FUN3D v12.4 Training

Session 11: Dynamic-Grid Simulations

Bob Biedron

Session Scope

- What this will cover
  - How to set up and run time-accurate simulations on dynamic meshes
    - Nondimensionalization
    - Choosing the time step
    - Body / Mesh motion options
    - Input / Output
- What not be covered
  - Specifics for overset and aeroelastic: covered in follow-on sessions
- What should you already be familiar with
  - Basic steady-state solver operation and control
  - Basic flow visualization

Introduction

- Background
  - Many of problems of interest involve involve moving or deforming geometries
  - Governing equations written in Arbitrary Lagrangian-Eulerian (ALE) form to account for grid speed
  - Nondimensionalization often more involved/confusing/critical
- Compatibility
  - Fully compatible for compressible/incompressible flows; mixed elements; 2D/3D
  - Not compatible with generic gas model
- Status
  - Compressible path with moving grids is exercised routinely; incompressible path much less so
  - 6-DOF option has had very limited testing / usage

Governing Equations

- Arbitrary Lagrangian-Eulerian (ALE) Formulation

$$\frac{\partial(\vec{Q}V)}{\partial t} = -\oint_{\partial V}\left(\overline{\overline{F}} - \vec{q}\vec{W}^T\right)\cdot\vec{n}dS - \oint_{\partial V}\overline{\overline{F_v}}\cdot\vec{n}dS = \vec{R} \qquad \vec{Q} = \frac{\oint_V \vec{q}\,dV}{V}$$

$\vec{W}$ = Arbitrary control surface velocity; Lagrangian if $\vec{W} = (u,v,w)^T$ (moves with fluid); Eulerian if $\vec{W} = 0$ (fixed in space)

- Discretize using Nth order backward differences in time, linearize $\vec{R}$ about time level n+1, and introduce a pseudo-time term:

$$\left[\left(\frac{V^{n+1}}{\Delta\tau} + \frac{V^{n+1}\phi_{n+1}}{\Delta t}\right)\overline{\overline{I}} - \frac{\partial\vec{R}^{n+1,m}}{\partial\vec{Q}}\right]\Delta\vec{Q}^{n+1,m} = \vec{R}^{n+1,m} - \frac{V^{n+1}\phi_{n+1}}{\Delta t}\left(\vec{Q}^{n+1,m} - \vec{Q}^{n}\right) - \ldots + \vec{R}^{n+1}_{GCL}$$

$$= \underline{\overline{\vec{R}^{n+1,m}} + O(\Delta t^N)}$$

- Physical time-level t^n ; Pseudo-time level τ^m
- Need to drive ***subiteration residual*** $\overline{\vec{R}^{n+1,m}} \to 0$ using pseudo-time subiterations at each time step – much more later – otherwise you have more error than the expected $O(\Delta t^N)$ truncation error

KEY POINT

Mesh / Body Motion (1/2)

- Motion is triggered either by setting **`moving_grid = .true.`** in **`&global`** (**`fun3d.nml`**), or by the command line **`--moving_grid`**
- All dynamic-mesh simulations require some input data via an auxiliary namelist file: **`moving_body.input`**
- A body is defined as a user-specified collection of solid boundaries in grid
- Body motion options:
  - Several built-in functions for rigid-body motion: translation and/or rotation with either constant velocity or periodic displacement
  - Read a series of surface files – body can be either rigid or deforming
  - Read a series of 4x4 transform matrices - rigid body
  - 6 DOF via UAB/Kestrel library "libmo"
    - Requires configuring with **`--with-sixdof=/path/to/6DOF`**
  - Application-specific: mode-shape based aeroelasticity (linear structures); rotorcraft nonlinear beam

Mesh / Body Motion (2/4)

- Chose a mesh-motion option than can accommodate the desired body-motion option
- Mesh motion options:
  - Rigid - maximum 1 body containing all solid surfaces (unless overset)
  - Deforming – allows multiple bodies without overset; can be limited to relatively small displacements before mesh cells collapse
  - Combine rigid and/or deforming with overset for large displacements / multiple bodies
- Rigid mesh motion performed by application of 4x4 transform matrix to all points in the mesh - fast; positivity of cell volumes guaranteed to be maintained
  - Complex transforms can be built up from simple ones: matrix multiply
  - Allows parent-child motion (child follows parent but can have its own motion on top of that)

