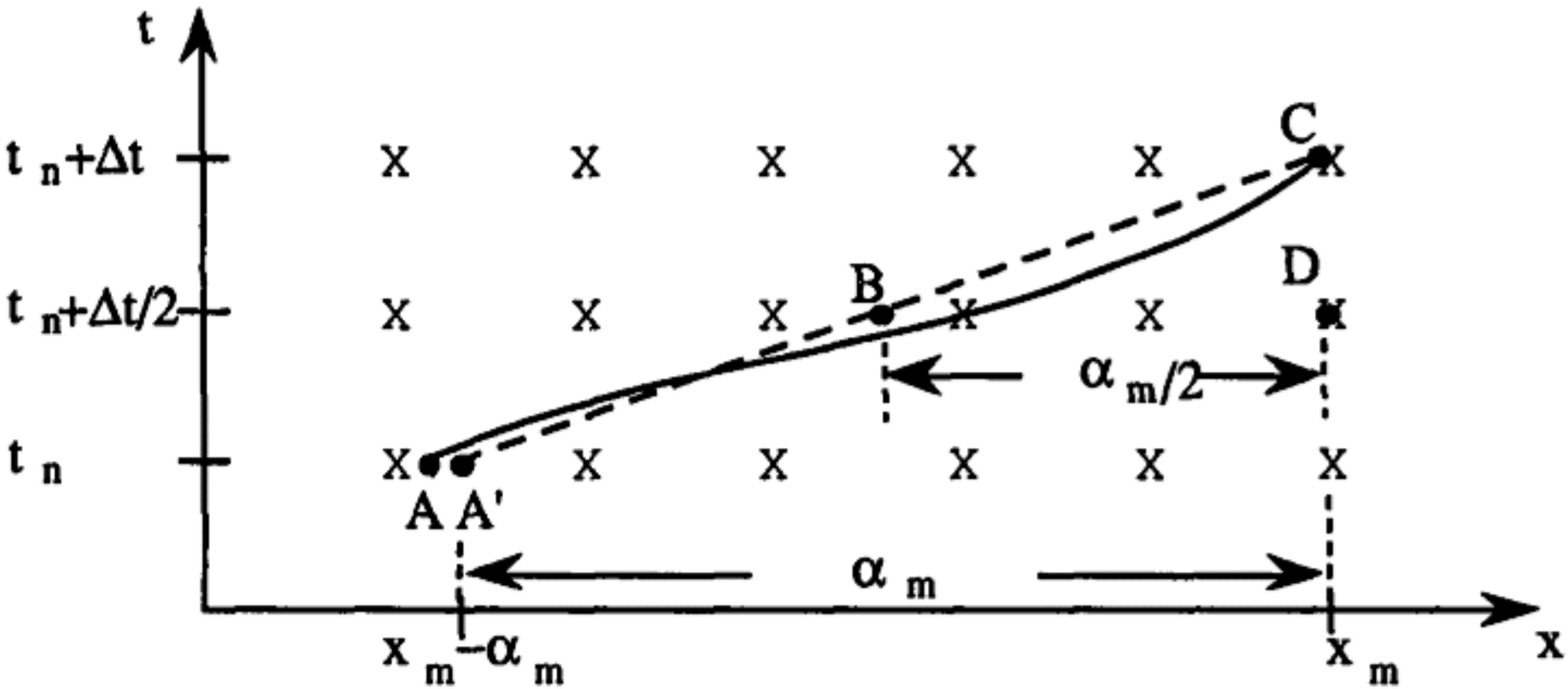
