Session 7: Time-Dependent and Dynamic-Mesh Simulations

Bob Biedron

Learning Goals

- What this will teach you
  - How to set up and run time-accurate simulations on static and dynamic (moving) meshes
    - Subiteration convergence: what to strive for and why
    - Nondimensionalization
    - Choosing the time step
    - Body / Mesh motion options
    - Input / Output
    - Visualization
- What you will not learn
  - Overset, Aeroelastic, or 6-DOF: covered in follow-on sessions
- What should you already know
  - Basic steady-state solver operation and control
  - Basic flow visualization

Setting

- Background
  - Many of problems of interest involve unsteady flows, most of which also involve moving 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 flows; mixed elements; 2D/3D
  - Not compatible with generic gas model
- Status
  - Incompressible flow: should be fully compatible with moving grids, ~~but currently has one or more bugs; working to fix~~ Fixed in V11.2
  - Isolated moving bodies generally do-able
  - Close approach / bodies in contact not so much - no near-term plans to address this

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

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

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

- Physical time-level t^n ; Pseudo-time level τ^m
- Need to drive ***subiteration residual*** $\overline{\vec{R}^{n+1,m}} \rightarrow 0$ 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

Time Advancement - Order of Accuracy

- Currently have several types of backward difference formulae (BDF) that are compatible with both static and moving grids:
  - In order of formal accuracy: BDF1 (**`1storder`**), BDF2 (**`2ndorder`**), $BDF2_{OPT}$ (**`2ndorderOPT`**), BDF3 (**`3rdorder`**), MEBDF4 (**`4thorderMEBDF4`**)
  - Can pretty much ignore all but BDF2 or $BDF2_{OPT}$
    - BDF1 is inaccurate and has little gain in CPU time / step over 2nd order schemes
    - BDF3 not guaranteed to be stable; feeling lucky?
    - MEBDF4 only efficient if working to very high levels of accuracy - *including spatial accuracy* - generally ***not*** where you will be with practical problems
    - $BDF2_{OPT}$ *(recommended)* is a stable blend of BDF2 and BDF3 schemes; formally 2nd order accurate but error is ~1/2 that of BDF2; also allows for a more accurate estimate of the temporal error for the error controller (p.7)

KEY POINT

Time Advancement - Subiterations (1/4)

- Pseudo-time helpful for large time steps (**`pseudo_time_stepping = "on"`**) – benefits convergence - we *always* use it in our applications
- Each time step is a mini steady-state problem in pseudo-time
- Subiterations (**`subiterations`** > 0) are essential
  - Subiteration control in *each time step* operates exactly like iteration control in a steady state case:
    - CFL ramping is available for mean flow and turbulence model – however, be aware that ramping schedule should be < **`subiterations`** or the specified final CFL won't be obtained
    - Ramping and **`first_order_iterations`** start over each time step
    - We *usually* don't ramp CFL or use 1st order in time-dependent cases
- How many subiterations? – that is the ~~$64k~~ $64B question
  - In theory, should drive subiteration residual "to zero" each time step – but you cannot afford to do that!
  - Otherwise have additional errors other than $O(\Delta t^2)$ (2nd order time)

Time Advancement - Subiterations (2/4)