Mesh / Body Motion (3/4)

- Mesh deformation handled via solution of a linear elasticity PDE:

$$\nabla \cdot [\mu(\nabla u + \nabla u^T) + \lambda(\nabla \cdot u)I] = f = 0$$

$$\lambda = \frac{E\upsilon}{(1+\upsilon)(1-2\upsilon)} \qquad \mu = \frac{E}{2(1+\upsilon)}$$

  - υ fixed; E is selectable as:
    - 1 / slen `--elasticity 1` (default)
    - 1 / volume `--elasticity 2` (rarely used anymore)
    - 1 / slen**2 `--elasticity 5` (last ditch for difficult problems)
- Elasticity solved via GMRES method; CPU intensive - can be 30% or more of the flow solve time; check convergence (screen output)
- Fairly robust, but *can* generate negative cell volumes; code stops
- "untangling" step attempted if neg. volumes generated – ***tet meshes only***

Mesh / Body Motion (4/4)

- GMRES solver used for mesh ***deformation*** has default parameter settings which can be adjusted in the namelist **`&elasticity_gmres`** (in the **`fun3d.nml`** file):

```
ileft   nsearch nrestarts       tol
    1      +50        10    1.e-06
```

  - You generally won't have to adjust these values
  - Exception: "structured" grids with very tight wake spacing can be very hard to deform and you may need to set **`tol`** very small, e.g. 1.e-12 (and will need more restarts); usually not an issue with typical grids
  - If negative volumes are generated and not successfully untangled, try reducing **`tol`**, which in turn may require a larger value of **`nrestarts`**

Nondimensionalization of Motion Data (1/2)

- Recall: * indicates a dimensional variable, otherwise nondimensional
- Typical motion data we need to nondimensionalize: translational velocity, translational displacement, angular velocity, and oscillation frequency
- Angular or translational displacements / velocities are input into FUN3D as magnitude and direction
- Displacement input: angular in degrees; translational $\Delta\vec{x} = \Delta\vec{x}^* / (L^*_{ref} / L_{ref})$
- Translational velocity is nondimensionalized just like flow velocity:
  - U^* = translation speed of the vehicle (e.g. ft/s)
  - $U = U^* / a^*_{ref}$ (comp.; this is a Mach No.) $U = U^* / U^*_{ref}$ (incomp)
- Rotation rate:
  - Ω^* = body rotation rate (e.g. rad/s)
  - $\Omega = \Omega^* (L^*_{ref}/L_{ref}) / a^*_{ref}$ (comp) $\Omega = \Omega^* (L^*_{ref}/L_{ref}) / U^*_{ref}$ (incomp)
  - Other variants on specified rotation rate are possible, e.g. rotor tip speed, from which $\Omega^* = U^*_{tip} / R^*$

Nondimensionalization of Motion Data (2/2)

- Oscillation frequency of the physical problem can be specified in different forms
  - $f^* =$ frequency (e.g. Hz)
  - $\omega^* =$ circular frequency (rad/s) (not to be confused with rotation rate)

    $= 2\pi f^*$
  - k = reduced frequency, $k = \frac{1}{2} L^*_{ref}\, \omega^* / U^*_{ref}$ (be careful of exact definition - sometimes a factor of ½ is not used)
- Built-in sinusoidal oscillation in FUN3D is defined as $\sin(2\pi f t + \delta)$ where the nondimensional frequency f and phase lag are user-specfied
- So the corresponding nondimensional frequency for FUN3D is
  - $f = f^* (L^*_{ref} / L_{ref}) / a^*_{ref}$ (comp) $\quad f = f^* (L^*_{ref} / L_{ref}) / U^*_{ref}$ (incomp)
  - $f = \omega^* (L^*_{ref} / L_{ref}) / (2\pi a^*_{ref})$ $\quad f = \omega^* (L^*_{ref} / L_{ref}) / (2\pi U^*_{ref})$
  - $f = k\, M^*_{ref} / (\pi L_{ref})$ $\quad f = k / (\pi L_{ref})$

