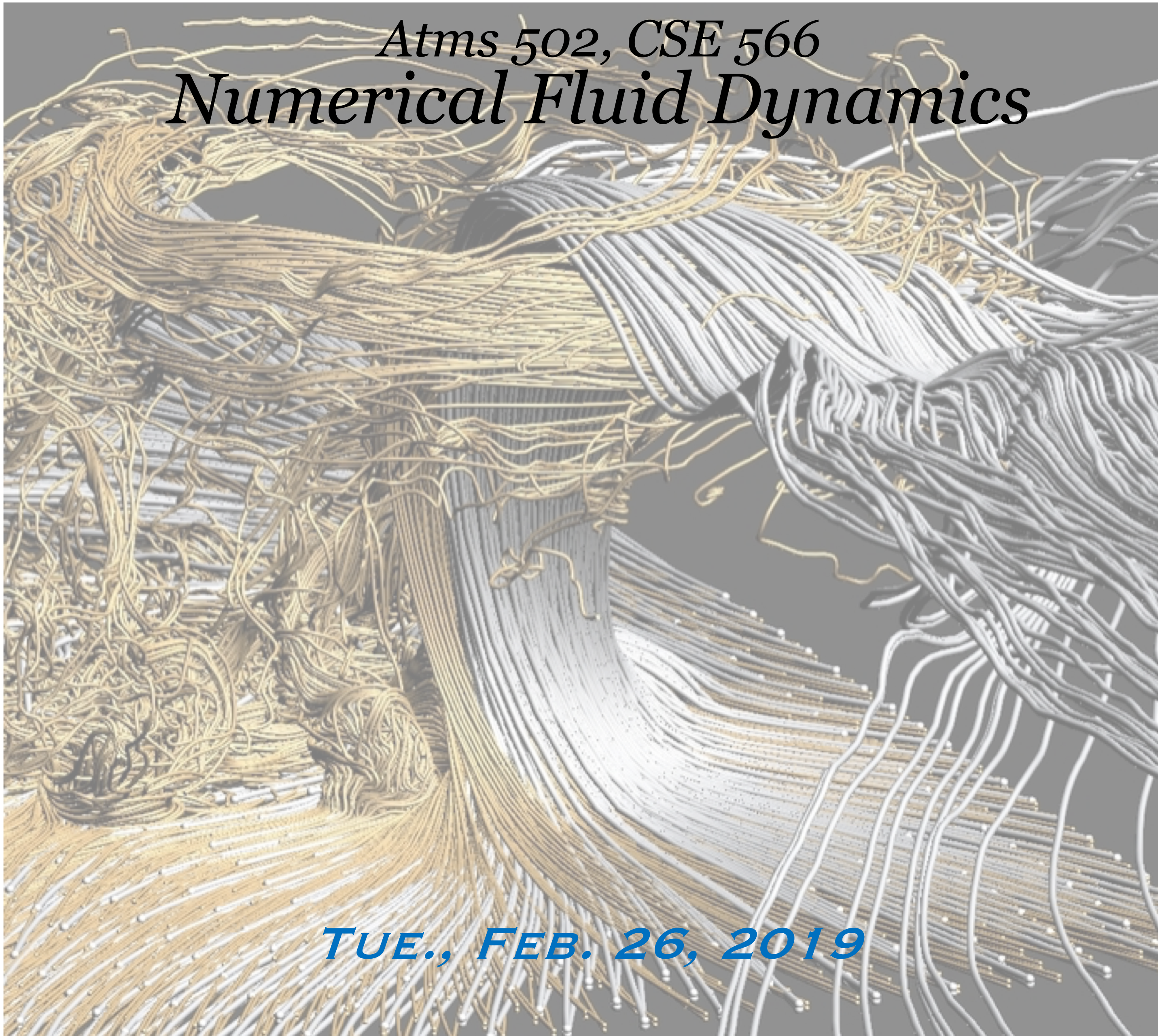
Atms 502, CSE 566
Numerical Fluid Dynamics



Tue., Feb. 26, 2019

Streamwise vorticity in a supercell thunderstorm
Science by Brittany Dahl and Amy McGovern - University of Oklahoma
Visualization by Greg Foss and Greg Abram, TACC
1500x1500x50 domain visualized with VisIt

https://www.tacc.utexas.edu/scientific-visualization-gallery#streamwise-vorticity-in-a-supercell-thunderstorm

ATMS 502
CSE 566

Tuesday,
26 February 2019

Class #13

• Pgm3 due Mar. ***5***

Plan for Today

- 1) Approximating derivatives
  - Back to Taylor series
- 2) Time differencing; Leapfrog
  - Computational *molecule*
  - Leapfrog method • stability analysis
- 3) Nesting, continued
  - Boundary conditions
  - 1-D view: interpolation, feedback
  - Grid refinement & clustering

Handouts:
- Skamarock & Klemp – AMR
- Skamarock – KE spectra & resolution

Approximating derivatives

- We have used **Taylor series** for truncation error...
- We can use the same series to **derive** approximations for derivatives