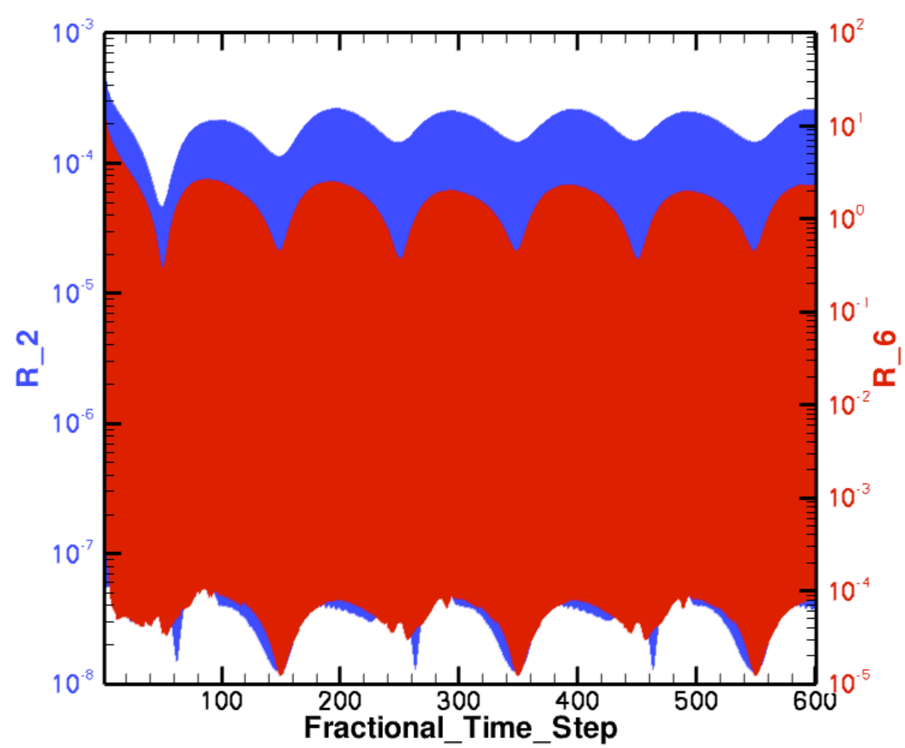
- In a perfect world, the answer is to use the ***temporal error controller***
  - Activated via the CLO **`--temporal_err_control  Real_Value`**
    - **`Real_Value = 0.1`** or **`0.01`** says iterate until the subiteration residual is 1 or 2 orders lower than the (estimated) temporal error
    - Subiterations kick out when this level of convergence is reached OR subiteration counter > **`subiterations`**
    - (empirically) 1 order is about the minimum; 2 orders is better, BUT…
    - Often, if the turbulence subiteration residual doesn't hang / converge slowly – the mean flow subiterations will, and the max subiterations you specify will be used (the world is *not* perfect – need solvers with better / faster convergence)
    - When it kicks in, the temporal error controller is the best approach, and the most efficient; even if it doesn't kick in, it can be informative
- Be wary reaching conclusions about the effect of time-step refinement unless the subiterations are "sufficiently" converged for each size step

Time Advancement - Subiterations (3/4)

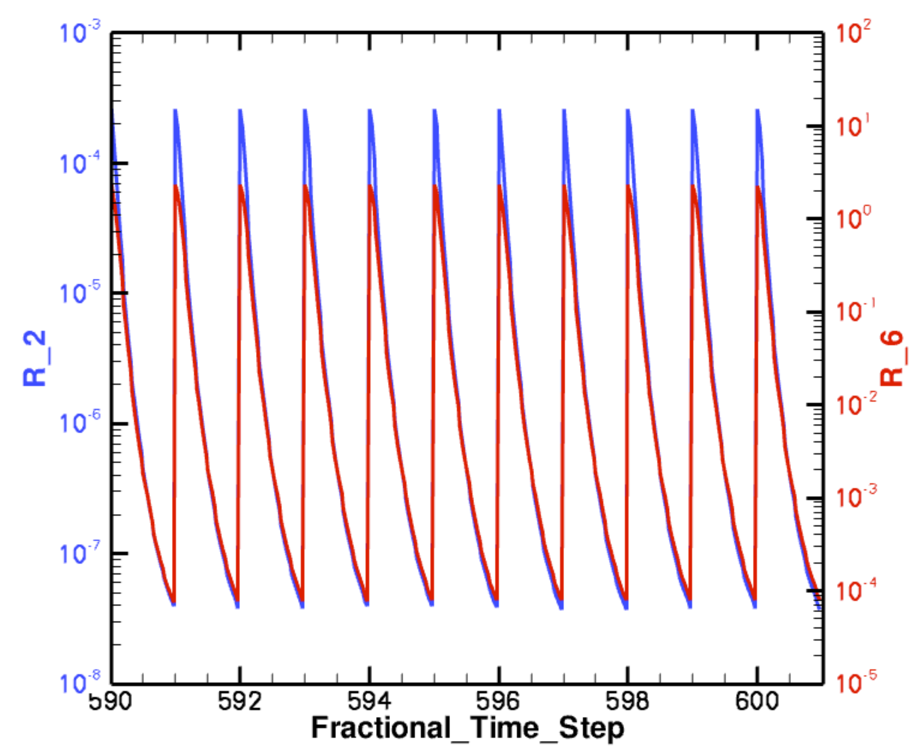
- How to monitor and assess the subiteration convergence:
  - Printed to the screen, so you can "eyeball" it
  - With temporal error controller, if the requested tolerance is not met, message(s) will be output to the screen:
    - **`WARNING: mean flow subiterations failed to converge to specified temporal_err_floor level`**
    - **`WARNING: turb flow subiterations failed to converge to specified temporal_err_floor level`**
    - Note: when starting unsteady mode, first timestep ***never*** achieves target error (no error estimate first step, so target is 0)
    - Note: x-momentum residual (**`R_2`**) is the mean-flow residual targeted by the error controller
  - Tecplot file with subiteration convergence history is output to a file: **`[project]_subhist.dat`**
    - Plot (on log scale) **`R_2`** (etc) vs **`Fractional_Time_Step`**