Overview of `moving_body.input`

- A body is defined as a collection of solid boundaries in the grid
- The specifics of body / mesh motion are set in one or more namelists that are put in a file called **`moving_body.input`** - this file *must* be provided when **`moving_grid`** is triggered (as a CLO or **`&global`** entry)
  - The **`&body_definitions`** namelist defines one or more bodies that move and is always needed in a dynamic-grid simulation
  - The **`&forced_motion`** namelist provides a limited means of defining basic translations and rotations as functions of time
  - The **`&motion_from_file`** namelist defines the motion of a rigid body from a sequence of 4x4 transform matrices
  - The **`&surface_motion_from_file`** namelist defines the motion of a rigid or deforming body from a time sequence of boundary surfaces
  - The **`&observer_motion`** namelist provides a means of generating boundary animation output from a non-stationary reference frame
- **`&body_definitions`** is required with **`moving_grid`** , others optional

Overview of `&body_definitions` Namelist

- Only most-used items shown here – see manual for complete list
- The **`&body_definitions`** namelist defines the bodies that move

```
&body_definitions                  ! below, b=body  i=boundary
  n_moving_bodies          = 0   ! how many bodies in motion
  body_name(b)             = ''  ! must set unique name for each
  parent_name(b)           = ''  ! child inherits motion of parent
  n_defining_boundary(b)= 0   ! how many boundaries define body
  defining_boundary(i,b)= 0   ! list of boundaries defining body
  motion_driver(b)         = 'none' ! mechanism driving body motion
  mesh_movement(b)         = 'static' ! specifies how mesh will
/
```

- ***Caution***: boundary numbers must reflect any lumping applied at run time!
- All variables above except **`n_moving_bodies`** are set for each body
- The a blank string(**`''`**) for **`parent_name`** => inertial frame (default)

Overview of `&body_definitions` (cont.)

- Options for **`motion_driver`** (default: **`'none'`**)
  - **`'forced'`**
    - Built-in forcing functions for rigid-body motion, const. or periodic
  - **`'surface_file'`**
    - File with surface meshes at selected times; interpolates in between
  - **`'motion_file'`**
    - File with 4x4 transforms at selected times; “interpolates” in between
  - **`'6dof'`**
    - relies on calls to “libmo” functions
  - **`'aeroelastic'`**
    - modal aeroelastics
  - Options for **`mesh_movment`** (default: **`'static'`**)
    - **`'rigid'`**
    - **`'deform'`**

Overview of `&forced_motion` Namelist

- Use **`&forced_motion`** namelist to specify a limited set of built-in motions

```
&forced_motion          ! below, index b=body#
 rotate(b)               ! how to rotate this body: 0 don't (default);
                         ! 1 constant rotation rate; 2 sinusoidal in time
 rotation_rate(b)        ! body rotation rate; used only if rotate = 1
 rotation_freq(b)        ! frequency of oscillation; use only if rotate = 2
 rotation_amplitude(b)   ! oscillation amp. (degrees); only if rotate=2
 rotation_vector_x(b)    ! x-comp. of unit vector along rotation axis
 rotation_vector_y(b)    ! y-comp. of unit vector along rotation axis
 rotation_vector_z(b)    ! z-comp. of unit vector along rotation axis
 rotation_origin_x(b)    ! x-coord. of rotation center (to fix axis)
 rotation_origin_y(b)    ! y-coord. of rotation center
 rotation_origin_z(b)    ! z-coord. of rotation center
/
```

- There are analogous inputs for translation (**`translation_rate`**, etc.)
- Note: FUN3D's sinusoidal oscillation function (translation or rotation) has 2π built in, e.g sin(2π **`rotation_freq t`**)

Output Files

- In addition to the usual output files, for forced / 6-DOF motion there are 3 ASCII Tecplot files for each body
  - **`PositionBody_N.dat`** tracks linear (x,y,z) and angular (yaw, pitch, roll) displacement of the "CG" (rotation center)
  - **`VelocityBody_N.dat`** tracks linear (V_x,V_y,V_z) and angular ($\Omega_x, \Omega_y, \Omega_z$) velocity of the "CG" (rotation center)
  - **`AeroForceMomentBody_N.dat`** tracks force components (F_x,F_y, F_z) and moment components (M_x,M_y,M_x)
  - Data in all files are nondimensional by default (e.g. "forces" are actually force coefficients); **`moving_body.input`** file has option to supply dimensional reference values such that *this* data is output in dimensional form - see manual/website for details
  - Forces are by default given in the inertial reference system; **`moving_body.input`** file has option to output forces in the body-fixed system - see manual/website for details