Approximations to derivatives

- Consider the first derivative in space -

$$\left(u_x\right)_j = \frac{u_{j+1} - u_j}{\Delta x} \; ; \; \left(u_x\right)_j = \frac{u_j - u_{j-1}}{\Delta x} \; ; \; \left(u_x\right)_j = \frac{u_{j+1} - u_{j-1}}{2\Delta x}$$

O(Δx) O(Δx) O(Δx^2)

$$\left(u_x\right)_j = \frac{4}{3}\left(\frac{u_{j+1} - u_{j-1}}{2\Delta x}\right) - \frac{1}{3}\left(\frac{u_{j+2} - u_{j-2}}{4\Delta x}\right)$$

O(Δx^4)

Following Wilhelmson

- Why *not* use higher-order approximations?
- When would you need 1-sided approximations?

Approximating derivatives: Example

- Example: centered 1st derivative
  - Let:

$$\frac{df}{dx} = f_x = \left[af(x - \Delta x) + bf(x) + cf(x + \Delta x) \right]$$

  - Insert Taylor series expansion; get 3 equations, 3 unknowns ...

$$\frac{df}{dx} = (a + b + c) f(x) \;+\; (c - a)\Delta x f_x \;+\; (a + c)\frac{\Delta x^2}{2!} f_{xx}$$

  - Result:

$$a = -1/2\Delta x,\ b = 0,\ c = 1/2\Delta x \ \therefore\ f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

Time differencing overview

SOME NOTATION AND BASIC IDEAS.

Time differencing; computational diagrams

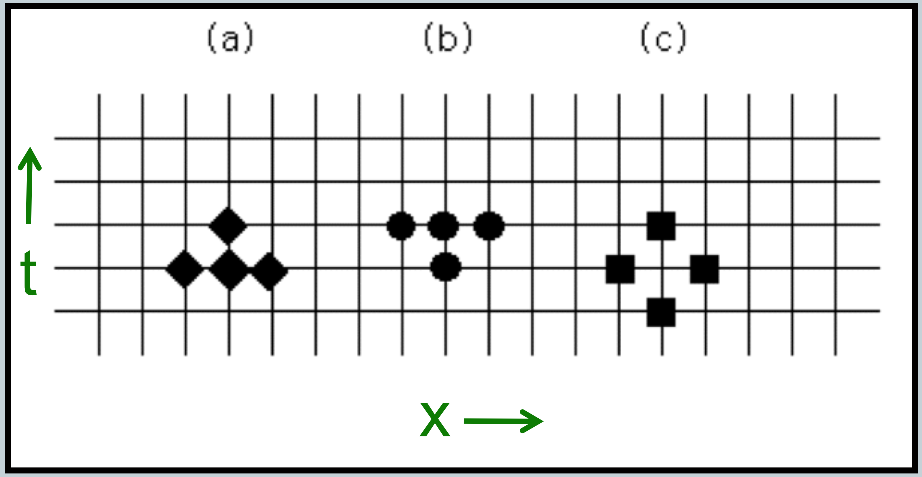
a) $$\frac{s_j^{n+1} - s_j^n}{\Delta t} + c\,\frac{s_{j+1}^n - s_{j-1}^n}{2\Delta x} = 0$$

b) $$\frac{s_j^{n+1} - s_j^n}{\Delta t} + c\,\frac{s_{j+1}^{n+1} - s_{j-1}^{n+1}}{2\Delta x} = 0$$

c) $$\frac{s_j^{n+1} - s_j^{n-1}}{2\Delta t} + c\,\frac{s_{j+1}^n - s_{j-1}^n}{2\Delta x} = 0$$

Computational *molecule*

Fwd time, ctr space | Back time, ctr space | Ctr time, ctr space



Wilhelmson

One-sided X difference: ____

Centered X difference: ____

Two time levels: ____________

Three time levels: ____________

Explicit scheme: ____________

Implicit scheme: ____________

The leap frog method

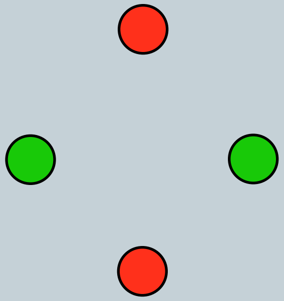
A 3-TIME-LEVEL SCHEME

Reference pages for this section:

- C003 – time levels
- C004 – numerical stencil
- C018 – complex numbers
- C020 – von Neumann analysis
- C052 – advection methods
- C054 – time differencing
- C055 – computational modes
- C056 – systems of linear equations
- C057 – eigenvalue problems
- C058 – characteristic equation
- C059 – time filtering

Leapfrog

- The *Leap Frog* method gets its name due to the way it makes use of data among 3 time levels.