Time Advancement - Subiterations (4/4)

All Time Steps



Final Few Time Steps



Nondimensionalization of Time

- Notation: * indicates a dimensional variable, otherwise nondimensional; the reference flow state is usually free stream ("∞ "), but need not be
- Define:
  - L^*_{ref} = reference length of the physical problem (e.g. chord in ft)
  - L_{ref} = corresponding length in your grid (*nondimensional*)
  - a^*_{ref} = reference speed of sound (e.g. ft/sec) (compressible)
  - U^*_{ref} = reference velocity (e.g. ft/sec; compressible: U^*_{ref} = Mach a^*_{ref})
  - t^* = time (e.g. sec)
- Then nondimensional time in FUN3D is related to physical time by:

KEY POINT

  - $t = t^* a^*_{ref} (L_{ref}/L^*_{ref})$ (compressible)
  - $t = t^* U^*_{ref} (L_{ref}/L^*_{ref})$ (incompressible)
  - *Usually* have L_{ref}/L^*_{ref} = 1*, but need not - e.g. typical 2D airfoil grid
  - L_{ref}/L^*_{ref} because Reynolds No. in FUN3D is defined *per unit grid length*

Determining the Time Step

- Identify a ***characteristic time*** t^*_{chr} that you need to resolve with some level of accuracy in your simulation; perhaps:
  - Some important shedding frequency f^*_{shed} (Hz) is known or estimated $t^*_{chr} \sim 1 / f^*_{shed}$
  - Periodic motion of the body $t^*_{chr} \sim 1 / f^*_{motion}$
  - You have lots of CPU time and you are hoping to resolve some range of frequencies in a DES-type simulation $t^*_{chr} \sim 1 / f^*_{highest}$
  - If none of the above, you can estimate the time it takes for a fluid particle to cross the characteristic length of the body, $t^*_{chr} \sim L^*_{ref} / U^*_{ref}$
  - $t_{chr} = t^*_{chr}\, a^*_{ref}\, (L_{ref}/L^*_{ref})$ (comp) $t_{chr} = t^*_{chr}\, U^*_{ref}\, (L_{ref}/L^*_{ref})$ (incomp)

KEY POINT

- Say you want N time steps within the characteristic time:
  - $\Delta t = t_{chr} / N$ (tip: use plenty of precision to compute, and input, Δt)
- Figure a minimum of N = 100 for reasonable resolution of t_{chr} with a 2nd order scheme - really problem dependent (*frequencies > f* may be important*); but don't over resolve time if space is not well resolved too

