FUN3D
Fully Unstructured Navier-Stokes

Current Release: **12.4-70371**
Site Last Updated: **Wed Jun 04 16:41:01 -0400 2014**

Previous (C6: Analysis) | Up | Next (C8: Design)

7. Adaptation and Error Estimation

7.1. Capabilities

Fun3D has three basic types of adaptation available, listed here in order of complexity.

- Mesh movement via spring analogy
- General adaptation using flow solution gradients for adaptation metric
- General adaptation using an adjoint-based error estimation

All-tetrahedral meshes are required for each of the adaptation strategies. The mesh movement scheme is easy to use and computationally inexpensive. It is primarily used to align the mesh with strong shocks in the flow field. The general adaptation schemes are more complex to use, and much more computationally intensive. They require an additional library, `refine`, which can be requested via the same channels as FUN3D (send an email). The adaptation mechanics for the general adaptation provide for node insertion and removal, node movement, and improvement of element quality via edge swapping. The criteria for adaptation is provided by either flow solution gradients (typically called feature-based) or by the error estimate developed by solving the adjoint equations in addition to the flow equations.

These methodologies are available for evaluation by users, but are still very much in development. The documentation for adaptation is incomplete at best. Please share your experiences with us so we can improve the adaptation further.

7.2. Mesh Movement via Spring Analogy

The mesh movement scheme within FUN3D is simple to use, and inexpensive computationally. We have had good experience with it in the generic gas path, and very little experience within the standard compressible perfect gas path. The methodology can be used for both viscous and inviscid flows.

To invoke, use the command line options:

```
--adapt_spring
```

```
--adapt_freq <num of itns between adapts>
```

Typically, a flow solution is run without the mesh movement for enough iterations to properly set up the flow features. The flow solution is then restarted with the mesh movement turned on, and the movement run every 50-100 iterations, for 1000-3000 iterations. The length of time to run the spring adaptation will be problem dependent. The algorithm is designed to move the mesh a little bit at a time, and at some point the movement will essentially have converged. The flow solution is then restarted again without the mesh movement, and allowed to fully converge on the final mesh.

7.3. Requirements and Configuring to Use `refine`

Using the grid adaptation features within FUN3D requires the `refine` libraries. At present, this must be requested separately through the same channels as requesting FUN3D. Contact FUN3D Support for more details.

General

There are 3 functions for the adaptation codes:

- Provide metrics on when and where to adapt. For the adjoint-based, this functionality is provided using using the adjoint solver. For the feature-based, this functionality is provided using the `adapt` code
- Adaptation mechanics (how to alter/add/reconnect, etc): This is the `refine` library, used for both adjoint and gradient-based adaptation.
- Constrain points in boundary faces to the surface geometry. This requires some knowledge of the geometry by the refine library, and can be accomplished (listed in decreasing complexity) by linking to a CAD-based geometry engine via `CAPrI` and `GridEx`, linking to a simplified geometry engine that can only handle planar surfaces (`FAUXGeom`), or by freezing the surface triangulation of specified boundary faces.

Setting up your geometry

To adapt a mesh on any surface, you must have some sort of geometry definition. `refine` can handle several types of geometry:

No geometry, where the surface nodes are frozen.

`refine` cannot adapt in the viscous layers, and so boundary faces that have high aspect ratio layers growing off of them must be frozen, along with the mesh for a specified distance away from the surface to maintain grid quality. This is invoked with the `--adapt_freezebl <real>`, where the real argument is a distance away from the no slip surfaces to freeze the mesh. This distance is chosen by the user, usually after probing the mesh to determine the maximum boundary layer height.

Additionally, specific surfaces that do not have a viscous boundary condition can be frozen by listing the surface numbers in a file named `[project].freeze`. For example, a file named `cylinder.freeze` with contents of

```
5
7
```

will freeze points on patches 5 and 7 for the `cylinder` project.

This is useful for certain outer boundary surfaces, where there is not an analytical definition handled by `FAUXGeom`. An example of its use is freezing the curved outer boundary of our example hypersonic cylinder problem.

FAUXGeom, for planar and cylindrical boundary surfaces

For viscous problems, where the mesh on the complex geometry of the body is frozen, `FAUXGeom` can be used to provide an analytical definition of the boundary surface. This allows adaptation to occur on the outer box or cylinder and symmetry plane surfaces of the mesh, even though the body mesh is frozen. This is particularly important for the symmetry plane. At present, `FAUXGeom` can only handle planar and cylindrical surfaces.

