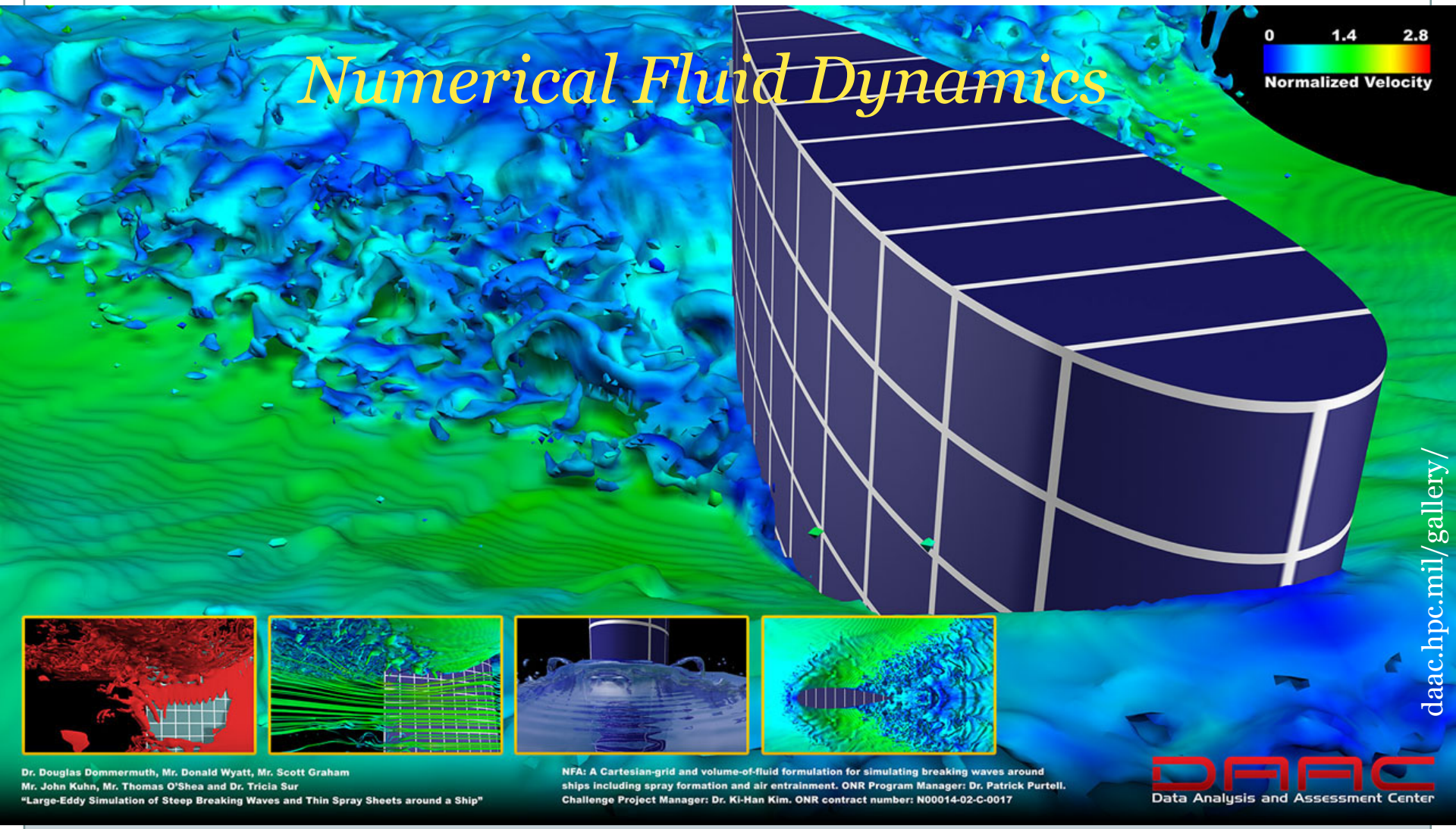


Class 15 - Tue., Mar. 5, 2019

Image: **Breaking waves around ship** including spray

ATMS 502
CSE 566

Tuesday,
5 March 2019

Class #15

- Pgm3 due Thu Mar. ***7***

Plan for Today

- 1) Space differencing
  - Differential-difference approach
  - Phase speed & group velocity
  - Impacts of higher order, added diffusion
- 2) Programming input/output
  - C and Fortran considerations
- 3) Program 4: *continued*
  - Nest placement / movement
- 4) Skamarock & Klemp
  - Global/local refinement
  - A full AMR method, explained

Space differencing

Reference pages for this section:

- Durran (2010) § 3.3, p. 100+
- C008 – Truncation error
- C009 – Resolution
- C010 – AMR / nesting
- C051 – Nesting: grid placement, movement

Review: Poorly resolved waves

- Centered schemes preserve amplitude of Fourier components, but dispersion yields numerical dissipation in solution
- "Almost all finite different schemes fail to propagate the $2\Delta x$ wave.
- Negative group velocities for $2\Delta x$ waves
- Higher accuracy does *not* help with poorly-resolved waves.

Overview: Space differencing

- Task: Isolate *just* the spatial errors
  - differential-difference equation
    - leave time derivative: as a derivative
    - apply differencing to spatial derivative
    - insert complex exponential involving time
    - simplify to obtain frequency
  - computing phase speed
    - frequency divided by wavenumber
    - if phase speed depends on k: dispersive.
      - if this was a linear, constant speed problem, wavenumber dependence *c(k)* represents *numerical error*.
  - compute group velocity
    - derivative of frequency w.r.t. wavenumber

Space Differencing

- *How to address space/time differencing separately?*
  - Space: *differential-difference equations*
    - Wherein we say: *what* time differencing?
    - Let's look at just *2nd-order centered space*

$$\frac{du_j}{dt} + c\left(\frac{u_{j+1} - u_{j-1}}{2\Delta x}\right) = 0$$

• *no n+1,n… superscripts*

    - *And* we substitute a slightly different form:

$$u_j(t) = e^{i(kj\Delta x - \omega_{2c} t)}$$

    - Key point: frequency ω can *still* be complex!!
      - Why is this important? (we'll discuss this further, later)