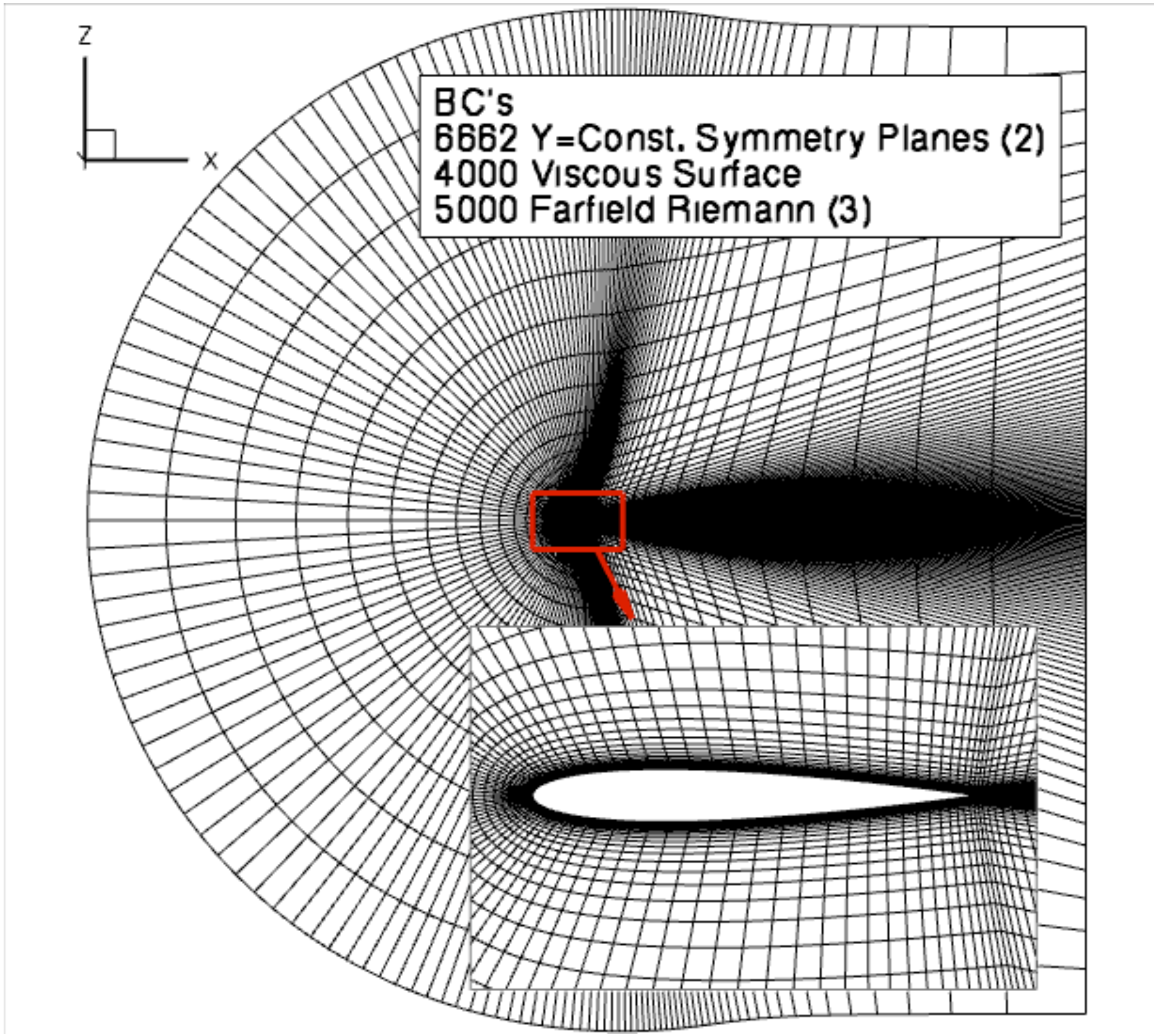
Example 1 - Unsteady Flow at High Alpha (1/9)

- Example 1 considers flow past a (2D) NACA 0012 airfoil at 45^o angle of attack - the flow separates and is unsteady
  - Re_{c^*} = 4.8 million, M_{ref} = 0.6, *assume* a^*_{ref} = 340 m/s
  - chord = 0.1m, chord-in-grid = 1.0 so L_{ref}/L^*_{ref} = 1.0/0.1 = 10 (m^{-1})
  - Say we know from experiment that lift oscillations occur at ~450 Hz
  - $t^*_{chr} = 1 / f^*_{chr}$ = 1 / 450 Hz = 0.002222 s
  - $t_{chr} = t^*_{chr}\, a^*_{ref}\, (L_{ref}/L^*_{ref})$ = (0.002222)(340)(10) = 7.555
  - $\Delta t = t_{chr} / N$ so Δt = 0.07555 for 100 steps / lift cycle
  - By way of comparison, for M = 0.6, a^*_{ref} = 340 m/s, and L^*_{ref} = 0.1 m it takes a fluid particle ~ (0.1)/(204) = 0.00049 s to pass by the airfoil; this leads to smaller, more conservative estimate for the time step, by about a factor of 5

Example 1 - Unsteady Flow (2/9)

- It takes more time than we have here to settle into a periodic state from free stream, so we'll run this as a *restart* from a previous solution, for 100 steps
- Log into your account on cypher-work14: and **`cd`** to **`Unsteady_Demos/High_Alpha`**
- There you will find a set of files:
  - **`n0012_i153.ugrid`**
  - **`n0012_i153.mapbc`**
  - **`fun3d.nml`**
  - **`n0012_i153.flow`**
  - **`qsub_high_alpha`**
  - **`time_history.lay, subit_history.lay, vort_animation.lay, u_animation.lay`**