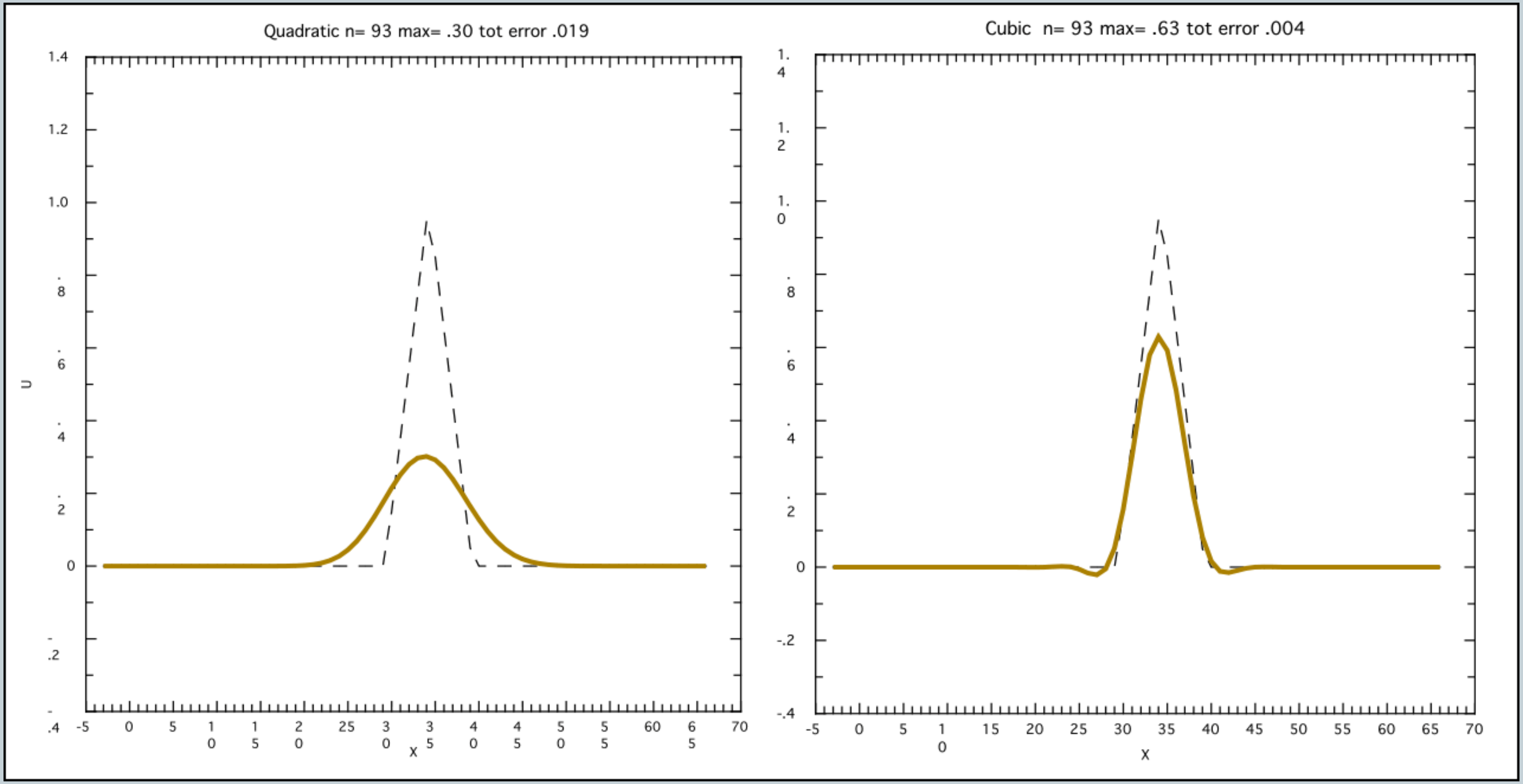