Example 1 - Pitching Airfoil (1/8)

- Example 2 is the one of the well known AGARD pitching airfoil experiments, "Case 1":
  - Re_{c^*} = 4.8 million, M_{inf} = 0.6, chord = c* = 0.1m , chord-in-grid = 1.0
  - Reduced freq. k = 2π f * / (U^*_{inf} / 0.5c*) = 0.0808, (f *= 50.32 Hz)
  - Angle of attack variation (exp): $\alpha = 2.89 + 2.41\sin(2\pi f^* t^*)$ (deg)
- Setting the FUN3D data:
  - **`angle_of_attack = 2.89  rotation_amplitude = 2.41`**
  - Recall f = k M^*_{ref} / π from the 2nd nondimensionalization slide
  - **`rotation_freq`** = f = 0.0808 (0.6) / 3.14… = 0.01543166
  - So in this case we actually didn't have to use any dimensional data since the exp. frequency was given as a reduced (non dim.) frequency

Example 1 - Pitching Airfoil (2/8)

- Setting the FUN3D data (cont):
  - Time step: the motion has gone through one cycle of motion when $t = T$, so that

    $\sin(2\pi\,\mathbf{rotation_freq}\,T) = \sin(2\pi)$

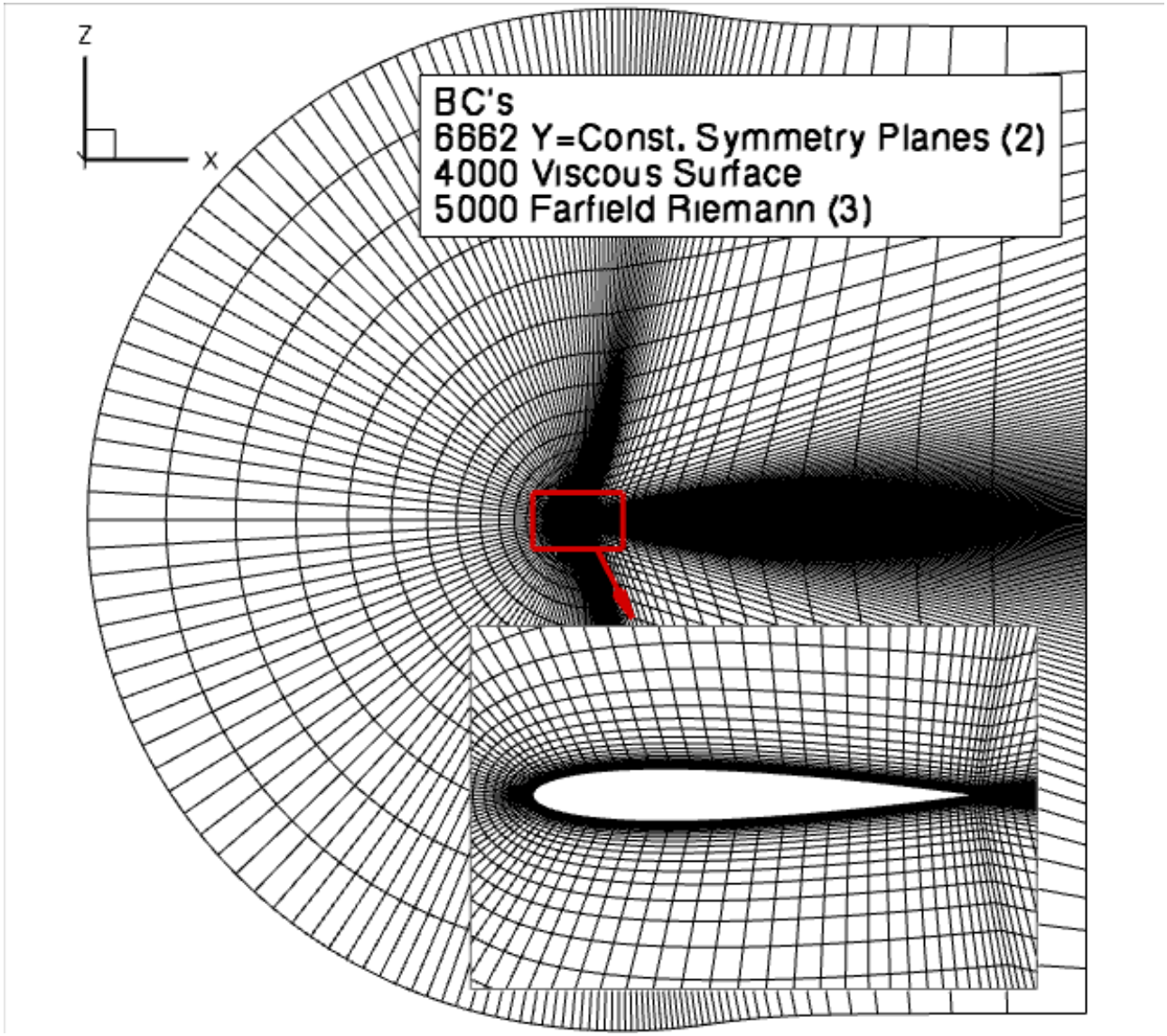
    $T = 1 / \mathbf{rotation_freq}$ (this is our t_{chr})