Example 1 - Unsteady Flow (3/9)



Example 1 - Unsteady Flow (4/9)

- Flow viz: output u-velocity and y-component of vorticity
- Relevant fun3d.nml namelist data

```
&project
   project_rootname = "n0012_i153"
   case_title = "NACA 0012 airfoil, 2D Hex Mesh"
/
&governing_equations
   viscous_terms = "turbulent"
/
&reference_physical_properties
   mach_number     = 0.60
   reynolds_number = 4800000.00
   temperature     = 520.00
   angle_of_attack = 45.0
/
&force_moment_integ_properties
   x_moment_center = 0.25
/
&turbulent_diffusion_models
   turb_model = "sa"
/
```

Example 1 - Unsteady Flow (5/9)

- Relevant fun3d.nml namelist data (cont)

```
&nonlinear_solver_parameters
    time_accuracy        = "2ndorderOPT" ! Our Workhorse Scheme
    time_step_nondim     = 0.07555 ! 100 steps/cycle @ 450 Hz
    pseudo_time_stepping = "on"    ! This is the default; set for emphasis
    subiterations        = 30
    schedule_cfl         = 50.00 50.00 ! constant cfl each step; no ramping
    schedule_cflturb     = 30.00 30.00
 /
 &linear_solver_parameters
    meanflow_sweeps   = 50
    turbulence_sweeps = 30
 /
 &code_run_control
    steps        = 100  ! need ~2000 steps to be periodic from freestream
    restart_read = "on" ! "off": start from freestream
 /                      ! "on_nohistorykept": start from steady state soln
 &raw_grid
     grid_format = "aflr3"
     data_format = "ascii"
     twod_mode   = .true.
 /
```

Example 1 - Unsteady Flow (6/9)

- Relevant fun3d.nml namelist data (cont)

```
&boundary_output_variables
   primitive_variables = .false. ! turn off default
   y = .false.      ! So tecplot displays correct 2D orientation by default
   u = .true.
   vort_y = .true.
 /                  ! no boundaries specified - default is one of sym. planes
```

- Look at the **`qsub_high_alpha`** script; we will terminate subiterations if residual is 10x smaller than error estimate and get boundary animation output every 5th time step:

```
mpirun -np 24 nodet_mpi --animation_freq +5
--temporal_err_control 0.1
```

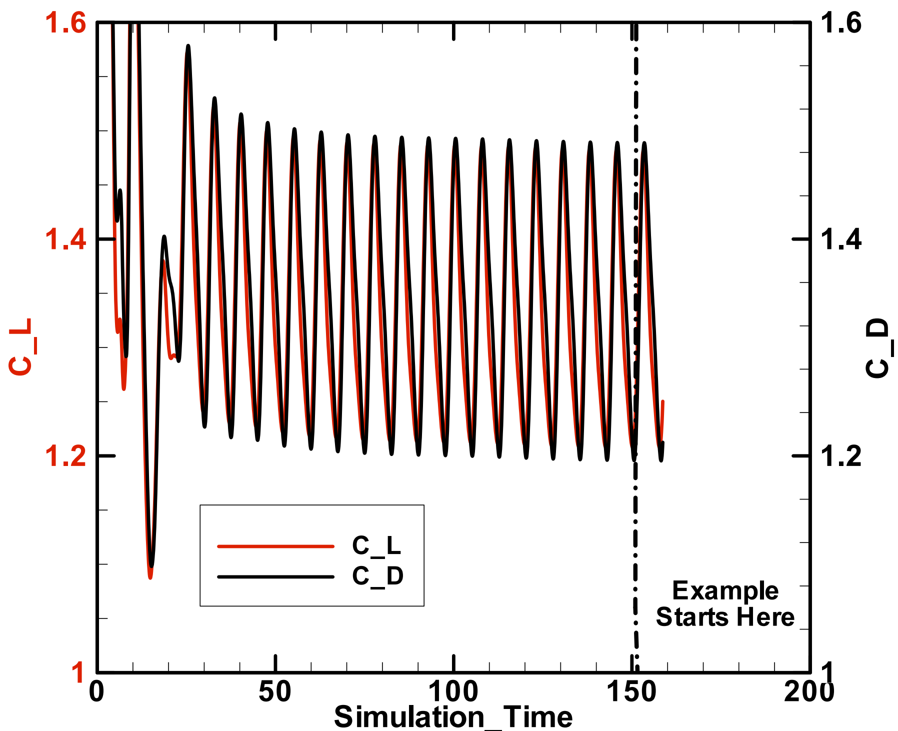
- **`qsub qsub_high_alpha`** ! will take ~4 minutes to run
- Did it work? As always, last line or screen output should be: **`Done.`**
- Subiterations converge? **`grep "WARNING" screen_output | wc`** to find zero occurrences – in this case they all did