FIG. 2. Schematic for two-time-level advection. Actual (solid curve) and approximated (dashed line) trajectories that arrive at mesh point x_m at time $t_n + \Delta t$. Here α_m is the distance the particle is displaced in x in time Δt.

STANIFORTH AND COTE

Semi-Lagrangian Methods

- Interpolation matters.

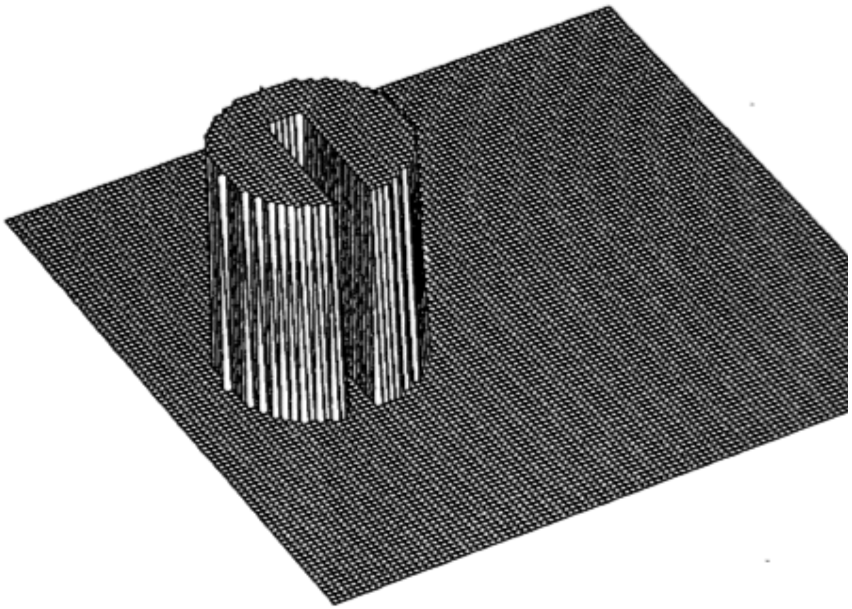


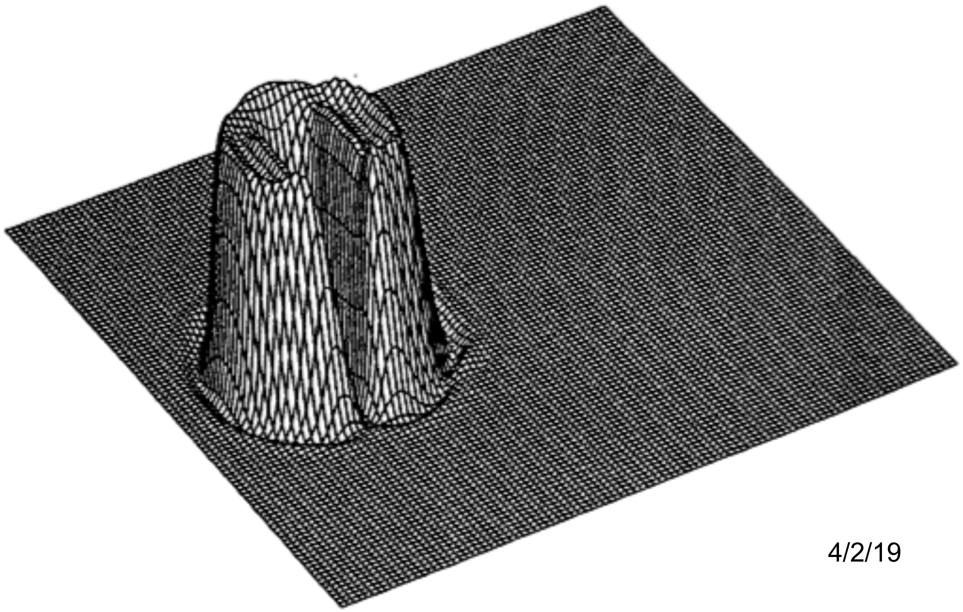
Semi-Lagrangian Methods

- Staniforth and Cote, 1991, Mon. Wea. Rev.

“Slotted” cylinder

6 revolutions





Program #5

2D NONLINEAR, COMPRESSIBLE FLOW

Program 5: Overview

- We are modeling nonlinear compressible flow.
  - nonlinear: time-evolving flow fields
  - compressible: well, *quasi*-compressible.
    - there are sound waves
    - we will *set* the sound wave speed C_s (300 m s^{-1} = full speed)
    - the only explicit density in our equations is a function of z, only.

Program 5: Overview

- We are modeling nonlinear compressible flow.
  - nonlinear: time-evolving flow fields
  - compressible: well, *quasi*-compressible.
    - there are sound waves
    - we will *set* the sound wave speed C_s (300 m s^{-1} = full speed)
    - the only explicit density in our equations is a function of z, only.
- Everything starts with temperature (θ)
  - we specify up to *two* temperature perturbations.
  - *perturbation* (potential) temperature = $\theta - \bar{\theta}$
    - with $\bar{\theta} = 300$ (a constant). When plotting θ, always plot $\theta - \bar{\theta}$.
  - Change θ > P changes > U and W respond.
  - Initial conditions: Specify initial θ; initial U,W,P' = *zero*.

Program 5: Structure, BCs, base state

- General program structure
  - input
  - time loop
    - start array update: *u3=u1*, etc.
    - call BC routine
    - call main physics subroutines
      - advection
      - diffusion
      - pgf
    - array update
  - *on to the next step*

- Boundary conditions
  - Symmetric
  - Periodic
  - Zero-gradient
  - "Slip"

- "Base state"
  - Only base state variable you *use*: density (1-D)
  - Pressure – perturbation?
  - θ/temperature – perturbation?