Space Differencing

- Frequency eqn *for 2nd-order ctr space*

$$\frac{du_j}{dt} + c\left(\frac{u_{j+1} - u_{j-1}}{2\Delta x}\right) = 0$$

substitute: $u_j(t) = e^{i(kj\Delta x - \omega_{2c} t)}$

- We get frequency equation:

$$-i\omega_{2c} u = -\frac{c}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right)u \quad \Rightarrow \quad \omega = \frac{c\sin\beta}{\Delta x}$$

Real *frequency: no amplitude error. See Durran § 3.3.1*

- Is this dispersive? Yes. *c(k) !*

$$c_{2c} = \frac{\omega}{k} = \frac{c\sin\beta}{k\Delta x}; \text{ for } \textit{small } \mathrm{k}\Delta\mathrm{x}, c_{2c} \approx c\left(1 - \frac{\beta^2}{6}\right)$$

Note $\beta \to 0$ case

Space Differencing

- Our expression *for the phase speed*:

$$c_{2c} = \frac{\omega}{k} = \frac{c \sin\beta}{k\Delta x}; \quad \text{for small k}\Delta\text{x}, \; c_{2c} \approx c\left(1 - \frac{\beta^2}{6}\right)$$

  - *Lagging phase error. Note $c_{2c}(2\Delta x)=0$!*

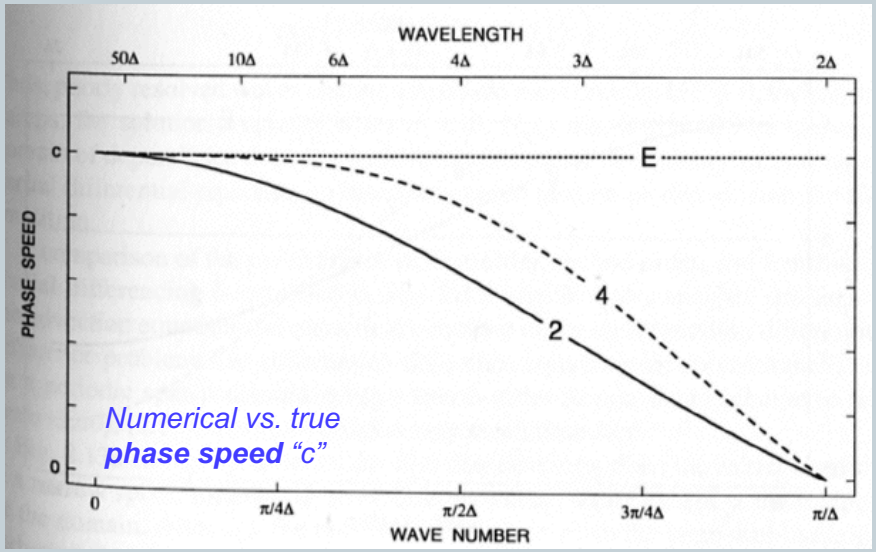
- Group velocity:

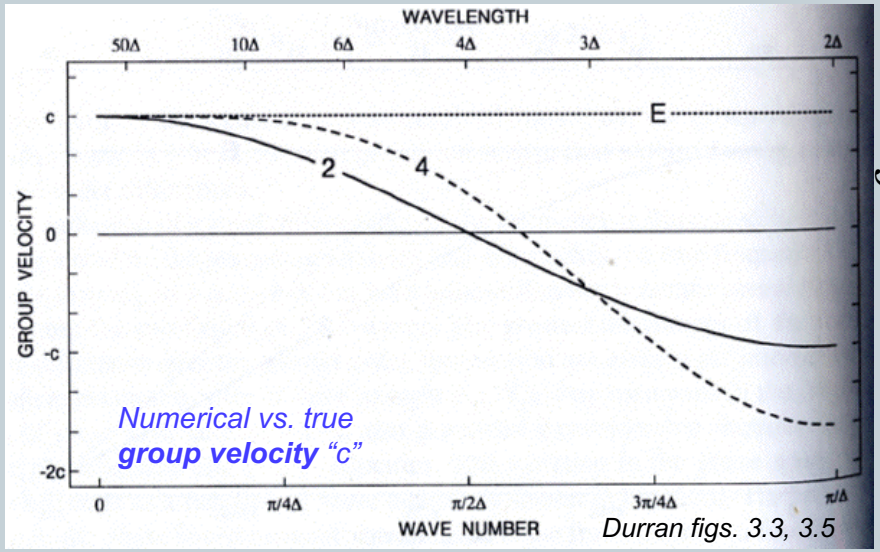
$$c_{g2c} = \frac{\partial\omega}{\partial k} = \frac{\partial}{\partial k}\left(\frac{c \sin\beta}{\Delta x}\right) = c(\cos\beta)$$

  - *good for small β (long waves);*
  - *terrible for small waves!! (why?)*

2^{nd} vs. 4^{th} order space diff.

