Session 15: Feature- and Adjoint-Based Mesh Adaptation

Mike Park
Computational AeroSciences Branch
Learning Goals

• A little background on adaptation
• Manual step-by-step output adaptation cycle of a turbulent flat plate
• Describe the Makefiles and Ruby scripts that automate this process
  • Turbulent flat plate output-based adaptation
  • Inviscid cut cell sonic boom output-based adaptation
  • Laminar cylinder feature-based adaptation
Local Error and Output Adaptation

**Local error based**
- Feature based adaptation
- Flow solver/physics agnostic
- Not as robust
- Requires more manual interaction

**Output error based**
- Requires adjoint solution
- More robust
  - Transport of errors
- Fewer user controlled parameters
Available Adaptation Modes

• Inviscid cut-cell output based adaptation
• Viscous body-fitted output based adaptation with frozen boundary layers
• Viscous body-fitted local error based adaptation with frozen boundary layers
• Others at various stages of development (contact us)
Output-Based Adaptation

- Mathematically rigorous approach involving the adjoint solution that reduces estimated error in an engineering output
- Uniformly reducing discretization error is not ideal from an engineering standpoint - some errors are more important to outputs
Shock Propagation Example

- Adaptation is targeted to improve off-body pressure integral output for diamond airfoil
Adaptation Process

Flow Solver

Adjoint Solver

Adaptation Metric

Grid Adaptation

Flow Solver

Adaptation Metric

Grid Adaptation
Adaptation Mechanics

- Parallel node insertion, node movement, element collapse, and element swap
  - Very general approach to iteratively drive mesh to satisfy an anisotropic metric $M$

\[
\begin{align*}
\times M &= \begin{array}{c}
\text{Ellipse} \\
\text{Triangle}
\end{array} \\
\times M &= \begin{array}{c}
\text{Circle} \\
\text{Right Triangle}
\end{array}
\end{align*}
\]
Adaptation Metric

- Output-based size specification scales the stretching and orientation of the Mach Hessian grid metric (Venditti and Darmofal)

\[ M = \left| \frac{\partial^2 \text{Mach}}{\partial x^2} \right| = X \begin{bmatrix} \left( \frac{1}{h_1} \right)^2 \\ \left( \frac{1}{h_2} \right)^2 \\ \left( \frac{1}{h_3} \right)^2 \end{bmatrix} X^T \]
Adaptation Metric

- Output-based size specification scales the stretching and orientation of the Mach Hessian grid metric (Venditti and Darmofal)

\[
e_{K} = \frac{|(\hat{\lambda} - \bar{\lambda}) R(\hat{u})| + |(\hat{u} - \bar{u}) R_{\lambda}(\hat{\lambda})|}{2}
\]

\[
\frac{h_{\text{request}}}{h_{\text{current}}} = \left( \frac{e_{\text{tol}}}{\sum e_{K} \frac{e_{\text{tol}}}{N e_{K}}} \right)^{\omega}
\]
Turbulent Flat Plate

- Change directory to the example
  - `cd`
  - `cd Adaptation_Demos`
  - `cd flat-plate`
Turbulent Flat Plate Contents

- rubber.data is for the adjoint cost function
  - ‘cd’ is specified as the output
- Flow/ is where the flow solve will be performed
- Adjoint/ is where the adjoint solve and adaptation will be performed
Turbulent Flat Plate Flow Solve

• cd Flow
• qsub flow-solver.pbs
• tail –F flow-solver.output
• box01.fgrid and box01.mapbc is the grid and boundary conditions
• box01_flow_fun3d.nml is the input file
Turbulent Flat Plate Flow Solve

• In the flow-solver.pbs script the command line arguments are specified.
  • --linear_projection wraps the standard linear solver
  • Kick out tolerance
  • Stabilizes unstable linear solves
  • --animation_freq -1 and --sampling_freq -1 produce tecplot output
• The box01_flow_fun3d.nml is copied to fun3d.nml
Turbulent Flat Plate Grid
Turbulent Flat Plate History
Turbulent Flat Plate Solution
Turbulent Flat Plate Adjoint Solve