Example 1 - Unsteady Flow (7/9)

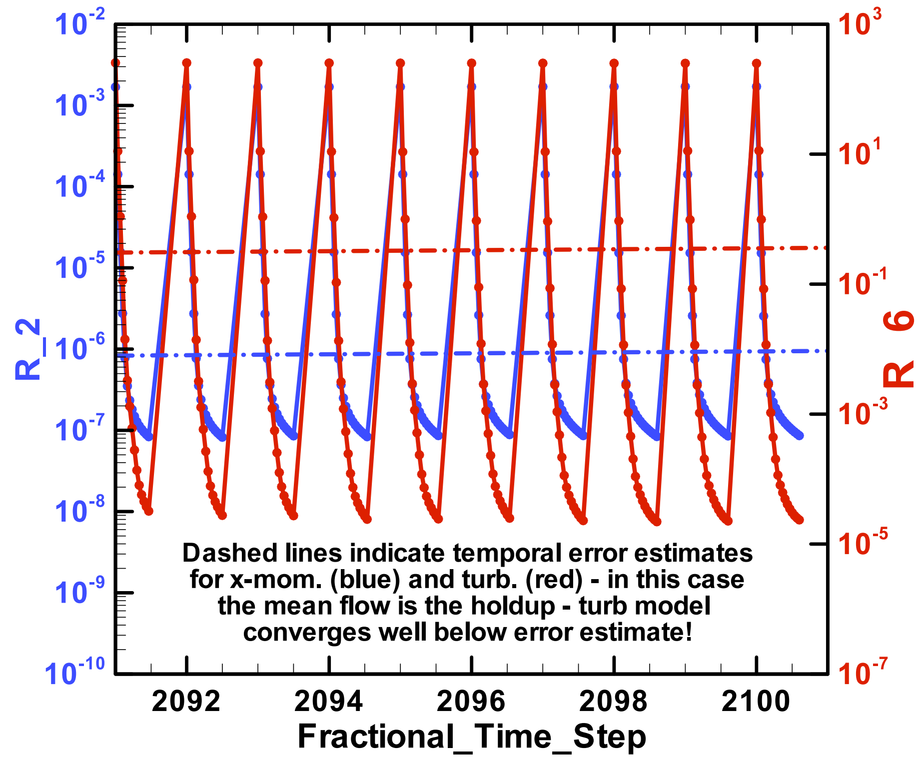
- Bring some files back for plotting…
- On cypher-work14:
  - **`tar -cvf output.tar *.lay *hist.tec n0012_i153_tec_boundary_timestep*.dat`**
- On your local machine:
  - **`mkdir High_Alpha`** and **`cd High_Alpha`**
  - **`scp cypher-work14:~/Unsteady_Demos/High_Alpha/output.tar .`**
  - **`tar -xvf output.tar`**
  - Should now have: **`time_history.lay, subit_history.lay, u_animation.lay, vort_animation.lay, n0012_i153_hist.tec, n0012_i153_subhist.dat, n0012_i153_tec_boundary_timestep2005.dat,`** … **`n0012_i153_tec_boundary_timestep2100.dat`**

Example 1 - Unsteady Flow (8/9)

Complete Time History
(time_history.lay)



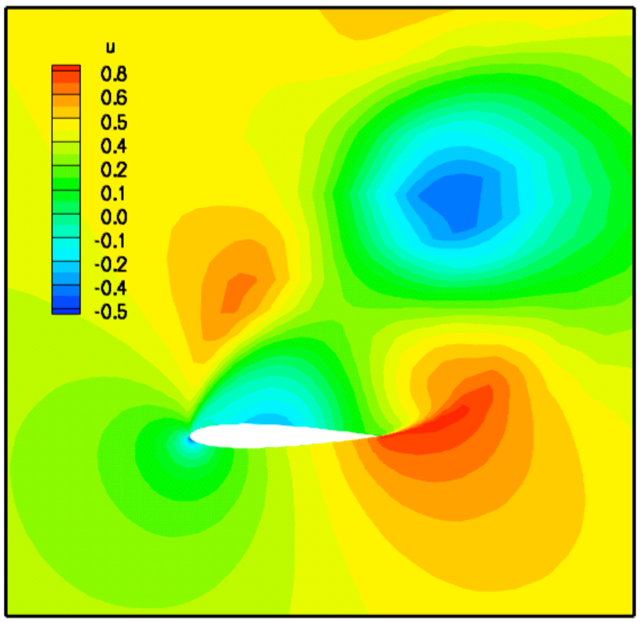
Subiteration Convergence, Final 10 Steps
(subit_history.lay)