    for N steps / cycle, $T = N\Delta t$ so

    $\Delta t = T / N = (1 / \mathbf{rotation_freq}) / N$
  - Take 100 steps to resolve this frequency:

    $\Delta t = (1 / 0.01543166) / 100 = 0.64801842$
  - Alternatively, could use $t_{chr} = (1/ f^*)\, a^*_{inf}\, (L_{ref}/L^*_{ref})$, with $f^* = 50.32$ Hz, and assume value for a^*_{inf}

Example 1 - Pitching Airfoil (3/8)



Example 1 - Pitching Airfoil (4/8)

- Relevant **`fun3d.nml`** data

```
&global
   moving_grid = .true.
/
&nonlinear_solver_parameters
   time_accuracy        = "2ndorderOPT" ! Our Workhorse Scheme
   time_step_nondim     = 0.64801842    ! 100 steps/pitch cycle
   subiterations        = 30
   schedule_cfl         = 50.00 50.00   ! constant cfl each step
   schedule_cflturb     = 30.00 30.00
/
```

- Relevant **`moving_grid.input`** data

```
&body_definitions
  n_moving_bodies      =  1,         ! number of bodies
  body_name(1)         = 'airfoil',  ! name must be in quotes
  n_defining_bndry(1)  =  1,         ! one boundary defines the airfoil
  defining_bndry(1,1)  =  5,         ! (boundary, body)
  motion_driver(1)     = 'forced'
  mesh_movement(1)     = 'rigid',
/
```

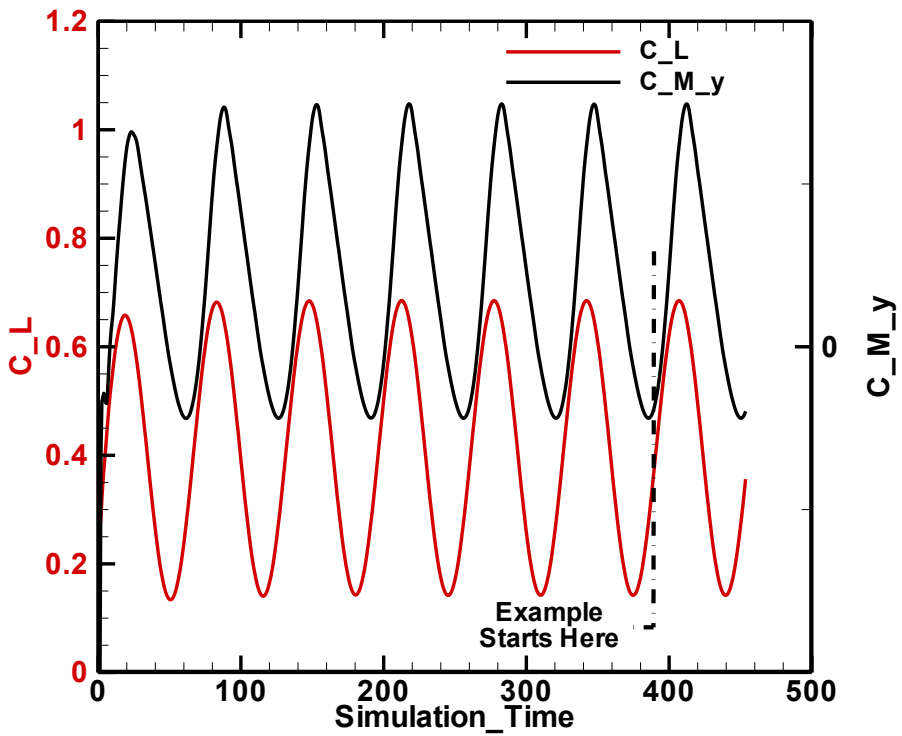
Example 1 - Pitching Airfoil (5/8)