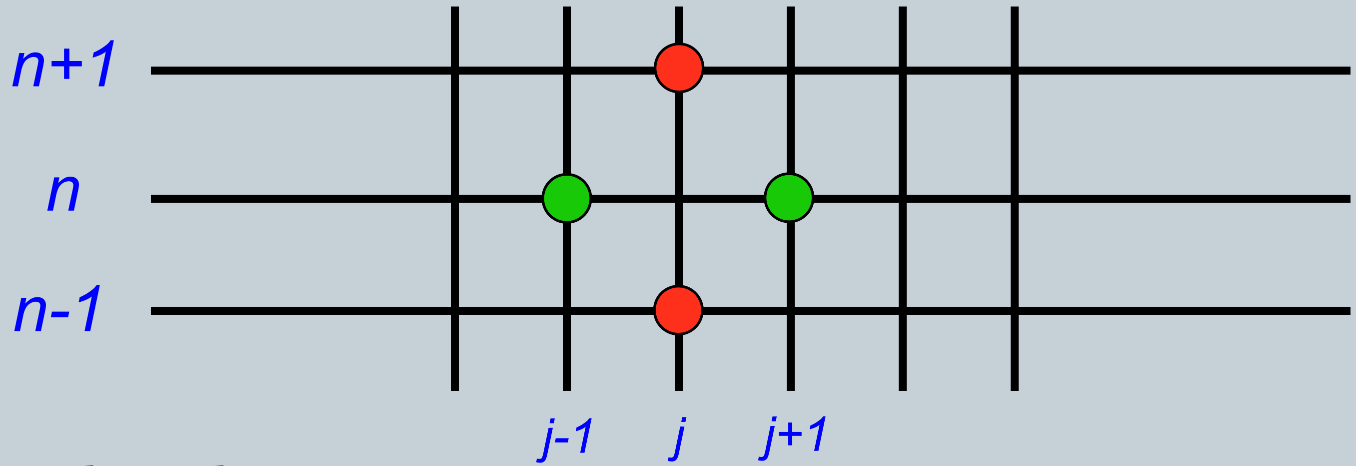
Leapfrog

- Leap frog method:

$$\frac{u_j^{n+1}-u_j^{n-1}}{2\Delta t} = -c\,\frac{u_{j+1}^{n}-u_{j-1}^{n}}{2\Delta x}$$

3-TIME-LEVEL SCHEME

- Computational molecule



- Time levels
  - evaluate space derivative at time *n*. Store *n*,*n-1* time levels.

Leapfrog - application

- Because Leapfrog has *3 time levels* …
  - Considerations:
    - Need "help" to get started: u2 = f(u1)
    - The first time step uses a *2-time-level method*

- Applying:
  - u3(j) = u1(j) - ν*(u2(j+1)-u2(j-1))
  - Update: copy u2 to u1; copy u3 to u2
  - u2 now contains latest results. Repeat.

Leapfrog - overview

- Leap frog method:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} = -c\,\frac{u_{j+1}^{n} - u_{j-1}^{n}}{2\Delta x}$$

- Accuracy
  - Leapfrog is consistent and accurate of order O[$(\Delta t)^2$, $(\Delta x)^2$].
  - There is considerable phase error.

$$u_t + c u_x = -\frac{(\Delta t)^2}{3!} u_{ttt} - c\,\frac{(\Delta x)^2}{3!} u_{xxx} + \ldots$$

- Stability
  - Leapfrog is stable for $|\nu| \le 1$
- Modes
  - There are *two* solutions from Leapfrog
    - These are the *physical* and *computational modes*
    - Results from the additional time level (higher accuracy in time)
    - A major drawback! Solution = sum of modes; comp. mode *undamped*

Leapfrog - stability

- Leap frog method:

$$u_j^{n+1} = u_j^{n-1} - \mu\left(u_{j+1}^n - u_{j-1}^n\right)$$

- Start out with usual Von Neumann method:

$$\tilde{u}^{n+1} = \tilde{u}^{n-1} - \mu\tilde{u}^n\left(e^{ik\Delta x} - e^{-ik\Delta x}\right)$$

- Introduce new variable - “looks” 2-time-level:

$$Let\ \tilde{v}^n = \tilde{u}^{n-1}, \text{ so } \tilde{v}^{n+1} = \tilde{u}^n, \text{ and we have :}$$

$$\tilde{u}^{n+1} = \tilde{v}^n - \mu\tilde{u}^n(2i\sin\beta)$$

$$\tilde{v}^{n+1} = \tilde{u}^n$$

Leapfrog – stability (2)

- We introduced a new variable ***v*** --

$$\text{Let } \tilde{v}^n = \tilde{u}^{n-1}, \text{ so } \tilde{v}^{n+1} = \tilde{u}^n, \text{ and we have:}$$

$$\tilde{u}^{n+1} = \tilde{v}^n - \mu\tilde{u}^n(2i\sin\beta)$$

$$\tilde{v}^{n+1} = \tilde{u}^n$$

- Now write in matrix form.