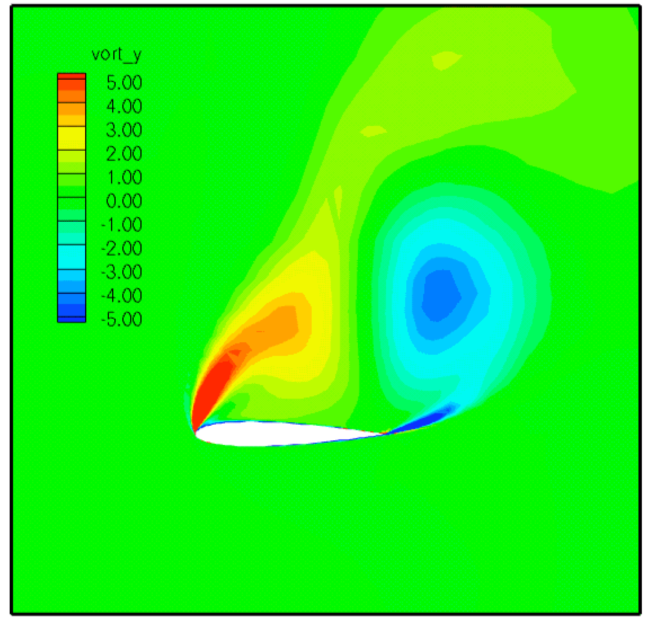


Example 1 - Unsteady Flow (9/9)

- Animation of Results

X-Component of Velocity
(u_animation.lay)





Y-Component of Vorticity
(vort_animation.lay)
note: Tecplot default contour levels too large – set levels to +/- 5 or so

Mesh / Body Motion (1/3)

- A body is defined as a user-specified collection of solid boundaries in grid
  - Generally, in `&raw_grid` input, should opt to lump multiple boundaries by family type to minimize subsequent input
- Body motion options:
  - Several built-in functions: translation and/or rotation with either constant velocity or periodic displacement – body is rigid
  - Read series of surface files – rigid or deforming (not covered here)
  - 6 DOF with UAB libraries (covered in another session)
  - Application-specific: mode-shape based aeroelasticity (linear structures); rotorcraft nonlinear beam (covered in other sessions)
- Mesh motion options – to accommodate body motion:
  - Rigid - maximum 1 body containing all solid surfaces (unless overset)
  - Deforming – can support multiple bodies without overset, but limited to small relative displacements
  - Combine with overset for large displacements (covered tomorrow)

Mesh / Body Motion (2/3)

- Rigid mesh motion via application of 4x4 transform matrix - fast; positivity of cell volumes guaranteed to be maintained
- 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; need refine package***

Mesh / Body Motion (3/3)

- 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 the default 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 untangled (don't have refine, or have mixed elements), try reducing **`tol`**
  - GMRES is not used for rigid motion
- ***All*** dynamic-mesh simulations require the CLO **`--moving_grid`**
- ***All*** dynamic-mesh simulations require some input data via an auxiliary namelist file: **`moving_body.input`**

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
  - Exception: 6-DOF and modal-based aeroelasticity use primarily *dimensional* data as inputs
- 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)

Nondimensionalization of Motion Data (2/2)

- Oscillation frequency of the physical problem can be specified in different forms
  - f^* = frequency (e.g. Hz)
  - ω^* = 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 $\sin(2 \pi f t)$ where, in terms of input variables f = **`rotation_freq`** or f = **`translation_freq`**
  note: currently no provision for a phase lag to sin()
- So the corresponding nondimensional frequency for FUN3D is
  - $f = f^* L^*_{ref} / a^*_{ref}$ (comp) $f = f^* L^*_{ref} / U^*_{ref}$ (incomp)
  - $f = \omega^* / (2\pi) \, L^*_{ref} / a^*_{ref}$ $f = \omega^* / (2\pi) \, L^*_{ref} / U^*_{ref}$
  - $f = k \, M^*_{ref} / \pi$ $f = k / \pi$