`FAUXGeom` reads the file `faux_input`. Here is an example file:

```
5
 5 xplane -5.0
 3 yplane -1.0
 1 zplane  1.0
16 general_plane 2.0
   0.707 0.707 0.0
11 cylinder 7.5
   0.0 0.0 0.0
   1.0 0.0 0.0
```

The first line is how many faux surfaces are being defined. The subsequent lines have a face number, type of face, and a distance associated with the particular geometry. In this example, the first faux face defined corresponds to surface 5 in the mesh and is a x=-5.0 constant plane. Similarly for x and y planes. Surface 16 is a plane located 2.0 away from the origin and perpendicular to a (0.707,0.707,0.0) normal. Surface 11 is cylindrical with a radius 7.5 measured from the axis defined by a vector from (0,0,0) to (1,0,0).

Real geometry, linked with GridEx

For tangency surfaces, you really do want to adapt the mesh on the surface. For this case, we have to link to the `SDK` libraries within `GridEx`, to satisfy requests from `refine` such as `place_point_on_surface`. So, adapting an inviscid case requires:

- `GridEx` (another form, another piece of software)
- Constructing your geometry such that it can be handled in GridEx. This requires some sort of CAD definition, and the ability to link GridEx to the CAD engine you have (Pro-E, Unigraphics, etc.). There is also a “native” CAD definition that GridEx can handle, which can be generated via `gridTool`
- Generating your mesh within GridEx so that your surface nodes are parameterized on your CAD surfaces.

In short, it is quite a learning curve to get to adapt on tangency surfaces if you are not already using GridEx for mesh generation..

Configuring

The refine package itself uses AutoTools, just like FUN3D versions 10.4 and higher. See the README and INSTALL files in the refine distribution for details.

The FUN3D AutoTools configure script will require the following flags:

- Tell FUN3D where refine is installed

```
--with-refine=/my/path/to/refine/installation
```

- To use FAUXGeom geometry, specify

```
--with-refineFAKEGeom=-lFAUXGeom
```

- To use the full-blown CAPRI-GridEx CAD geometry, contact FUN3D Support. It is *not* trivial to set up.

Pre-processing meshes that will be adapted.

- Do NOT lump boundaries.
- When using `party`, use the command line option `--no_renum` to ensure consistent numbering with the original mesh (required for CAPRI-GridEx only)

Command-line options for adaptation, Pre Version 11.4

The following command-line options are common to all applications (adjoint and feature adaptation) that use the `refine` library.

`--adapt_project <name>` : name for adapted grid, default is to append a `_R` to current project, `current_project_name_R`.

`--adapt_freezebl` : This is a distance above the surface, in your grid units. You will need to look at the mesh (tecplot) to pick an appropriate value. There will be no adaptation between the surface and this value above the surface, and this is used to keep the boundary layer mesh looking like a boundary layer mesh. Eventually, we will have adaptation that will be able to add boundary layer points while keeping the desired stretching, but `refine` isn't there yet.

`--adapt_smooth_surface` : This option should almost always be used. Without it, no movement of nodes will occur on boundaries (including planar boundaries), so the resulting mesh with inserted points will be not smooth.

`--adapt_cycles 3` : number of adaptation cycles to run. 2 or 3 is a good place to start, default is 6.

The remaining options are problem/grid size dependent, so some trial & error is usually required

`--output_error <real>` : This is target maximum value for the adaptation parameter, either adjoint- or feature-based. The smaller the number, the more new mesh you get. A negative value is permitted for adjoint-based adaptation where it is a relative error reduction instead of an absolute error tolerance.

`--adapt_maxedge 1.0` : This is the maximum edge size you want to allow in your mesh. It should be consistent with the size of the elements in your outer boundary, or you will see lots of refinement out there.

`--adapt_maxratio 10` : maximum aspect ratio allowed in adapted cells. Mike says you can go up to 100, Karen usually goes up to 30. It is more of an issue as to what you want the flow solver to try to deal with.

`--adapt_coarsen 2.0` : limits coarsening allowed by feature-based adaptation.