$$\left.\begin{aligned} \tilde{u}^{n+1} &= \tilde{v}^n - \mu\tilde{u}^n(2i\sin\beta) \\ \tilde{v}^{n+1} &= \tilde{u}^n \end{aligned}\right\} \text{ so } \begin{pmatrix} \tilde{u}^{n+1} \\ \tilde{v}^{n+1} \end{pmatrix} = \begin{pmatrix} -2i\mu\sin\beta & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \tilde{u}^n \\ \tilde{v}^n \end{pmatrix}$$

Leapfrog stability (3)

- Linear algebra: y=Ax, Ax=λx, (A-λI)x=0.
  - We were here:

$$\begin{pmatrix} \tilde{u}^{n+1} \\ \tilde{v}^{n+1} \end{pmatrix} = \begin{pmatrix} -2i\mu\sin\beta & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \tilde{u}^{n} \\ \tilde{v}^{n} \end{pmatrix}$$

  - For a nontrivial solution, the *characteristic determinant* det(A-λI)=0:

$$\begin{vmatrix} -2i\mu\sin\beta - \lambda & 1 \\ 1 & 0 - \lambda \end{vmatrix} = 0 \qquad (1)$$

  - (1) is the *characteristic equation* corresponding to our matrix (*A*).

Leapfrog stability (4)

- Characteristic equation:

$$\begin{vmatrix} -2i\mu\sin\beta-\lambda & 1 \\ 1 & 0-\lambda \end{vmatrix} = 0$$

- Solve:

$$\lambda^2 + 2i\mu\sin\beta\lambda - 1 = 0$$

- Two roots: things are getting interesting.

$$\lambda = -i\mu\sin\beta \pm \sqrt{1-\mu^2\sin^2\beta} = -ip \pm \sqrt{1-p^2}$$

  - If the square root is real, $|\lambda|^2=1$ and $|\mu|\leq 1$.
  - If the square root is imaginary, $|\lambda|>1$.
  - Our stability condition is: $|\mu|\leq 1$

NO AMPLIFICATION ERROR!

Leapfrog - modes

- We have *two modes* to the solution.
  - This comes from the ± below.

$$\lambda = -i\mu \sin\beta \pm \sqrt{1-\mu^2 \sin^2\beta}$$

  - One is real (physical). One is computational.

Leapfrog - modes

- We have *two modes* to the solution.
  - This comes from the ± below.

$$\lambda = -i\mu \sin\beta \pm \sqrt{1-\mu^2\sin^2\beta}$$

  - One is real (physical). One is computational.
  - To find out which is which, take $\Delta t, \Delta x \rightarrow 0$.
    - Then μ goes to 0; one root goes to +1, one to -1.
    - Root of +1 is physical; *no growth*, all is well.
    - Root of -1: *switches sign every time step* (λ^n).
  - This is a not-so-good consequence of our 3-time-level numerical scheme.

Leapfrog stability - review

- We rewrote the 3-level scheme as 2-level:

$$\left.\begin{aligned}\tilde{u}^{n+1} &= \tilde{v}^{n} - \mu\tilde{u}^{n}(2i\sin\beta) \\ \tilde{v}^{n+1} &= \tilde{u}^{n}\end{aligned}\right\} \text{ so } \begin{pmatrix}\tilde{u}^{n+1} \\ \tilde{v}^{n+1}\end{pmatrix} = \begin{pmatrix}-2i\mu\sin\beta & 1 \\ 1 & 0\end{pmatrix}\begin{pmatrix}\tilde{u}^{n} \\ \tilde{v}^{n}\end{pmatrix}$$

- Write above as matrix, subtract l from diagonal, set determinant to zero. Characteristic equation:

$$\begin{vmatrix}-2i\mu\sin\beta - \lambda & 1 \\ 1 & 0 - \lambda\end{vmatrix} = 0$$

- Solve; 2 roots; physical and computational modes

$$\lambda = -i\mu\sin\beta \pm \sqrt{1 - \mu^2\sin^2\beta} = -ip \pm \sqrt{1 - p^2}$$

- As Δt and p ⇒0: ”+” root approaches 1, “-” root: -1
  - |λ|=-1 means amplitude varies as $(-1)^n$

Nesting

Nested grid BCs

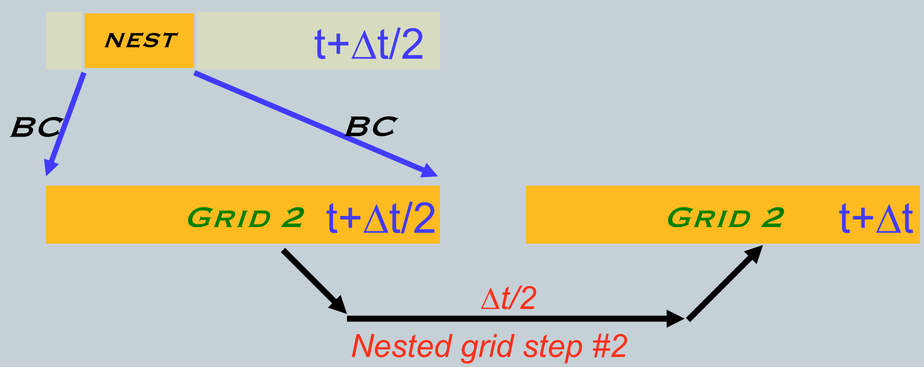
- Nested grid:
  - Shown below: grid-1 time step, q1 to q2
  - Added: nested grid step, refinement factor