- Relevant **`moving_grid.input`** data (cont)

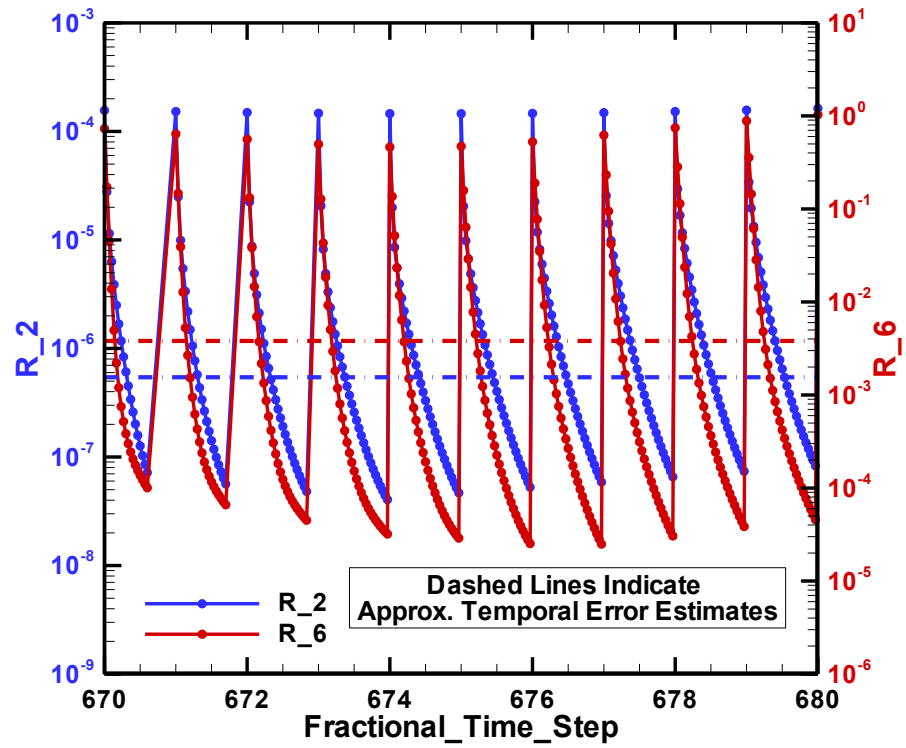
```
&forced_motion
  rotate(1)              = 2,          ! type: sinusoidal
  rotation_freq(1)       = 0.01543166, ! reduced rotation frequency
  rotation_amplitude(1)  = 2.41,       ! pitching amplitude
  rotation_origin_x(1)   = 0.25,       ! x-coordinate of rotation origin
  rotation_origin_y(1)   = 0.0,        ! y-coordinate of rotation origin
  rotation_origin_z(1)   = 0.0,        ! z-coordinate of rotation origin
  rotation_vector_x(1)   = 0.0,        ! unit vector x-component along
   rotation axis
  rotation_vector_y(1)   = 1.0,        ! unit vector y-component along
   rotation axis
  rotation_vector_z(1)   = 0.0,        ! unit vector z-component along
   rotation axis
/
```

Example 1 - Pitching Airfoil (6/8)

Time History
(time_history.lay)