Program 5: Organization

- We'll group functions by physical process
  - Each physical process: One subroutine
  - (1): ***advection*** routine will
    - handle θ advection via calls to your *advect1d* routine, as before
    - *advection* will also handle advection of U, W.
  - (2): ***diffusion*** routine will
    - carry out all diffusive terms, involving θ, U, W
  - (3): ***PGF*** routine handles all terms involving pressure
    - PGF = *pressure gradient force*
    - This routine will do the final contributions to U, W
    - These new (u3, w3) terms are used to find the new P', p3.

Program 5: 2D continuous equations

- Full program 5 equations
  - all v terms and y-derivatives are ignored.

u-momentum:	$u_t = -uu_x \boxed{\quad} - wu_z - \dfrac{1}{\bar{\rho}} p'_x + K\left(u_{xx} \boxed{\quad} + u_{zz}\right)$
v-momentum: (program 6 only)	$\boxed{\quad}$
w-momentum: $\left(\theta' = \theta - \bar{\theta}\right)$	$w_t = -uw_x \boxed{\quad} - ww_z - \dfrac{1}{\bar{\rho}} p'_z + g\dfrac{\theta'}{\bar{\theta}} + K\left(w_{xx} \boxed{\quad} + w_{zz}\right)$
Perturbation pressure:	$p'_t = -c_s^2\left(\bar{\rho}\dfrac{\partial u}{\partial x} \boxed{\quad} + \dfrac{\partial}{\partial z}\left(\bar{\rho} w\right)\right)$
θ *(pot. temperature):*	$\theta_t = -\left(u\theta\right)_x \boxed{\quad} - \left(w\theta\right)_z + \theta\left(u_x \boxed{\quad} + w_z\right) + K\left(\theta_{xx} \boxed{\quad} + \theta'_{zz}\right)$

Processes: Advection

- Full program 5 equations
  - Advection highlighted.

Tests "B" and "C" are only for θ advection.

u-momentum:	$u_t = -uu_x - wu_z - \frac{1}{\bar{\rho}} p'_x + K\left(u_{xx} + u_{zz}\right)$
v-momentum: (program 6 only)	
w-momentum: $\left(\theta' = \theta - \bar{\theta}\right)$	$w_t = -uw_x - ww_z - \frac{1}{\bar{\rho}} p'_z + g\frac{\theta'}{\bar{\theta}} + K\left(w_{xx} + w_{zz}\right)$
Perturbation pressure:	$p'_t = -c_s^2\left(\bar{\rho}\frac{\partial u}{\partial x} + \frac{\partial}{\partial z}\left(\bar{\rho}w\right)\right)$
θ *(pot. temperature):*	$\theta_t = -\left(u\theta\right)_x - \left(w\theta\right)_z + \theta\left(u_x + w_z\right) + K\left(\theta_{xx} + \theta'_{zz}\right)$

"advection" routine now includes *u*, *w* (nonlinear)

Processes: Diffusion

- Full program 5 equations
  - Diffusion highlighted.

u-momentum:	$u_t = -uu_x - wu_z - \frac{1}{\bar{\rho}}p'_x + \boxed{K\left(u_{xx} + u_{zz}\right)}$
v-momentum: (program 6 only)	
w-momentum: $(\theta' = \theta - \bar{\theta})$	$w_t = -uw_x - ww_z - \frac{1}{\bar{\rho}}p'_z + g\frac{\theta'}{\bar{\theta}} + \boxed{K\left(w_{xx} + w_{zz}\right)}$
Perturbation pressure:	$p'_t = -c_s^2\left(\bar{\rho}\frac{\partial u}{\partial x} + \frac{\partial}{\partial z}(\bar{\rho}w)\right)$
θ *(pot. temperature):*	$\theta_t = -(u\theta)_x - (w\theta)_z + \theta\left(u_x + w_z\right) + \boxed{K\left(\theta_{xx} + \theta'_{zz}\right)}$