- *Summary: 2nd-order centered space differencing*





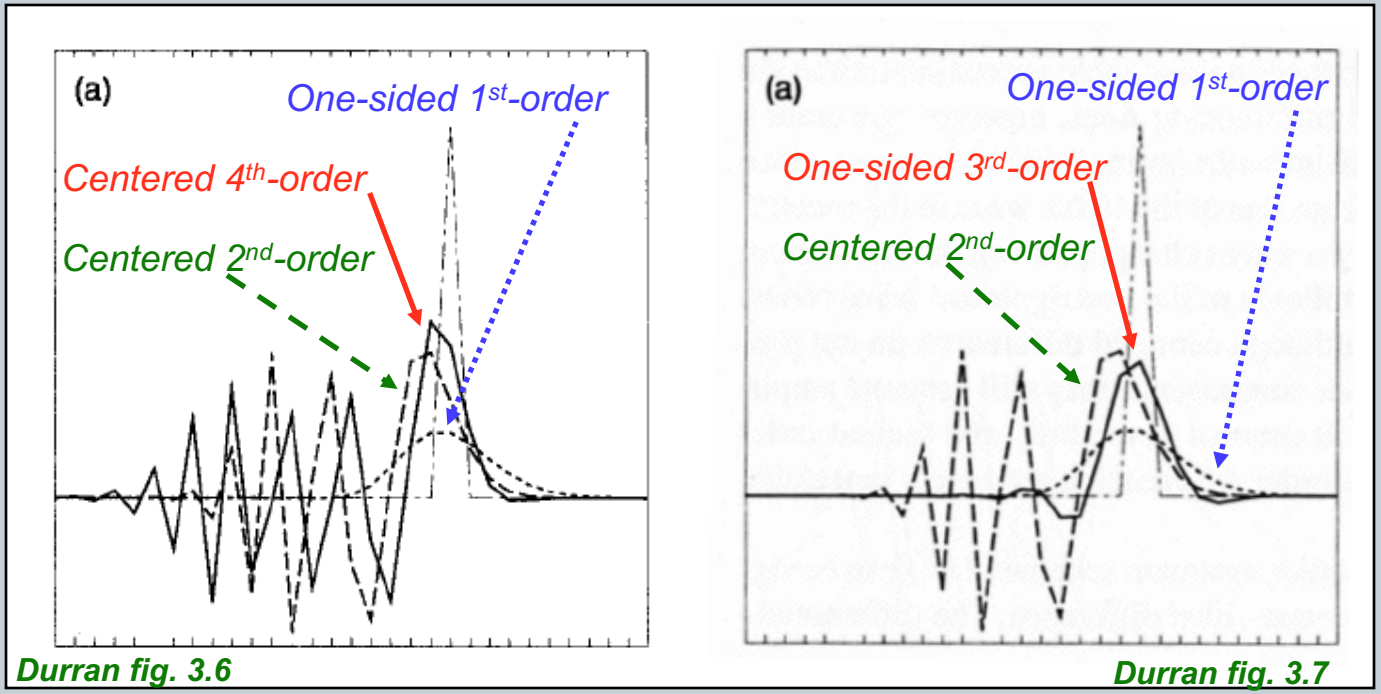
Durran figs. 3.3, 3.5

higher is better !

- Phase speed (left), group velocity (right)
- *Second order (solid), 4th order (dashed)*
- *4th better at "intermediate" wavelengths*

1st - 4th order space differencing

- *Higher centered vs. odd-ordered schemes*



Durran fig. 3.6

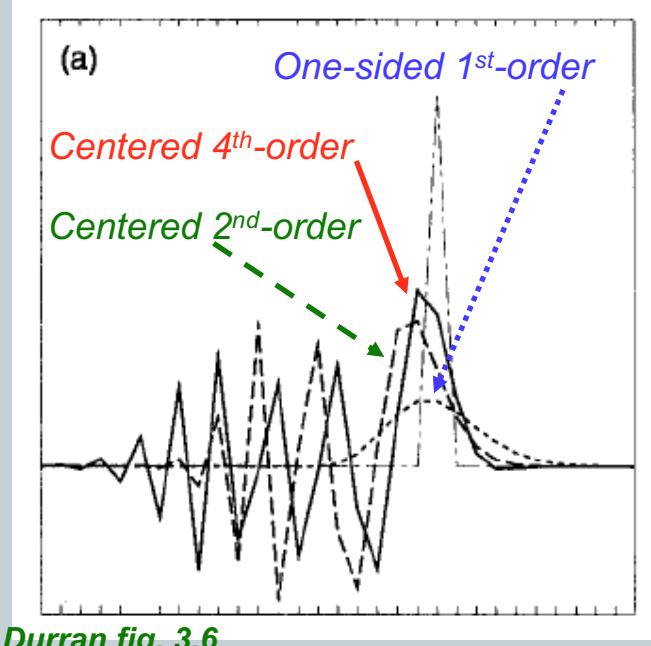
Durran fig. 3.7

1-sided 1st order vs. centered 2nd,4th

1-sided 1st, 3rd order vs. centered 2nd

Distortion of the solution

- "**Centered schemes** preserve the amplitude of each individual Fourier component, [but] the various *components propagate at different speeds*, and thus the superposition of these components ceases to properly represent the true solution"
  - *Durran pp. 106-107*
  - With phase errors, wavelengths interfere – causing amplitude errors

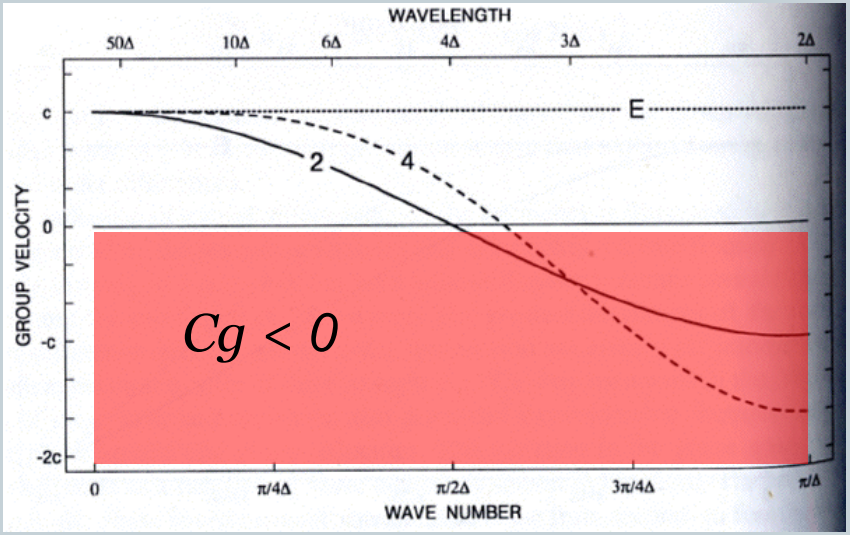


Durran fig. 3.6

Group velocities

- "In the *absence of dissipation* the large negative group velocities associated with the 2Δx wave rapidly spread short wavelength noise away from regions where 2Δx waves are forced."