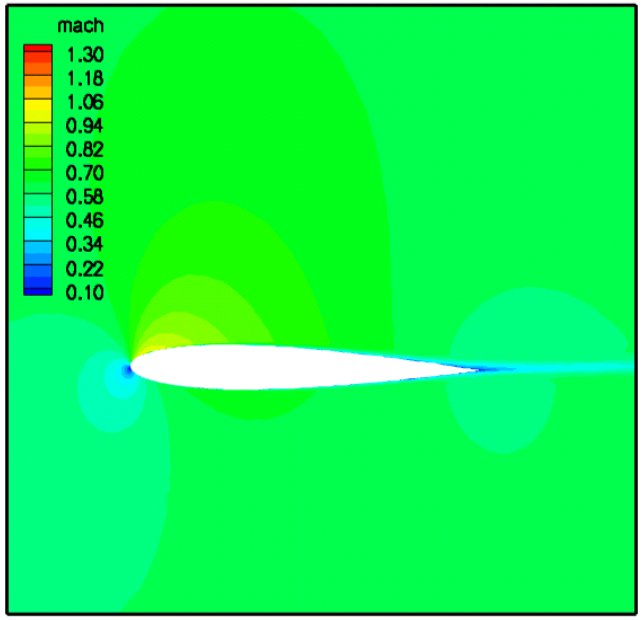


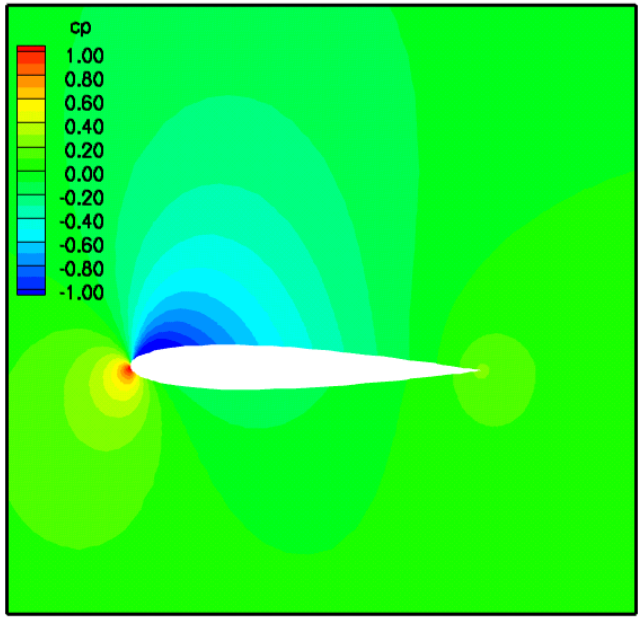
Sample Subiteration Convergence
(where mean flow just misses tolerance)
(subit_history.lay)



Example 1 - Pitching Airfoil (7/8)

Mach Number
(mach_animation.lay)





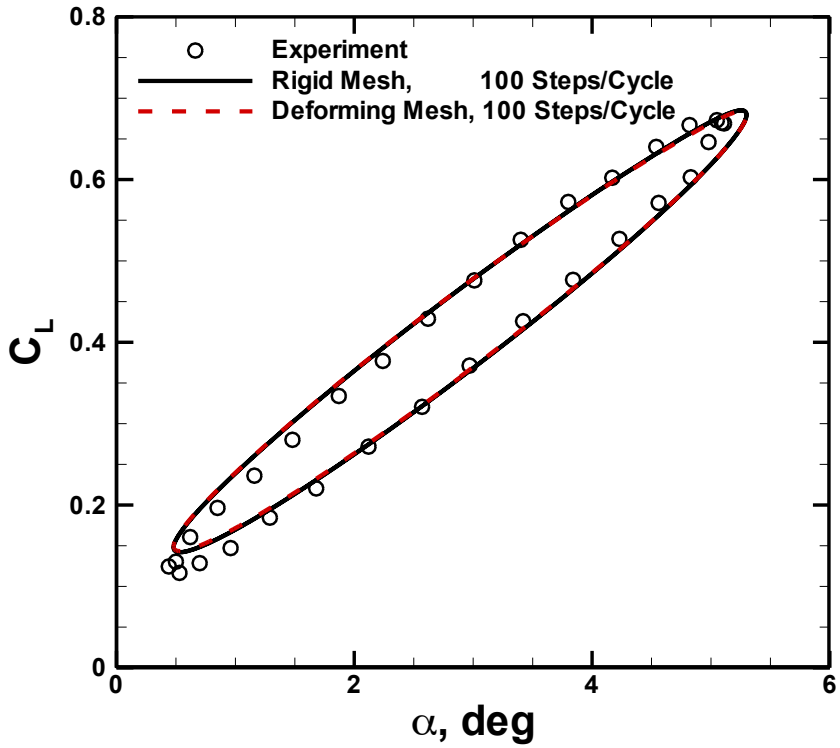
Pressure Coefficient
(cp_animation.lay)