"diffusion" evaluates derivatives at *(n-1)* for u & w; *(n)* for θ

Processes: PGF+buoyancy

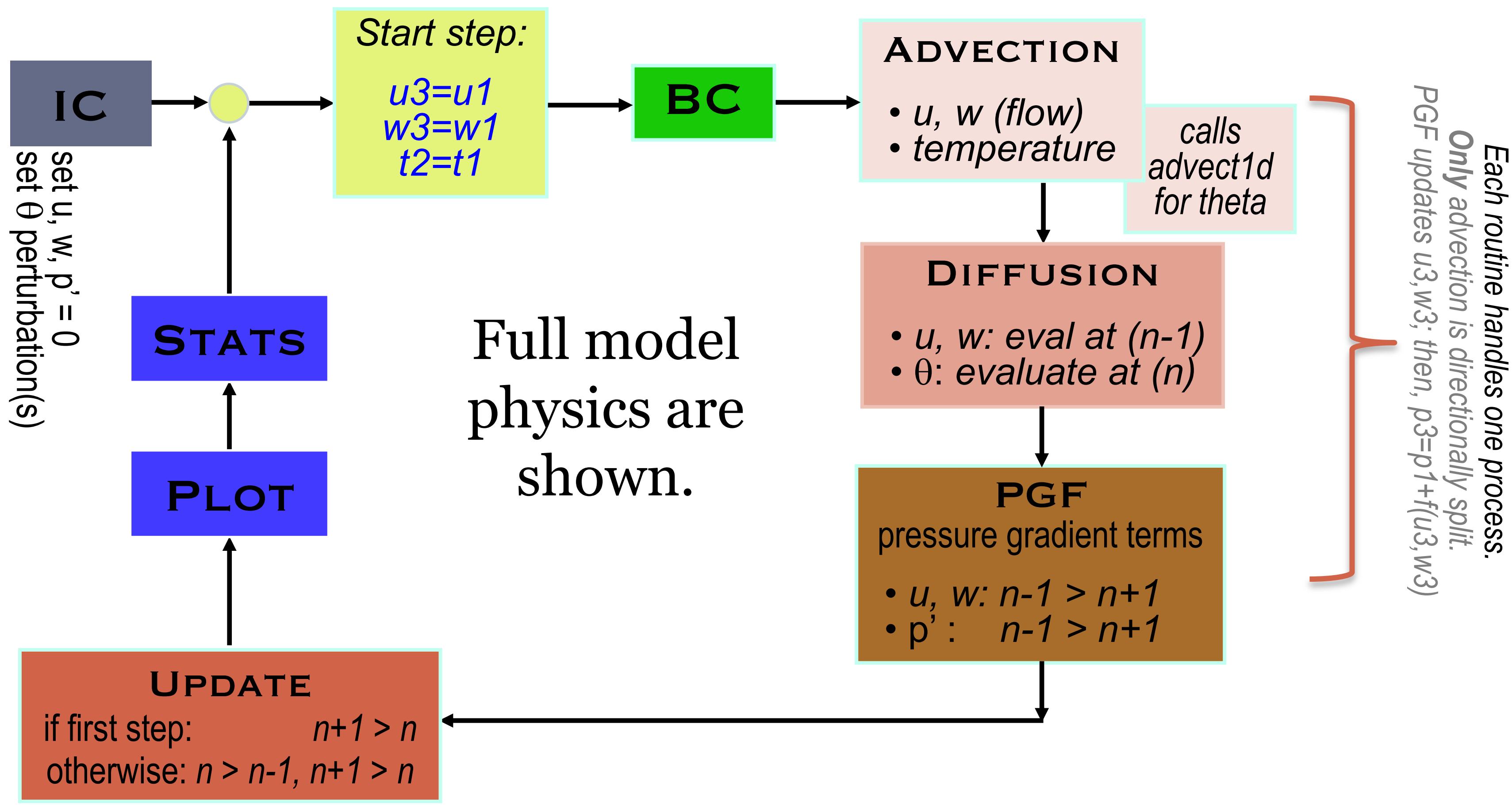
- Full program 5 equations
  - pressure gradient & buoyancy highlighted.