2

$$q1_{grid\,2} = (1-F)\,q1_{grid\,1} + F \bullet q2_{grid\,1}$$

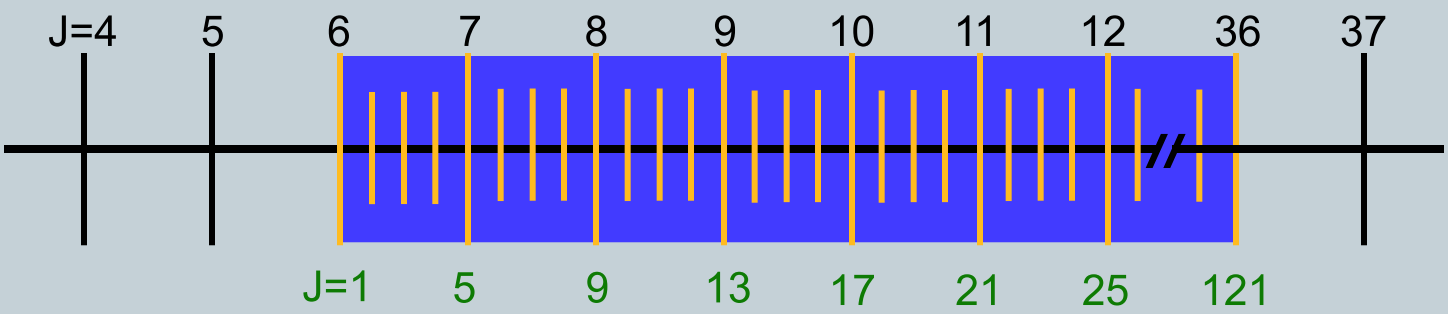
$$\text{where} \quad F = \frac{\left(nstep_{grid\,2} - 1\right)}{\#\,steps_{grid\,2}}$$

Time-interpolated boundary conditions



Interpolation

- *Interpolation: coarse ⇒ nested*

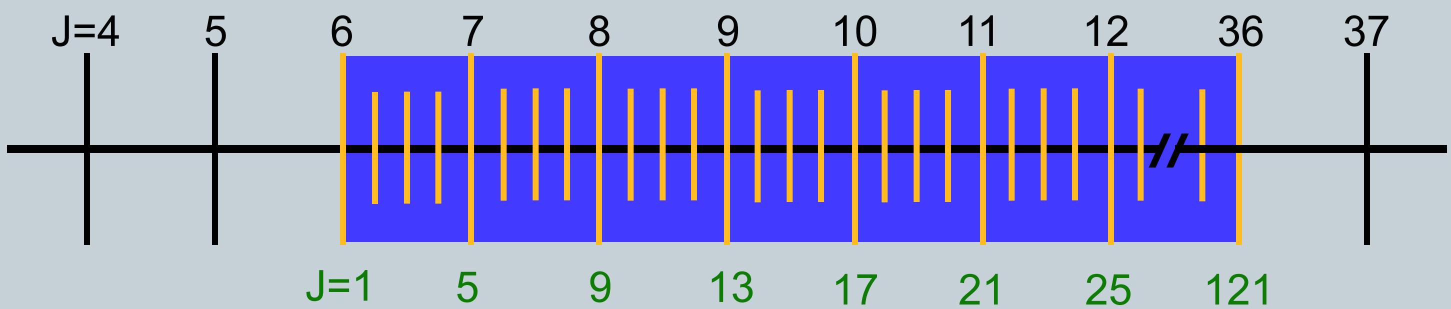


Nested grid size … example

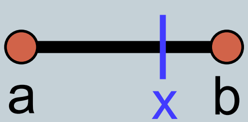
- nx=121 *(both coarse and nested grids)*
- nested grid is 121 *nest* points wide, and
- nested grid is (121-1)/4=30 *coarse* points wide