  - Durran pp. 106-107



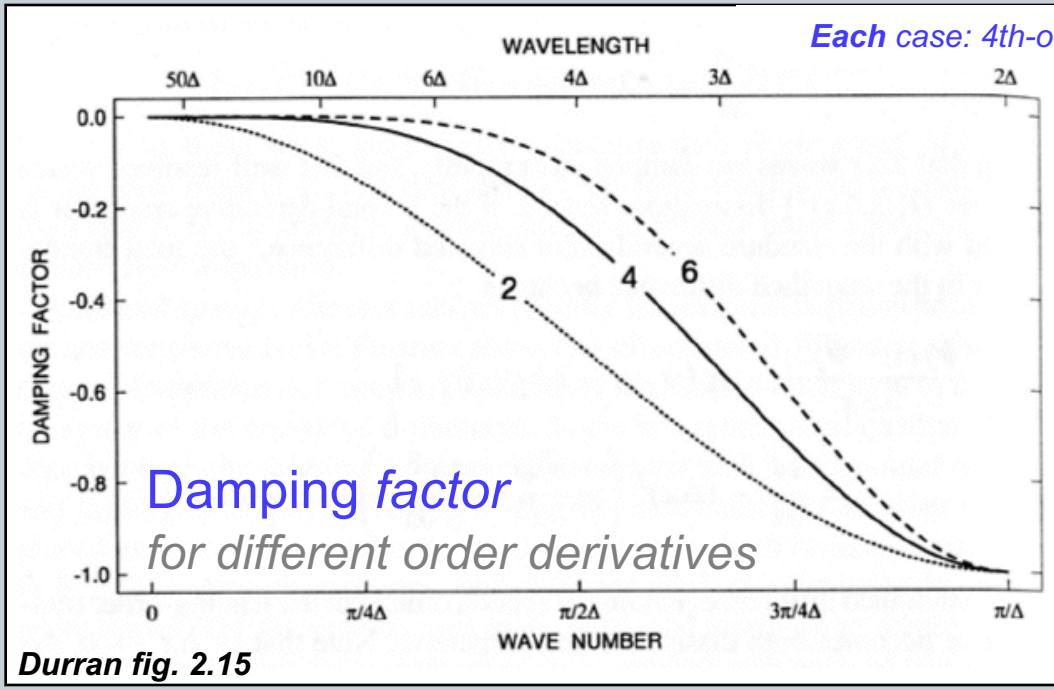
Group velocities

- "The upstream propagation of the 2Δx wave [is worse with 4th order than 2nd].
- "Switching to a higher-order scheme does **not** improve the performance of finite-difference methods *when they are used to model poorly resolved features* like the spike … in many respects the **4th-order solution is worse** than the 2nd-order result.

  - Durran pp. 106-107

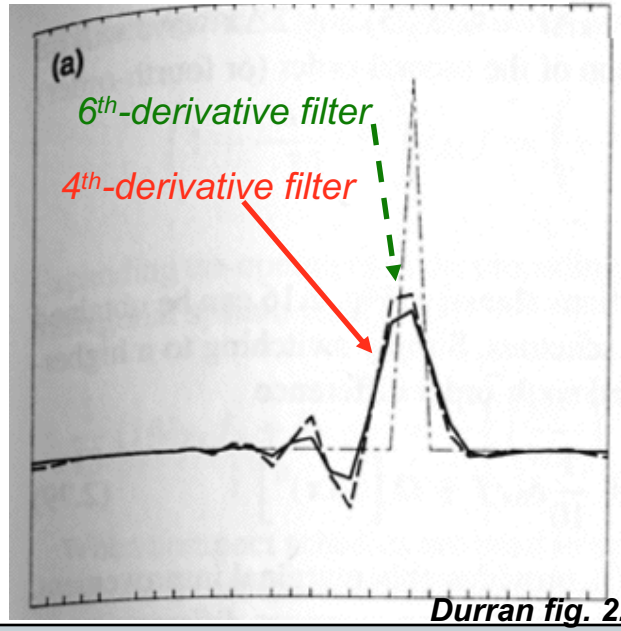
Explicit Artificial Dissipation

- *Added explicit damping can be beneficial*



Durran fig. 2.15

***Each** case: 4th-ordered centered space **differencing**!*

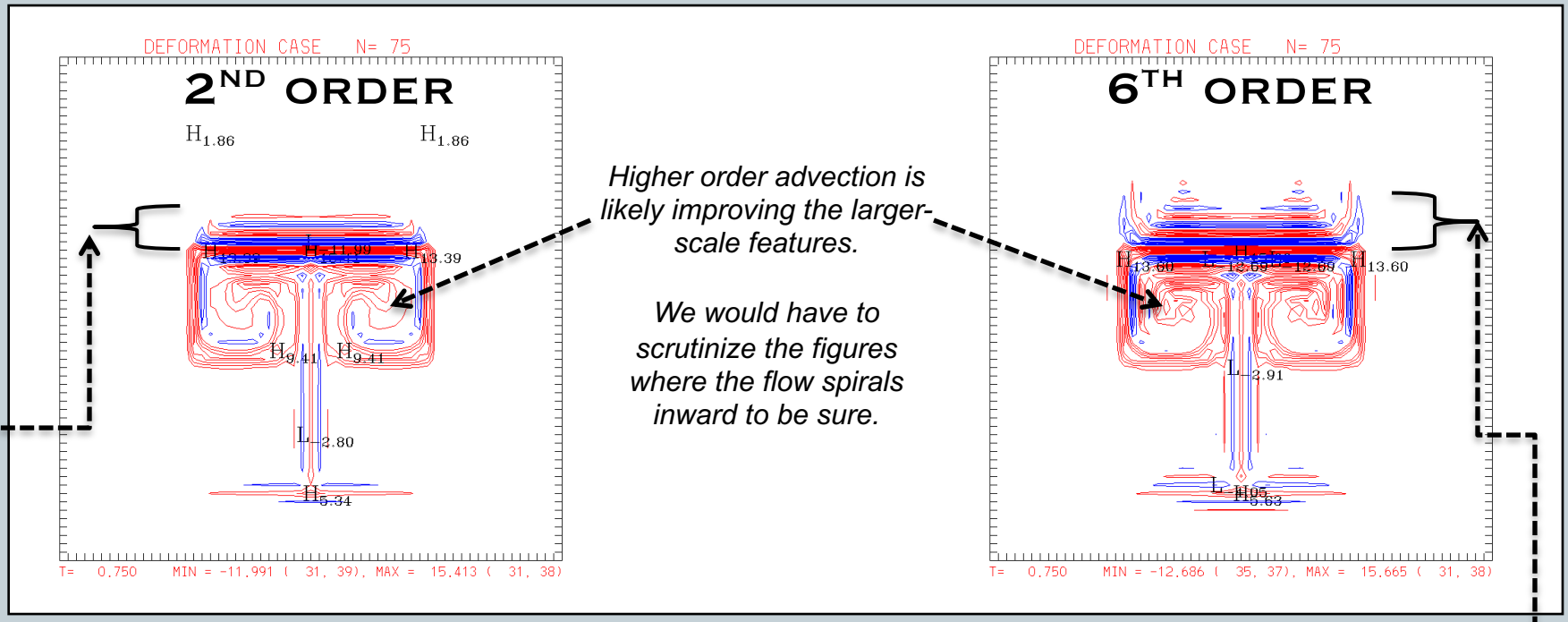


Durran fig. 2.16

Without filtering, errors w/dispersion of poorly-resolved Fourier components propagate without damping. Explicit diffusion reduces amplitude of these short waves.