This is **test "A"**

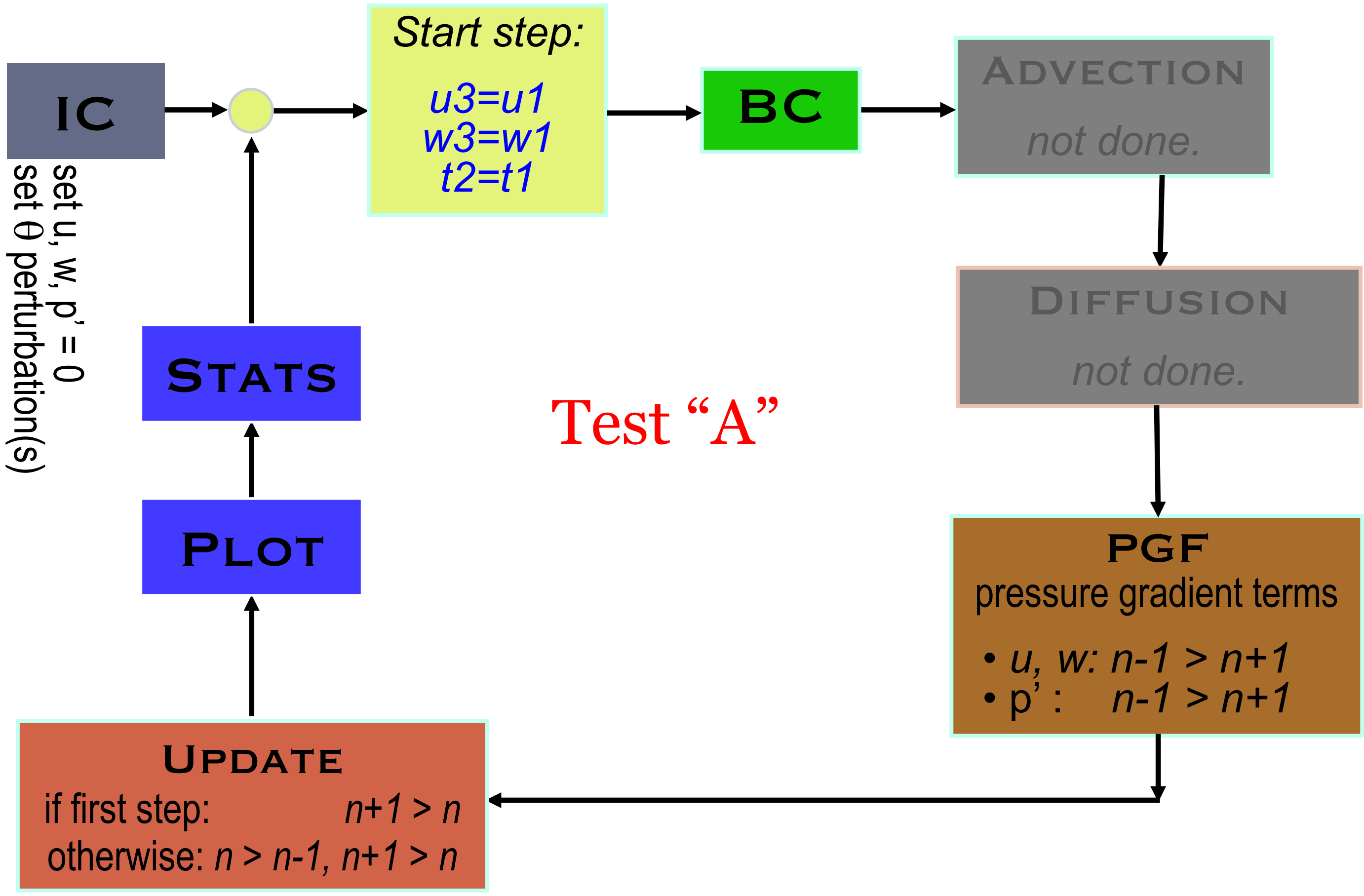
u-momentum:	$u_t = -uu_x \boxed{} - wu_z \boxed{-\frac{1}{\bar{\rho}}p'_x} + K\left(u_{xx} \boxed{} + u_{zz}\right)$
v-momentum: (program 6 only)	
w-momentum: $\left(\theta' = \theta - \bar{\theta}\right)$	$w_t = -uw_x \boxed{} - ww_z \boxed{-\frac{1}{\bar{\rho}}p'_z + g\frac{\theta'}{\bar{\theta}}} + K\left(w_{xx} \boxed{} + w_{zz}\right)$
Perturbation pressure:	$p'_t = \boxed{-c_s^2\left(\bar{\rho}\frac{\partial u}{\partial x} \boxed{} + \frac{\partial}{\partial z}\left(\bar{\rho}w\right)\right)}$
θ *(pot. temperature):*	$\theta_t = -\left(u\theta\right)_x \boxed{} - \left(w\theta\right)_z + \theta\left(u_x \boxed{} + w_z\right) + K\left(\theta_{xx} \boxed{} + \theta'_{zz}\right)$

u3 = u3 + …; w3 = w3 + …; *set u, w BCs*; p3 = p1 + …

Program 5: Structure



Program 5: test case "A"