Namelist options for adaptation, Version 11.4 and higher

FUN3D has migrated many of the adapt command line options to namelists. For convenience, the defaults are listed in the sample namelists below.

```
&adapt_mechanics
adapt_project = ''
adapt_freezebl = -1.0
adapt_cycles = 2
/
```

`adapt_project` Name for adapted grid, default is to append a `_R` to current project

`adapt_freezebl` Distance of body, in grid units, to freeze nodes

`adapt_project` Number of adaptation cycles to run

```
&adapt_metric_construction
adapt_hessian_key = 'mach'
adapt_hessian_method = 'lsq'
adapt_max_anisotropy = 100.0
adapt_max_edge_growth = 2.0
adapt_output_tolerance = -0.5
adapt_error_estimation = 'embed'
adapt_exponent = 0.2
adapt_feature_scalar_key = 'density'
adapt_feature_scalar_form = 'delta'
/
```

`adapt_hessian_key` Specifies variable used to determine the anisotropic Hessian. Options: `entropy`, `mach`, `pressure`, `temp` (temperature), and `density`

`adapt_hessian_method` Specifies how the Hessian is calculated, currently ‘lsq’ is the only option

`adapt_max_anisotropy` Maximum allowable limit on cell aspect ratio.

`adapt_max_edge_growth` [undocumented]

`adapt_output_tolerance` Error tolerance. Feature based adaptation requires a positive number, while output based adaptation can be negative to indicate a relative error reduction

`adapt_error_estimation` Indicates whether to use single or embedded grid error estimates in output based path

`adapt_exponent` [undocumented]

`adapt_feature_scalar_key` Flow feature on which to adapt. Options `entropy`, `mach`, `pressure`, `temp` (temperature), and `density` `adapt_feature_scalar_form` Method to calculate refinement indicator for feature based path. `delta`: max delta across edges at each node, default

`delta-l`: max (delta * edgeLength^exp), to provide some scaling with mesh size

`ratio`: max ratio of flow quantity across an edge, typically used for a pressure ratio.

7.4. Adjoint-Based Adaptation

To appear… for now, please see the publications and applications areas.

7.5. Gradient/Feature-Based Adaptation

Feature-based Adaptation Options, Pre Version 11.4

The following options apply only to the feature-based adaptation, using `adapt`.

`--adapt_feature_type <0>` : This option controls the general formulation of the adaptation metric. The default formulation constructs the adaptation metric by combining a scalar quantity for the adaptation strength with the Hessian of another flow-based quantity to provide stretching of the resulting mesh. The formulation of the scalar and Hessian quantities is controlled by additional command line arguments.

Additional, more complex, formulations of the metric are planned. Currently, only the default formulation, `<0>`, is allowed.

`--adapt_feature_scalar_key <density>`: Flow quantity to be used in computing the scalar adaptation magnitude. `density` is the default quantity. Allowed options are:

`entropy`, `mach`, `pressure`, `temp` (temperature), `density`, and `manufact`

`--adapt_feature_scalar_form <delta>`: Method for computing the adaptation scalar from the flow quantity. Available options are:

`delta`: max delta across edges at each node, default

`delta-l`: max (delta * edgeLength^exp), to provide some scaling with mesh size

`ratio`: max ratio of flow quantity across an edge, typically used for a pressure ratio.

`hess_max`: max eigenvalue of Hessian of key_var at each node, used to scale mesh size with 2nd derivative.

`--adapt_feature_length_exp <0.5>`: used with `delta-l` to set how to scale the scalar parameter with edge length. The default is 1/2, to scale by sqrt(edge length).

`--adapt_feature_hess_key <mach>`: Flow quantity to be used in the Hessian (2nd derivative) computation. Allowed options are the same as for `--adapt_feature_scalar_key`. The default for the Hessian is `mach`

`--adapt_felisa` : Use this option if you are adapting from a FELISA flow solution, so that the flow variables are mapped properly.

Using Feature Adaptation with other flow solvers.

`adapt` can theoretically be used with other flow solvers that use tetrahedral meshes. Currently, Karen uses `adapt` regularly with the `FELISA` inviscid solver. Interfaces for other codes could be developed. Contact FUN3D Support for more information.

User test case

Please see the tutorial section of the manual, **Adaptation Tutorial – Testing Solution Gradient-based Adaptation**

The first case (no_flow) will exercise the `adapt_mpi` code without requiring a flow solution. This case will just show that the adaptation code is compiled and running correctly. The second case will run the solver for a few iterations, adapt to that flow solution, and then run the solver for more iterations. Again, this case demonstrates the process for running `adapt_mpi`, but does not itself produce complete, converged results.

Publications

See bibb_capsules for basic information

7.6. Error Estimation

To appear… for now, please see the publications and applications areas.

Previous (C6: Analysis) | Up | Next (C8: Design)

Today's NASA Official: Bill Jones, a member of The FUN3D Development