- cd ../Adjoint
- qsub adjoint-solver.pbs
- tail –F adjoint-solver.output
- In the adjoint-solver.pbs script the command line arguments are specified.
  - --linear_projection wraps the standard linear solver
    - Kick out tolerance
    - Stabilizes unstable linear solves
  - --outer_loop_krylov stabilizes and accelerates adjoint iterative convergence (linear problem)
- The box01_adjoint_fun3d.nml is copied to fun3d.nml (in the ../Flow directory)
Turbulent Flat Plate History
Hybrid Adaptation

- Adapted Grid
- Frozen Grid
- Original Grid
Turbulent Flat Plate Adaptation

- qsub adaptation.pbs
- tail –F adaptation.output
- In the adaptation.pbs script the command line arguments are specified.
  - --adapt activates adaptation with refine
  - --embedrad embeds that grid and forms the output-based adaptation metric
  - --adaptation_project box02 is the name of the new project that is produced by adaptation
  - --adapt_freezebl 0.001 thickness of the frozen boundary layer grid
- The faux_input file specifies the geometry of adaptation planar faces
- The box01_adjoint_fun3d.nml is copied to fun3d.nml (in the ../Flow directory)
Adapted Turbulent Flat Plate Flow Solve

- cd ../Flow
- qsub adapted-restart.pbs
- tail –F adapted-restart.pbs
- In the adapted-restart.pbs script the command line arguments are specified.
  - --linear_projection wraps the standard linear solver
    - Kick out tolerance
    - Stabilizes unstable linear solves
  - --animation_freq -1 and –sampling_freq -1 produce tecplot output
  - The box02_flow_fun3d.nml is copied to fun3d.nml
Turbulent Flat Plate History
Turbulent Flat Plate Solution
refine and knife

- refine
  - Grid adaptation library
  - Called by FUN3D through an API
- knife
  - Cut cell library
  - Utilities for extracting and visualizing cut surface from volume grids
refine and knife

- Linked into the version of FUN3D we will use today
- Require separate user agreements
- Use the same autotoool process to build
  - ./configure
  - make
  - make install
- See the README and INSTALL files in each package
Git the Tutorial

• Obtain a copy of the examples with git
  • cd
  • git clone ../funshop-files/grid-adaptation-tutorial.git
Start the flat plate case running

• Go to the testcase
  • cd
  • cd grid-adaptation-tutorial
  • cd flat-plate-frozen-bl
  • cd case
• Start it running
  • make test
• Is it running?
  • make view
  • [press ctrl-c to exit]
Input files

- **fun3d.nml**
  - Standard fun3d input file
- **rubber.data**
  - Standard adjoint input file
  - Quiz… what is the adjoint output function?
    - Hint: line 76
Grids

- Flat plate grid (body-fitted)
  - Created by box01.f90 and box01.sh
  - See the 'grid' directory
- faux_input (background grid geometry)
  - Number of background grid faces
  - Face id, type, position
- Frozen to a height listed in “case_specifics”
  - rad_cl “—adapt_freezebl 0.001”
Hybrid Adaptation

- Adapted Grid
- Frozen Grid
- Original Grid
pbs_run

- Ruby wrapper to make pbs “act like” mpirun/mpiexec
  - See grid-adaptation-tutorial/pbs_job
  - Specified in casepecifics
    - mpirun_command './././pbs_job/bin/pbs_run'
  - Generates a pbs job with xMoDaHrMnS.pbs
    - Month, Day, Hour, Min, tens of Sec.
  - Output is in xMoDaHrMnS, xMoDaHrMnS.pbsout, xMoDaHrMnS.pbserr
Makefile

- Provides targets
  - `make test` starts the adaptation
  - `make shutdown` abruptly stops adaptation (may require qdel)
  - `make view` watch the status of the case
  - `make hist` plot the convergence history
  - `make nodes` grid size
  - `make remain` remaining error estimate in output
Status

• **output**
  • Lists the commands as they are executed
• **Flow/flow\_out**
  • flow solver
• **Adjoint/dual\_out**
  • Adjoint solve
• **Adjoint/rad\_out**
  • Output-based adaptation

(when using pbs\_run the *_out files will list the pbs queue status and a tail command to see screen output while running)
Iterative Convergence