Interpolation

- *Interpolation: coarse ⇒ nested*



NESTED GRIDS: *INTERPOLATION*

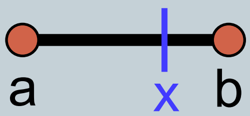


$$f(x) = f(a) + \left[f(b) - f(a) \right] \left(\frac{x - a}{b - a} \right)$$

Interpolation

- *Interpolation: coarse ⇒ nested*

NESTED GRIDS: *INTERPOLATION*

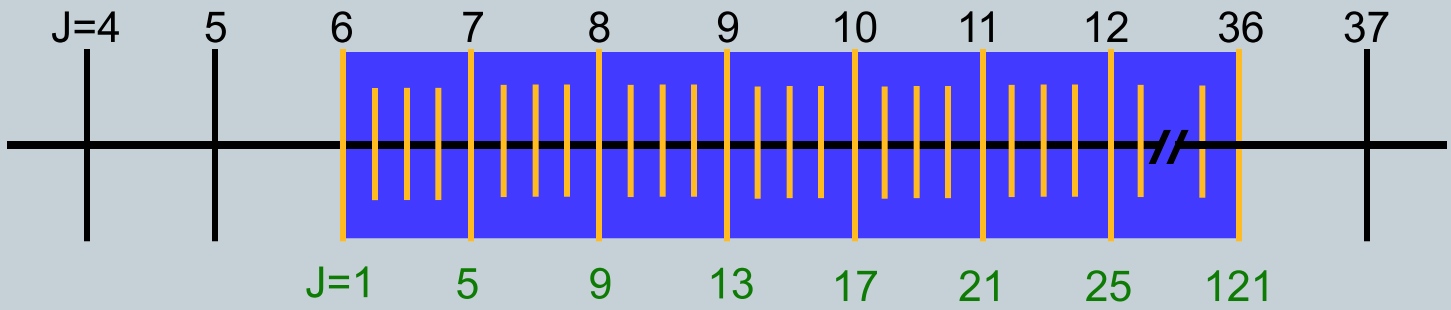


$$f(x) = f(a) + \left[f(b) - f(a)\right]\left(\frac{x-a}{b-a}\right)$$

```
nest(inest) = coarse(icoarse) +
                  ( coarse(icoarse+1)-coarse(icoarse) )*fraction
```

Interpolation

- *Interpolation: coarse ⇒ nested*

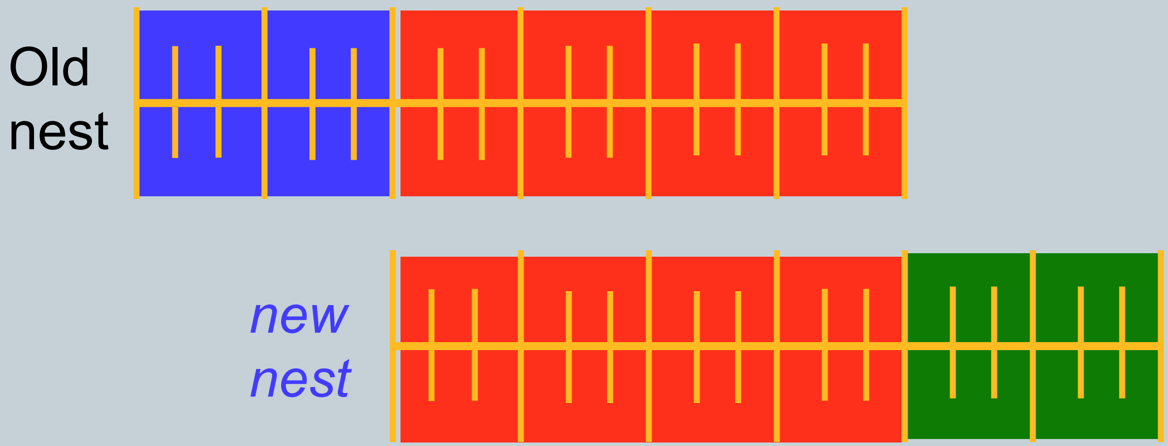


NESTED GRIDS: *COORDINATES*

- Example above: nested grid index *j=1* … is at coarse grid index *J=6*.
- icoarse = (inest-1)/ratio + first.nest.point

Interpolation

- *Interpolation: old vs. new nested grids*



THIS EXAMPLE: 3:1 NESTING

- Nested grid re-location:
  - ✓ interpolate from coarse ⇒ new nested grid
  - ✓ copy <u>overlap region</u> of old nest to new nest