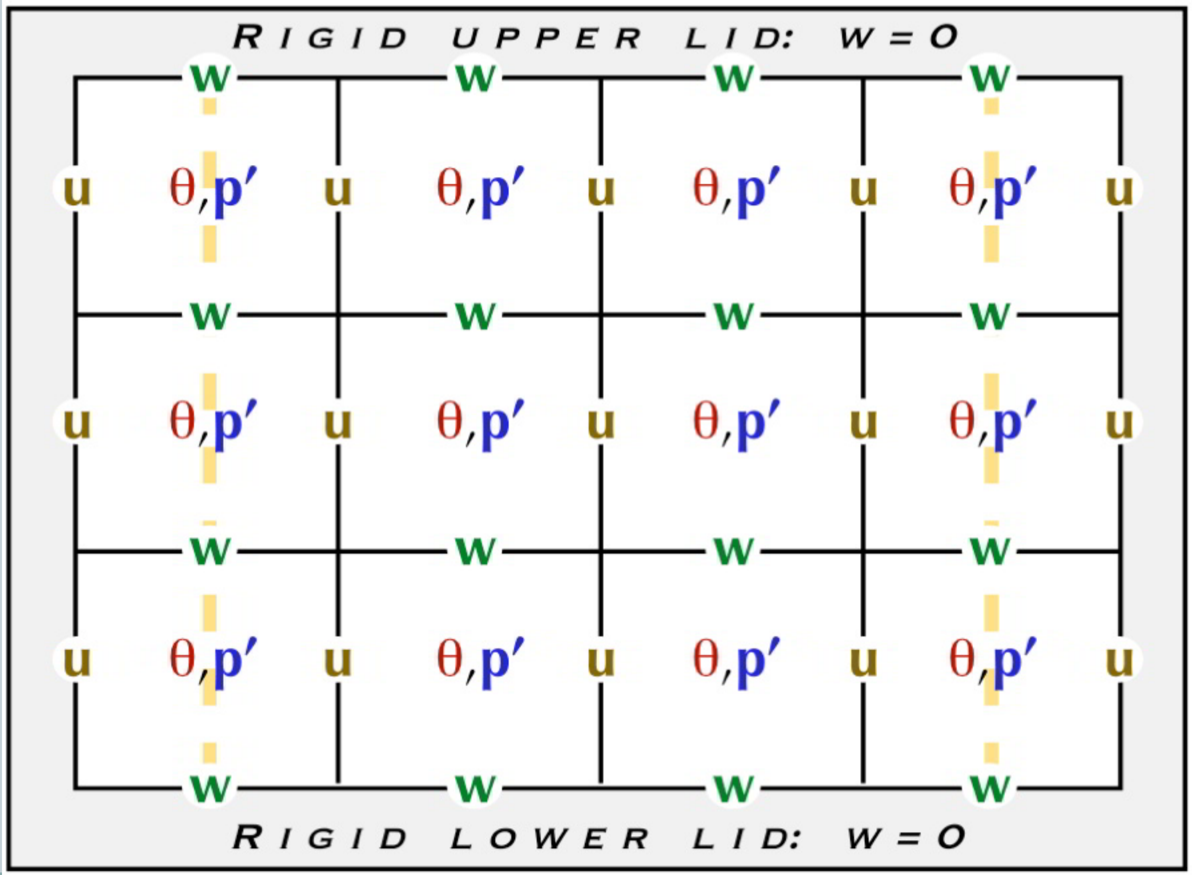
- in this test we only use processes in "PGF"
- other routines are *not called*
- θ does *not change* from the IC values.

Program 5: Dimensions

- Array dimensions
  - Theta (θ) is treated as before, except …
    - we have added a dissipation term
  - Your 2-D arrays : NX ≠ NZ !!!
  - You have additional 2-D arrays now that we are **nonlinear**:
    - arrays for U, W are now time-evolving and need ghost zones!
    - new array: P (for perturbation pressure. needs ghost zones too)
  - New 1-D arrays
    - for density at *theta/u/p levels* (altitudes)
    - for density at *w-levels* (in-between those for theta/u/p)
      - this density is **not** time-varying. Set it only once…
    - other arrays are used as part of initialization
      - and are never needed again.

Program 5: BCs

- Boundary conditions (B.C.'s)
  - Z: B.C.'s as before
    - *zero-gradient*
    - *(w: zero @ top, bottom)*
  - X: BCs
    - **symmetric** B.C.'s
      - for W, P, θ
    - **asymmetric** in X
      - *only* for U



Program 5: Time integration

- Inside main program
  - Before integration loop:
    - tstep = Δt
  - Main integration loop, near top:
    - t2 = t1
    - u3 = u1
    - w3 = w1
    - *call* subroutines *advection, diffusion, pgf*

This is the first part of
$u^{n+1} = u^{n-1} + tstep*terms$ ...