- **make hist**
  - Invokes a Ruby script that converts */*_hist.tec to */*_hist.jpg
  - Uses gnuplot under the hood
  - Lists all 5 (or 6 when turbulent)
Executing by hand

- See ‘output’ for command with arguments
- For namelist, see
  - Flow/rootXX_flow_fun3d.nml
  - Flow/rootXX_dual_fun3d.nml
  - Flow/rootXX_rad_fun3d.nml
Start the Cut Cell Case Running

• Go to the testcase
  • cd
  • cd grid-adaptation-tutorial
  • cd diamond-airfoil
  • cd cutcell
• Start it running
  • make test
• Is it running?
  • make view
  • [press ctrl-c to exit]
Cut-Cell Method

- Background volume grid
- Surface grid of geometry
  - Boolean subtracted from median dual background grid
Grids

- Surface grid
  - Describes the cut surface
  - Can come from many sources
  - VGrid in this case
  - See the 'surface' directory and README

- Background grid
  - Describes the computational domain that the cut surface will intersect
  - Created by domain01.f90 and domain01.sh
  - See the 'background' directory

- faux_input (background grid geometry)
  - Number of background grid faces
  - Face id, type, position
Input files

- **fun3d.nml**
  - Standard fun3d input file

- **rubber.data**
  - Standard adjoint input file (only care about line 76)

- **sonic_boom.input**
  - Header is self-describing
case_specifs

- Ruby “domain specific language” input deck
  - Key-value pairs
  - `code_cl` are the command line options
  - `code_nl['key']='value'` modifies fun3d.nml for this code
  - `root_project 'root'` expects the first grid to be root01
  - `iterations first..last` run from first to last iteration
    - Each iteration uses root01, root02, etc
  - Read by the grid-adaption-tutorial/fun3d.rb script
Cut cell (knife) input files

- **project01.knife**
  - Location of the cut surface grid.[tri,fgrid]
  - Do the surface normals point inward or outward
  - Optionally translate, flip, rotate surface
  - List faces to cut with, or omit to include all faces
- **project01.cutbc**
  - Fun3d boundary condition for cut surfaces
  - Overload face with boundary condition
Volume Slice

- Described in fun3d.nml &sampling_parameters namelist
- Produced with –sampling_freq -1
- Run tecplot on Flow/domainXX_tec_sampling_geom1.dat
  - Where XX is 01, 02, 03, … the series of adapted grids
  - The grid is the arbitrary intersection of a plane and the volume tetrahedra
  - The cut cells are not included
Cut surface and cut cells

- Run tecplot on `Flow/domainXX_cut_surf.t`
  - Primitive variables interpolated to cut surface grid
- Run tecplot on `Flow/domainXX_*_cut.t`
  - Primitive variables on boundaries and cut cells
  - One file per processor
Pressure signature

- Run tecplot on `Adjoint/domainXX_pressure_signature.tec`
  - Pressure at the `sonic_boom.input` specified locations
Start the feature-based supersonic cylinder case running

- Go to the testcase
  - cd
  - cd grid-adaptation-tutorial
  - cd supersonic-cylinder
  - cd case
- Start it running
  - make test
- Is it running?
  - make view
  - [press ctrl-c to exit]
Grids

• Body-fitted grid in cylindrical coordinates
  • Created by box01.f90 and box01.sh
  • See the ‘grid’ directory
• faux_input (background grid geometry)
  • Number of background grid faces
  • Face id, type, position
    – Cylinder faces also have radius, center, and normal
• Frozen to a height listed in “case_specifics”
  • rad_cl “—adapt_freezebl 0.05”
• The outer boundary is frozen with “box01.freeze”
  • Each face id is listed one per line
case_specifics

• Ruby “domain specific language” input deck
  • Overloaded the iterate method to do feature-based adaptation

```ruby
def iterate
  iterations.each do |iter|
    iteration iter
    setup if ( 1 == iteration )
    flo
    adapt
  end
end
```

• Adapt code gets `rad_cl` and `rad_nl`
  - `--adapt_coarsen 0.0` do not coarsen grid
  - `--output_error 0.1` target density edge jumps of 0.1