Review: Added Dissipation

- No damping here; just deformational flow tests with 2nd order (Lax-Wendroff) and 6th-order (Crowley) advection.



But notice the packing of contours at the top. *This appears* ***worse*** *for 6th order, and probably represents greater dispersion in the region of sharpest gradients, in the deformational flow near the top of each figure.*

Programming input/output

To learn more:

- **web-search for**
  - *Linux I/O redirect*
  - *[Fortran or C] I/O*
  - *XML programming [Fortran or C]*
  - *NetCDF example [Fortran or C] or go here*

Linux I/O

- Three classes of I/O, and ways to *redirect* I/O:
  - stdin: standard input
    - Fortran: *read*,nplot* or *read(5,*) nplot*
    - C: *scanf("%d", &nplot);*
    - Linux shell: *program < filename*
  - stdout: standard output
    - Fortran: *print*,nplot* or *write(6,*) nplot*
    - C: *printf("%d", nplot);*
    - Linux shell: *program > filename* or *program >> appendFile*
    - Linux shell, in+out: *program < inputFile > outputFile*
  - stderr: standard error
    - stderr: the *other* output stream …
    - this can also be *redirected*.
    - the syntax depends on your command interpreter (shell).
      - C-shell & *tcsh*: redirect *both* stdout, stderr with: *program >& file*
      - Bash: redirect *both* stdout, stderr with: *program > file 2>&1*

A short script to run your program

- So are you **tired** of this yet?
  - login2% pgm2
    360
    .075
    5
    y
- *So* much better to run your program with the input specified directly in a script.
  - See example at right!
  - Good resource to get answers to your programming questions:
    stackoverflow.com

- In a *tcsh* or *bash* script:

```
#!/bin/tcsh  or  #!/bin/bash
pgm2 << EOF
360
.075
5
y
1
...
EOF
```

EOF is a common LINUX abbreviation, short for "End of File". Really any word will work here!!

```
pgm2 << SickOfPgm2
 ...
SickOfPgm2
```

- Don't forget to make the script ***executable***

  chmod u+x myscript
  - adds *execute* permission for the ***user*** (you).

Fortran and C: read/write text to file

Fortran

- read from file directly:

```
open(1,file='filename', &
   status='old')
read(1,*) nplot
read(1,*) morestuff
close(1)
```

- write to file directly:

```
open(1,file='newfile', &
    status='unknown')
rewind(1)
write(1,*) nplot
close(1)
```

C

- read from file directly:

```
#include <stdio.h>
FILE *fp,*fopen();
fp=fopen("filename","r");
fscanf(fp,"%d", &nplot);
fclose(fp);
```

- write to file directly:

```
#include <stdio.h>
FILE *fp,*fopen();
fp=fopen("newfile","w");
fprintf(fp,"%d", nplot);
fclose(fp);
```

Fortran input via a *namelist*

- The code looks like:

```
namelist /namelist_name/ variable1, variable2, variable3 …
! … you still have to declare all these variable types!
open(1,file='namelist_filename',status='old')
read(1, NML=namelist_name)
close(1)
```

- The namelist file looks like:

```
&namelist_name
variable1   = -12.45,
variable2   = 73.123,
variable3   = .true.,
/
```

Reading and writing *XML*

- XML – eXtensible Markup Language
  - designed to carry data.
  - example at right

```
<?xml version="1.0"?>
<p5input>
   <dt> 0.5</dt>
   <dx>100</dx>
</p5input>
```

- For Fortran: *not really supported, but try these:*
  - http://xml-fortran.sourceforge.net
  - http://homepages.see.leeds.ac.uk/~earawa/FoX/
  - https://www.sciencedirect.com/science/article/pii/S235271101630036X
- For C:
  - Google "C xml parser library" : code.google.com/p/libroxml
  - Gnome xml C parser, libxml : xmlsoft.org; also, wikipedia
  - https://stackoverflow.com/questions/9387610/what-xml-parser-should-i-use-in-c