- Remember:
  - *Theta is forward-time*
  - *Everything else is centered-time*

- Inside subroutines *advection, diffusion, pgf*
  - *t2 = t2 + Δt • [forcing terms]*
  - *u3 = u3 + tstep • [forcing terms]*
  - *w3 = w3 + tstep • [forcing terms]*

Program 5: Update

- Inside main program
  - Update step, bottom of integration loop
    - if *(this was the first time step)*

      u2 = u3 *copy n+1 data over (replacing) "n" array*

      w2 = w3 *copy n+1 data over (replacing) "n" array*

      p2 = p3 *copy n+1 data over (replacing) "n" array*

      t1 = t2 *copy n+1 data over (replacing) "n" array*

      tstep = 2•Δt *from now on, take 2Δt steps.*
    - otherwise *(time step 2 onwards)*

      u1 = u2; u2 = u3 *copy n > n-1, and n+1 > n*

      w1 = w2; w2 = w3 *copy n > n-1, and n+1 > n*

      p1 = p2; p2 = p3 *copy n > n-1, and n+1 > n*

      t1 = t2 *theta is forward time, always.*

Review: Program 5 coding

- Changes and additions for:
  - initial condition routine
    - no spatial variation specified for wind (or pressure)
    - multiple perturbations for theta
  - boundary condition routine
    - two dimensions: edges
  - main time step loop: starting
    - beginning the leapfrog time step; preparing theta
  - main time step loop: finishing
    - switching from forward to leapfrog time
  - routines
    - advection, diffusion, and pgf (pressure-gradient-force/buoyancy)

Program 5: Questions

- Ghost points – when? where?
  - To simplify things for myself I dimensioned almost everything 0:nx+1, 0:ny+1, i.e. one ghost point.
  - But what is really needed ? *discuss 1-D, 2-D*
- Official case not yet ready
  - *yes.*
- "nx" and "nz" refer to – what variable?
  - *theta* (potential temperature) and p (perturbation pressure).
- Grid points or cells?
  - *yes*. Consider as points except in context of Piecewise Linear method
- What limits of arrays? *discuss*
- Diffusion: X, Z, both, how? *discuss*
- Strang splitting – *not yet*

Program 5: Coding practice

- Starting a time step
  - Before doing anything else:
    - t2 = t1
    - u3 = u1
    - w3 = w1
    - All later routines *add to* these "n+1" arrays.

This also lets us turn processes on or off – we have taken the 'first part' of each time step – before starting.

  - So in *advection, diffusion, PGF, you will code …*
    - *t2 = t2 + … Δt • [forcing terms]*
    - *u3 = u3 + … $2\Delta t$ • [forcing terms] (same for W)*
  - Exception: pressure
    - Only one step to pressure: *p3 = p1 + $2\Delta t$ • [forcing terms]*

Program 5: First time step

- Straightforward coding would look like …
  - Forward step:
    - u2 = u1 + $\Delta t \bullet$ *[forcing terms]*
  - Centered step:
    - u3 = u1 + $2\Delta t \bullet$ *[forcing terms]*
  - Writing all that code out twice is annoying.
- Instead, we will do …
  - For the first time step, *tstep=Δt;* otherwise, *tstep=2Δt*
  - And so our equations look like …
    - u3 = u1 + tstep • *[forcing terms] (same for W, P)*
    - works because we *also* initialize our arrays u1=u2=0 *(same for W, P)*
  - *Except* for the temperature: θ is *always* forward in time.

Program 5: Where do I start?

- Suggested order of development for Program 5
  - Modify program 2 or 3 code to everywhere to assume NX ≠ NZ
  - Change physical dimensions; domain no longer [-0.5 :+0.5]
  - Create initial θ field, plot $\theta - \bar{\theta}$; verify it looks OK.
    - So the initial θ plot will look like a circular field surround by zeroes
  - Set up all arrays:
    - u, w, p: three time levels, 2-D
    - t: two time levels, 2-D
    - density (for p/t/u levels) and density for w-levels: 1-D
  - Create initial 1-D base-state fields for density
  - Create routine for boundary conditions
  - Test in this order: *PGF*; *linear* θ *advection* θ; θ *diffusion*.
  - *Now*: full physics.