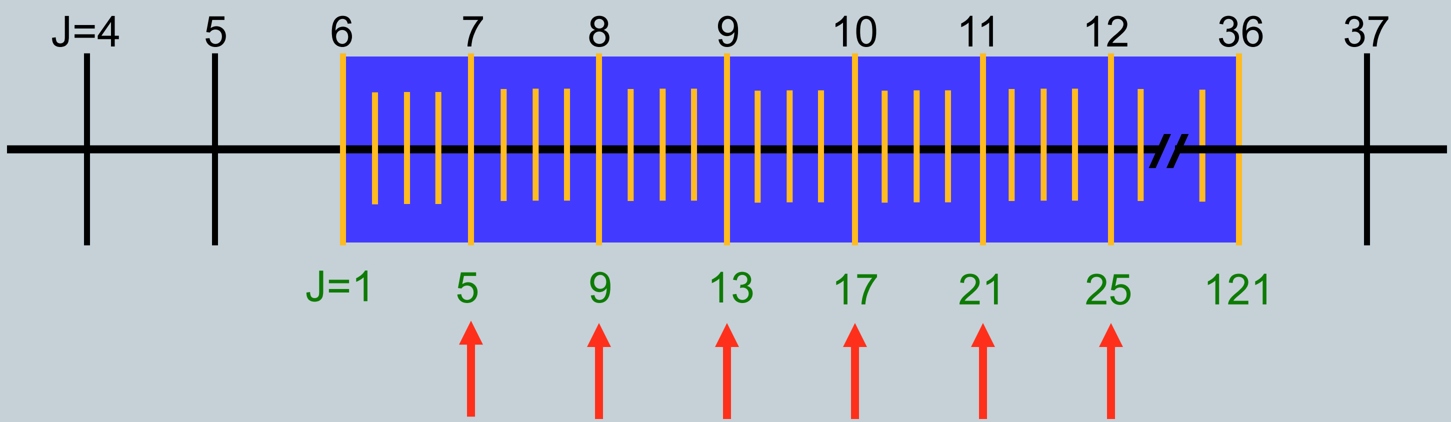
Interpolation

How often should we relocate the nest?

- As often as possible?
  - Minimizes copying coarse data $\Rightarrow$ nest
  - This is more computationally expensive
    - Imagine computing the truncation errors over a larger 2d domain, frequently
- As rarely as possible?
  - Eventually features of interest leave the nest
  - Much of new nest would then be copied from coarse grid

Feedback

- *Feedback: copy nested ⇒ coarse*



UPDATING THE COARSE GRID

- We require every 4th nest point to overlap a coarse point
- In feedback, every 4th *interior nested grid point* is copied back to the coarse grid.
- What alternate approach might we consider ??

Grid refinement & Clustering

ADAPTIVE MESH REFINEMENT

Reference pages for this section:

- C008 – Truncation error
- C009 – Resolution
- C010 – AMR / nesting
- C051 – Nesting: grid placement, movement

Regridding procedure

- Regridding (Skamarock dissertation, pp. 12-13)
  - 1) flag points needing refinement
    - flagged if estimated error exceeds user threshold
  - 2) cluster the flagged points – for two reasons
    - a) separates "spatially distinct phenomena [like] shocks or fronts"
    - b) subdivide to use several grids instead of one large region
  - 3) fit rectangular grids around the clustered points
  - 4) repeat steps 2,3, using different methods if necessary
    - simple method [nearest neighbor] – ok for clustering, not rectangles
    - connecting points – use *minimum spanning trees* or *nearest neighbor graphs*
  - clustering, fitting rectangles "most difficult part of regridding"