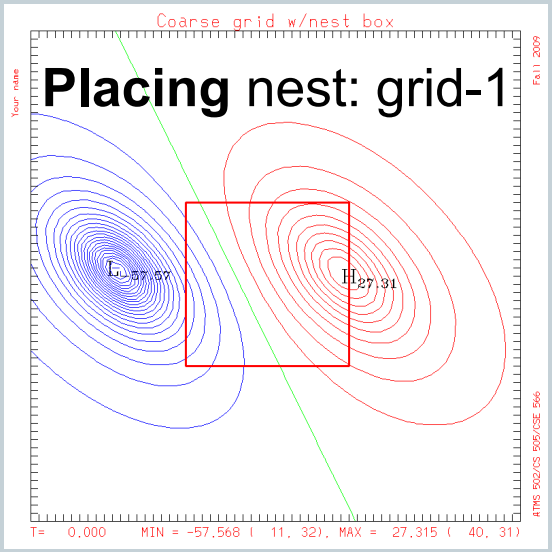
Reading and writing *NetCDF*

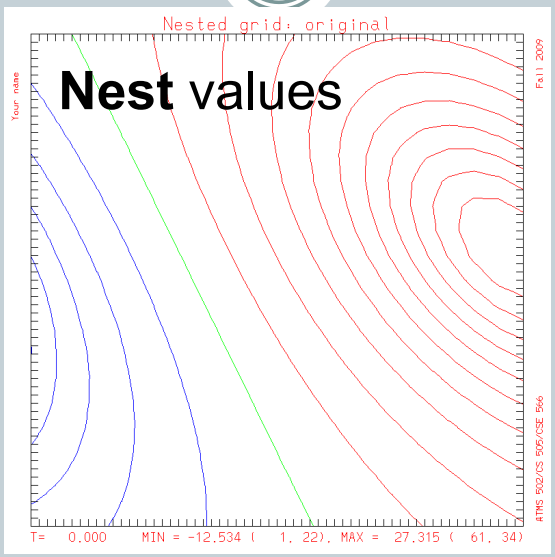
- NetCDF is a data format with C and Fortran *APIs*
  - an API is *an Application Program Interface*
    - *"a set of routines, protocols and tools for building software applications. An API specifies how software components interact"* - webopedia
- NetCDF data has *your* data but also *metadata*
  - *metadata* is a set of data that describes and gives information about other data *(google)*
  - so your file could contain your array dimensions, field names, field *units*, and of course the data itself.
- Learn more here:
  - https://www.unidata.ucar.edu/software/netcdf/examples/programs/

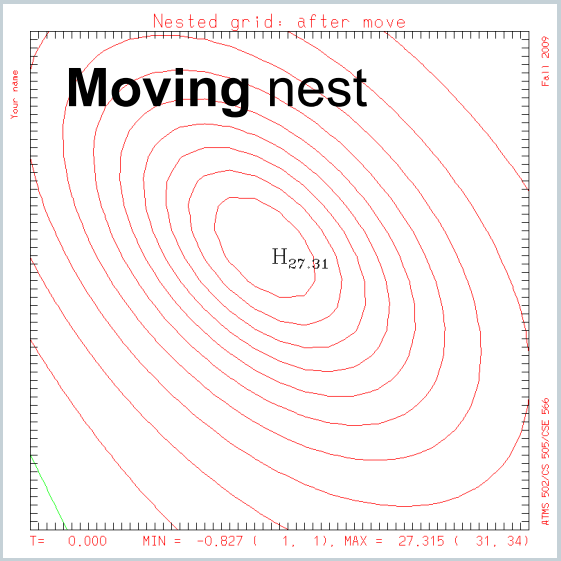
Program 4

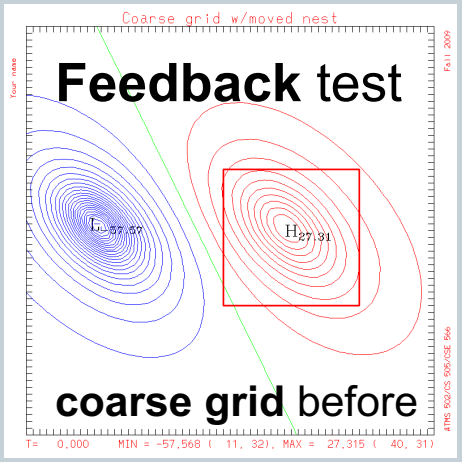
NEST PLACEMENT ALGORITHM

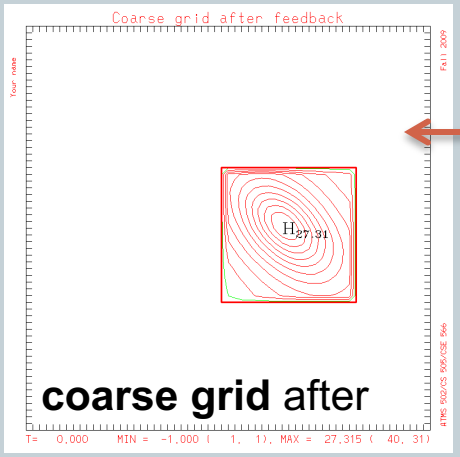
Review: Nesting routines









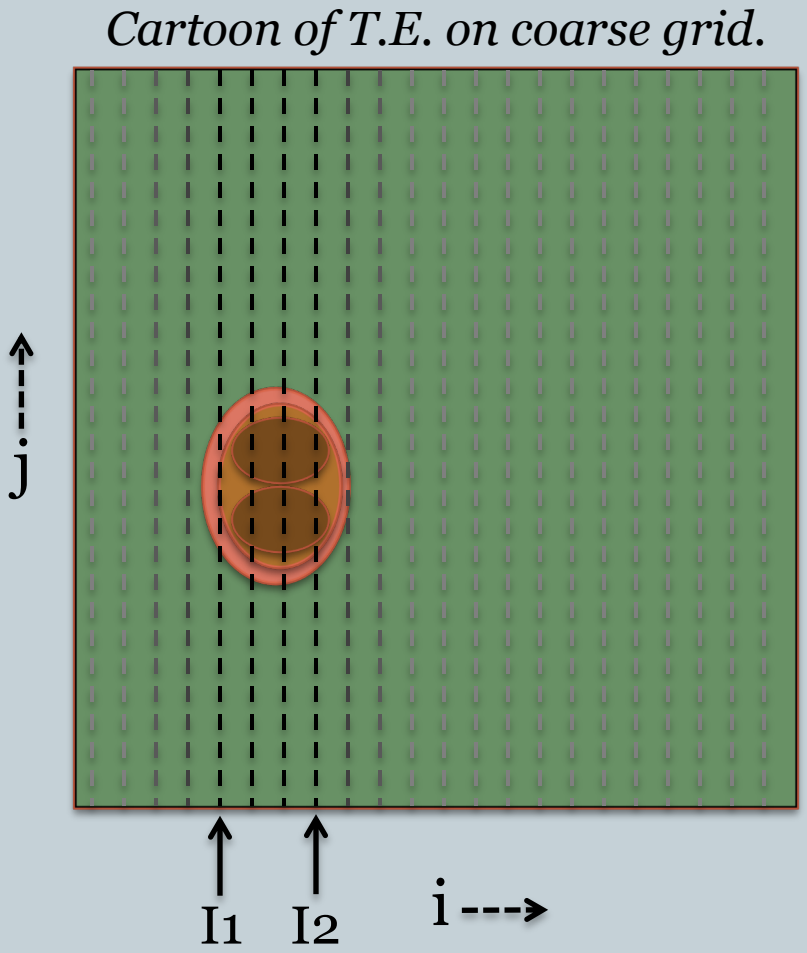


Result *after* I set coarse grid to -1 everywhere, and then did ***feedback.***

Determining the nest location (1)