Example 1 - Pitching Airfoil (8/8)

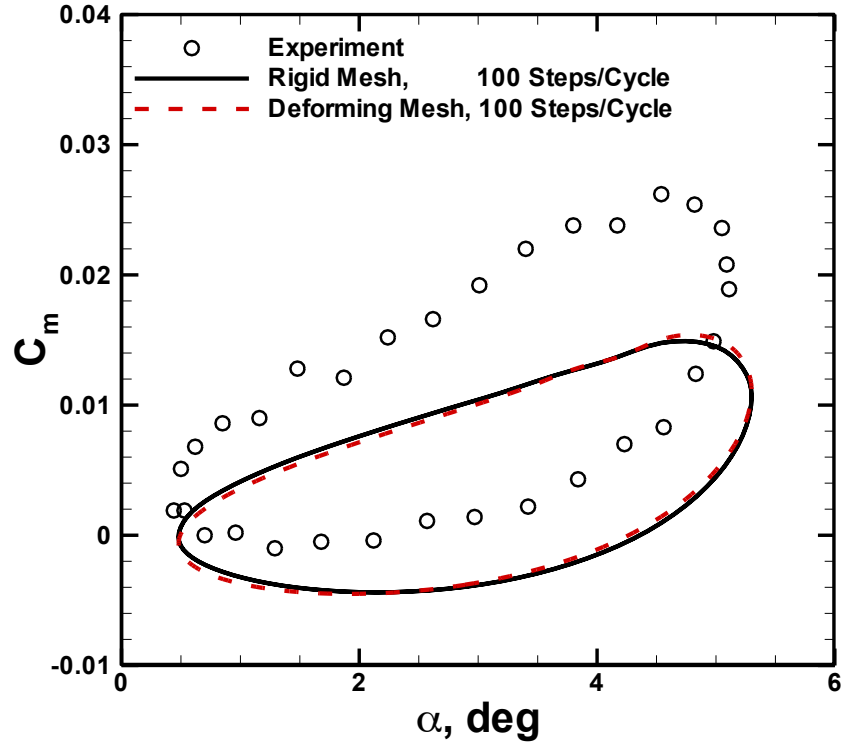
Comparison with Landon, AGARD-R-702, Test Data,1982

Note: comparison typical of other published CFD results

Lift vs. Alpha



Pitching Moment vs. Alpha



Rigid mesh and deforming mesh produce nearly identical results

Troubleshooting Body / Grid Motion

- When first setting up a dynamic mesh problem, suggest using either the following in the **`&global`** namelist
  - **`body_motion_only = .true.`**
  - **`grid_motion_only = .true.`**
- Both options turn off the flow solution for faster processing (memory footprint is the same however)
  - **`body_motion_only`** especially useful for 1st check of a deforming mesh case since the elasticity solver is also bypassed
  - **`grid_motion_only`** performs all mesh motion, including elasticity solution – in a deforming case this can tell you up front if negative volumes will be encountered
  - Caveat: can't really do this for aeroelastic or 6DOF cases since motion and flow solution are coupled
- Use these with some form of animation output: only *solid boundary* output is appropriate for **`body_motion_only`**; with **`grid_motion_only`** can look at any boundary, or use sampling to look at interior planes, etc.

List of Key Input/Output Files

- Beyond basics like **`fun3d.nml`**, etc.:
- Input
  - **`moving_body.input`** (code stops if dynamic mesh and not found)
- Output
  - **`[project]_subhist.dat`**
  - **`PositionBody_N.dat`** (forced motion / 6-DOF only)
  - **`VelocityBody_N.dat`** (forced motion / 6-DOF only)
  - **`AeroForceMomentBody_N.dat`** (forced motion / 6-DOF only)