Regridding / clustering

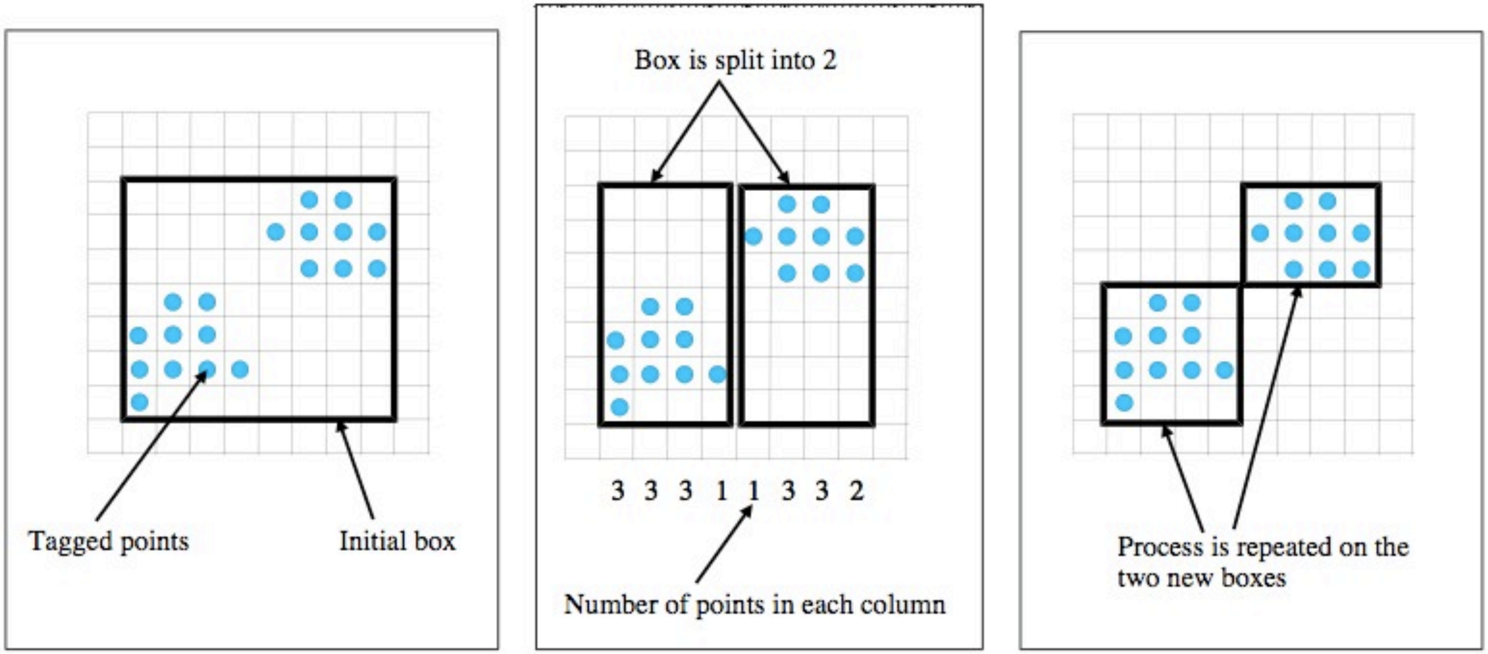
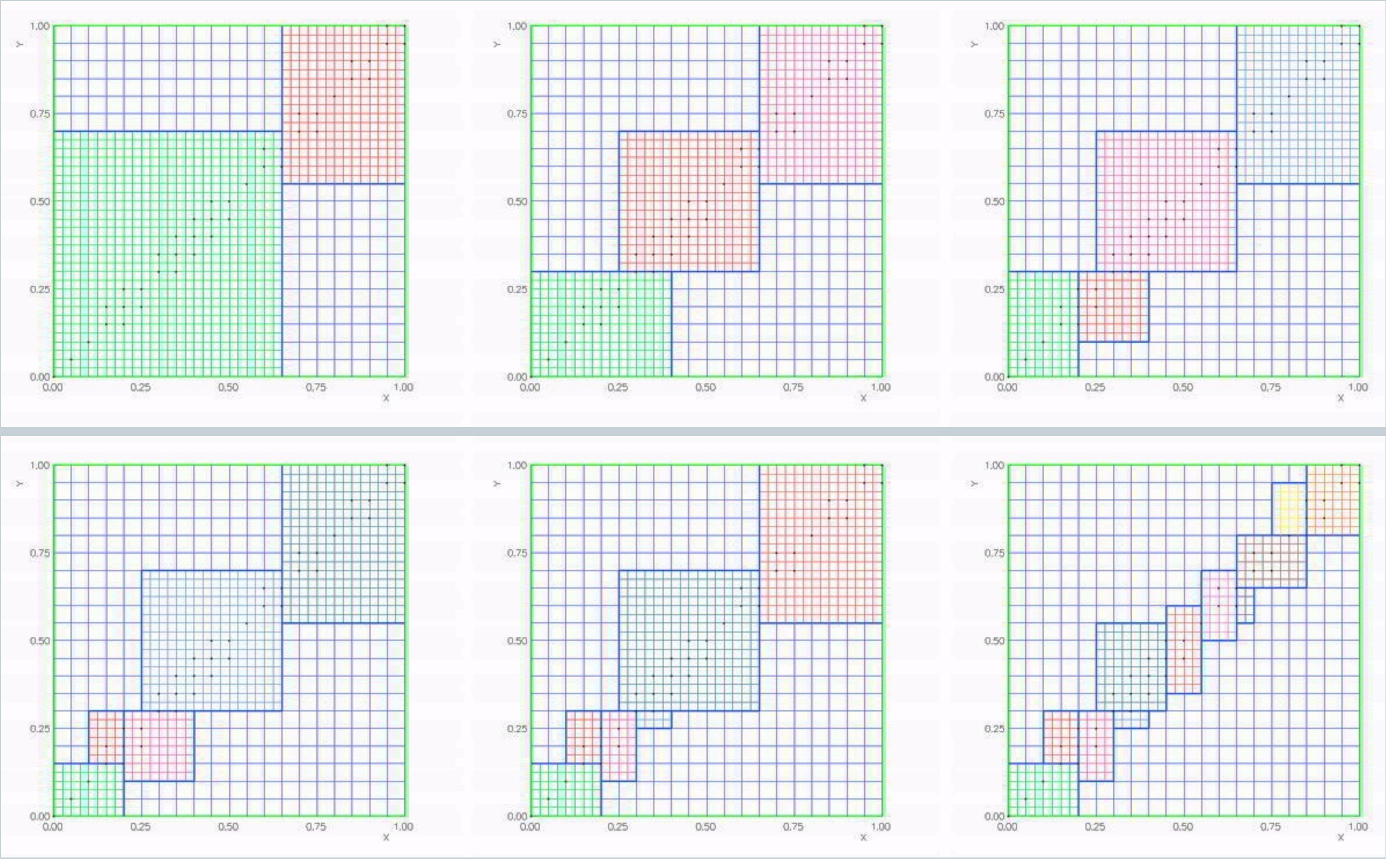


Figure 2: The 3 basic steps in regridding are (1) tag error cells and enclose in a box, (2) split the box into 2 based on a histogram of the column or row sums pf tagged cells, (3) fit new boxes to each split box and repeat if the ratio of tagged to untagged cells is too small.

"Adaptive mesh refinement routines for Overture" - *William Henshaw, 2011 (link)*

Optimal grid size, number, locations

Regridding / clustering



"Adaptive mesh refinement routines for Overture" - *William Henshaw, 2011 (link)*

Optimal grid size, number, locations