- Compute the 2-D truncation error *(T.E.)*
  - Done *consistent with 1-D directional splitting*; estimates T.E. from the max of the absolute value in the X- and Y-directions
- Locating the nest *(position on coarse grid)*
  - Set T.E. *threshold* as 50% of max T.E. over the 2-D T.E. array
  - "Search" inward from left, right, top, bottom edges, noting the I,J bounds at which you first find T.E. ≥ the *threshold* value.
  - Nest *center* = integer, truncated value: (I1+I2)/2, (J1+J2)/2
  - Nest *size* = (NX-1)/refinement_ratio
  - NestX1 = Icenter-nestsize/2; NestX2 = NestX1 + nestsize
  - NestY1 = Jcenter-nestsize/2; NestY2 = NestY1 + nestsize
  - Check if nest runs off coarse grid edge; fix if so.

Determining the nest location (2)



- Example of finding left, right edges of T.E. region
  - For each *i* column, left to right … check TE(i,j) for all j rows – determine max value
  - Find first and last column (i) for which max ≥ threshold.
- Do same for top/bottom.
- Average I1,I2,J1,J2 for nest *center*. Nest *edges* depend on nest *size*.

Skamarock & Klemp

ADAPTIVE MESH REFINEMENT PAPER

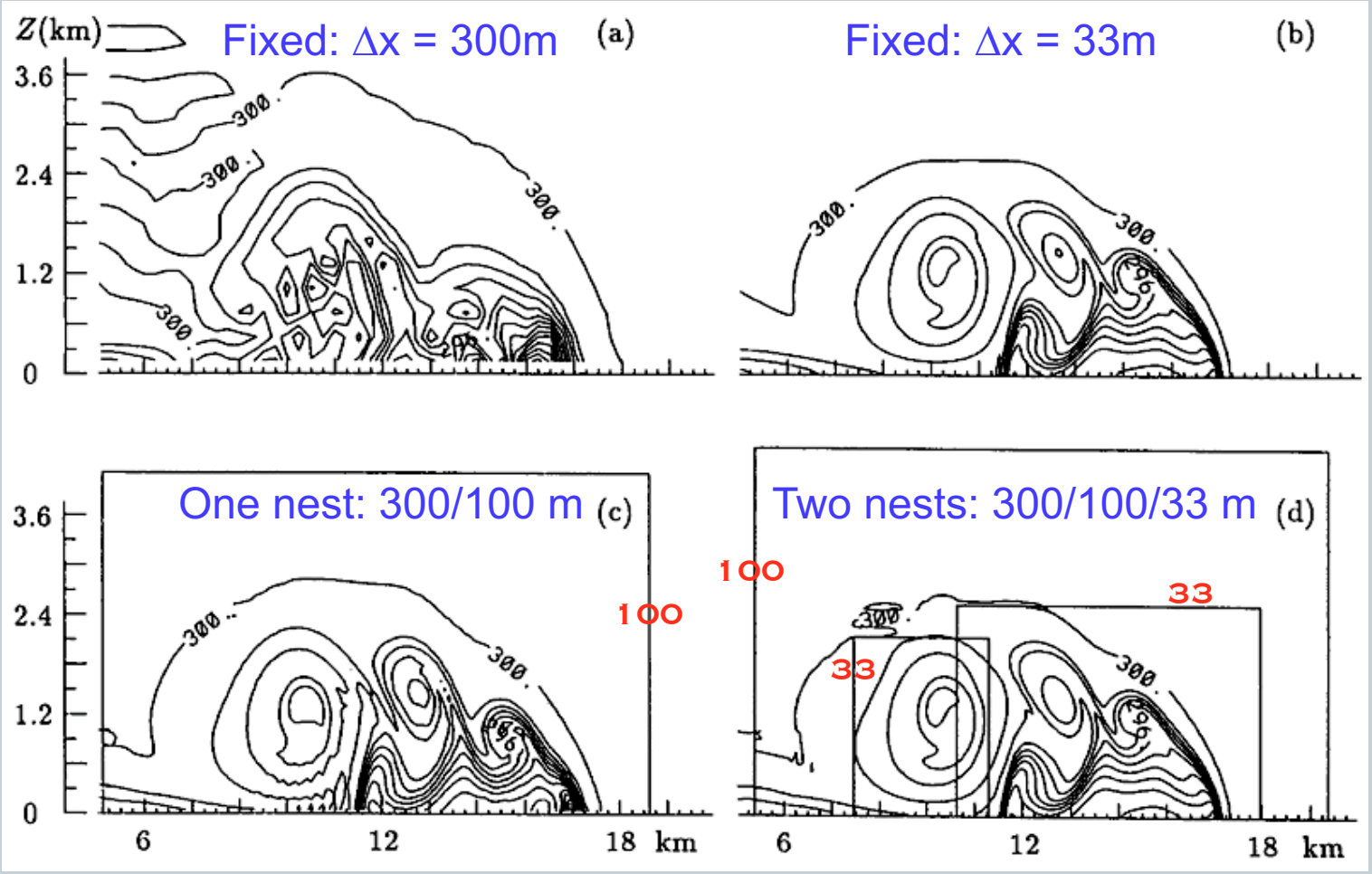
Nesting strategy

- Identification
  - grid points exceeding some threshold, e.g. truncation error
- Clustering
  - fit to enclose points
  - general AMR allows overlapping grids, arbitrary orientation

- Nest: Initial conditions
  - Interpolated from coarse grid, or existing nests
- Nest: Bound. conditions
  - Time dependent
    - from coarse grid, using current and 'next' step.
  - Spatial dependence
    - interpolated from coarse grid. in our case, nest BC overlap with coarse points

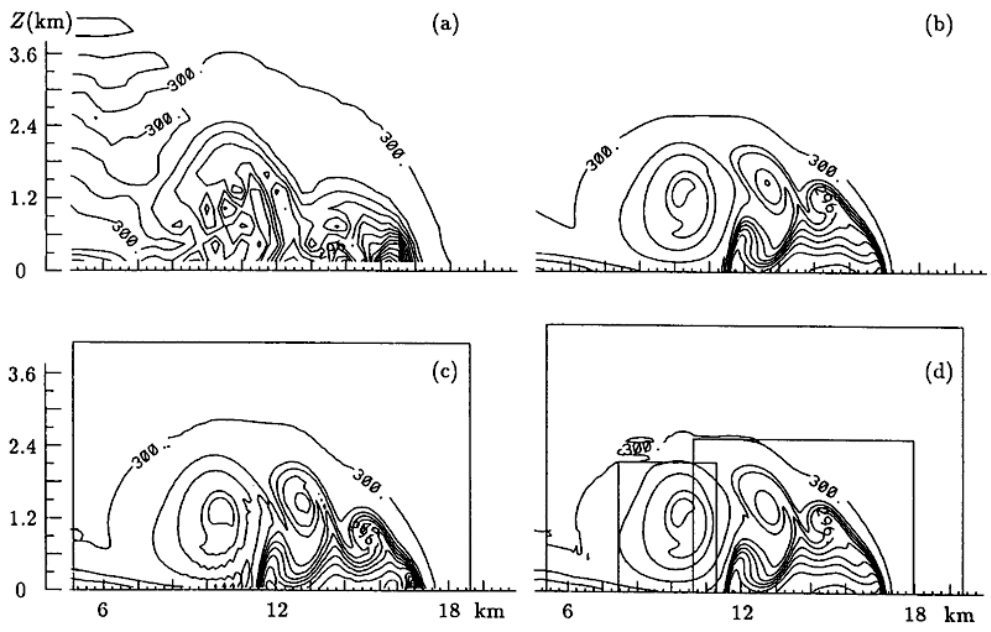
- Feedback: nest to coarse grid
  - Average nest interior to coarse points

Results: 2D outflow problem



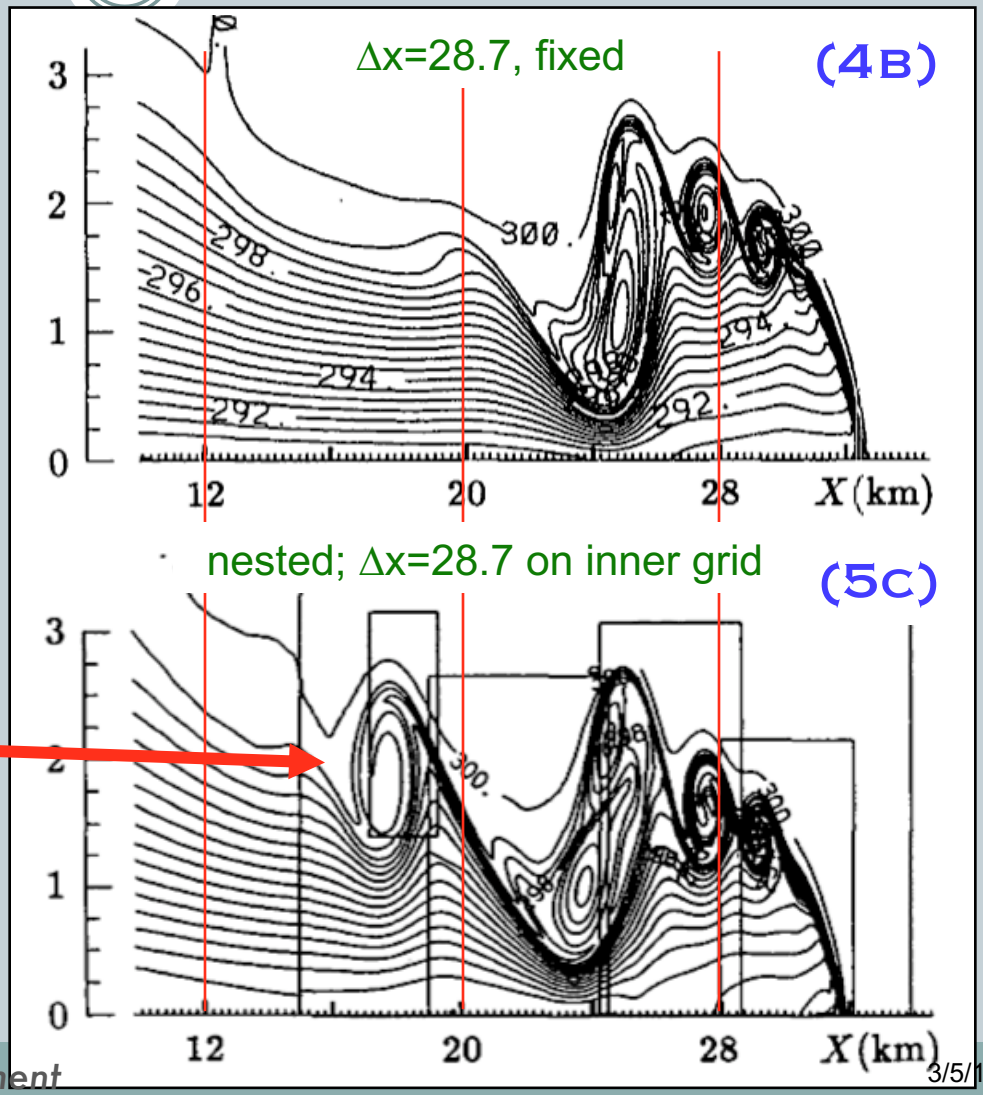
How much refinement?

- *Their AMR is a <u>local</u>* refinement method!
  - *efficiency reduced as larger area refined (AMR has a cost)*
  - *break-even: 50-60% of coarse grid refined (= max to refine)*
  - *if you need to refine a bigger area: just refine entire grid*



BC noise, spurious waves

- Fixed resolution run shown (top)
- Nested solution shown (bottom)
- Inner nested grid $\Delta x = 28.7m$
- This is not physical !!!



Parameterization, convergence

- Parameterization of physical processes
  - this refers to treatment of a sub-grid process using information on larger scales
  - common practice
  - may affect convergence!

- Truncation error vs. grid size
  - did not decrease as expected with improved resolution – error sometimes increased!
    - *as a consequence of grid-scale-dependent parameterizations*
  - smaller Δx => smaller-scale features!!