Overview of **`moving_body.input`** (1/2)

- Note: just the most-used items shown here – see web site for complete list; *all input is dimensionless unless noted*
- The **`&body_definitions`** namelist defines the body(s) in motion:

```
&body_definitions          ! below, index b=body#  i=boundary#
 n_moving_bodies           ! how many bodies in motion
 body_name(b)              ! set unique name for each body
 n_defining_boundary(b)    ! # boundaries to define this body; shortcut:
                           ! a value -1 will use all solid walls;
                           ! only use if n_moving_bodies = 1
 defining_boundary(i,b)    ! list of boundaries that define this body; if
                           ! n_defining_boundary = -1 list one value;0 OK
 motion_driver(b)          ! mechanism by which the body is moved:
                           ! 'none','forced','aeroelastic','file', '6dof'
 mesh_movement(b)          ! specifies how mesh will move to accommodate
                           ! body motion: 'rigid', 'deform'
/
```

- Caution: boundary numbers must reflect any lumping applied at run time!
- All variables above except **`n_moving_bodies`** are set for each body
- Current limitation: value of **`mesh_movement`** must be same for all bodies

Overview of `moving_body.input` (2/2)

- 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`**), frequency is *no*t a circular frequency

Output Files

- In addition to the usual output files, for moving-grids 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 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 website for details

Example 2 - Pitching Airfoil (1/10)

- 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)
- Same grid and mapbc files as Example 1; *other files differ*
- Setting the FUN3D data:
  - **`angle_of_attack = 2.89  rotation_amplitude = 2.41`**
  - Recall f = k M^*_{ref} / π
  - **`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 2 - Pitching Airfoil (2/10)

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

    sin(2π **`rotation_freq`** T) = sin(2π)

    T = 1 / **`rotation_freq`** (this is our t_{chr})

    for N steps / cycle, T = N Δt so

    Δt = T / N = (1 /**`rotation_freq`**) / N
  - Again, use 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, as for the previous example, assume a^*_{inf}

Example 2 - Pitching Airfoil (3/10)

- Again, run as a 100 step (1 pitch cycle) *restart* from a previous solution
- Log into your account on cypher-work14: and **`cd`** to **`Unsteady_Demos/Pitching_Airfoil`**
- There you will find a set of files:
  - **`n0012_i153.ugrid`** **(same as example 1)**
  - **`n0012_i153.mapbc`** **(same as example 1)**
  - **`fun3d.nml`**
  - **`moving_body.input`**
  - **`n0012_i153.flow`**
  - **`qsub_pitching_airfoil`**
  - **`time_history.lay, subit_history.lay, mach_animation.lay, cp_animation.lay`**

Example 2 - Pitching Airfoil (4/10)

- Relevant fun3d.nml namelist data (only namelists that differ are shown)
- Use "sampling" output on plane rather than boundary output

```
&reference_physical_properties
  …
   angle_of_attack = 2.89
/
&nonlinear_solver_parameters
  …
   time_step_nondim     = 0.64801842  ! 100 steps/pitch cycle
/
&sampling_output_variables
  primitive_variables = .false.
  y    = .false.
  cp   = .true.
  mach = .true.
/
&sampling_parameters
  number_of_geometries = 1
  type_of_geometry(1)  = 'plane' ! 2D case, should get same as sym. plane!
  plane_center(:,1)    = 0., -0.5, 0.  ! x,y,z
  plane_normal(:,1)    = 0.,  1.0, 0.
/
```

Example 2 - Pitching Airfoil (5/10)

- 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,          ! all solid boundaries constitute body (though only have 1)
  defining_bndry(1,1) =  0,          ! index 1: boundary number index 2: body number
  motion_driver(1)     = 'forced',   ! 'forced', '6dof', 'file', 'aeroelastic'
  mesh_movement(1)     = 'rigid',    ! 'rigid', 'deform'
/
&forced_motion
  rotate(1)              = 2,           ! rotation type: 1=constant rate 2=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 2 - Pitching Airfoil (6/10)

- Look at the **`qsub_pitching`** script: this is a moving grid case so we must indicate that; terminate subiterations when residual is 10x smaller than error estimate, and get sampling animation output every 5th time step:

```
mpirun -np 24 nodet_mpi --moving_grid --sampling_freq +5
--temporal_err_control 0.1
```

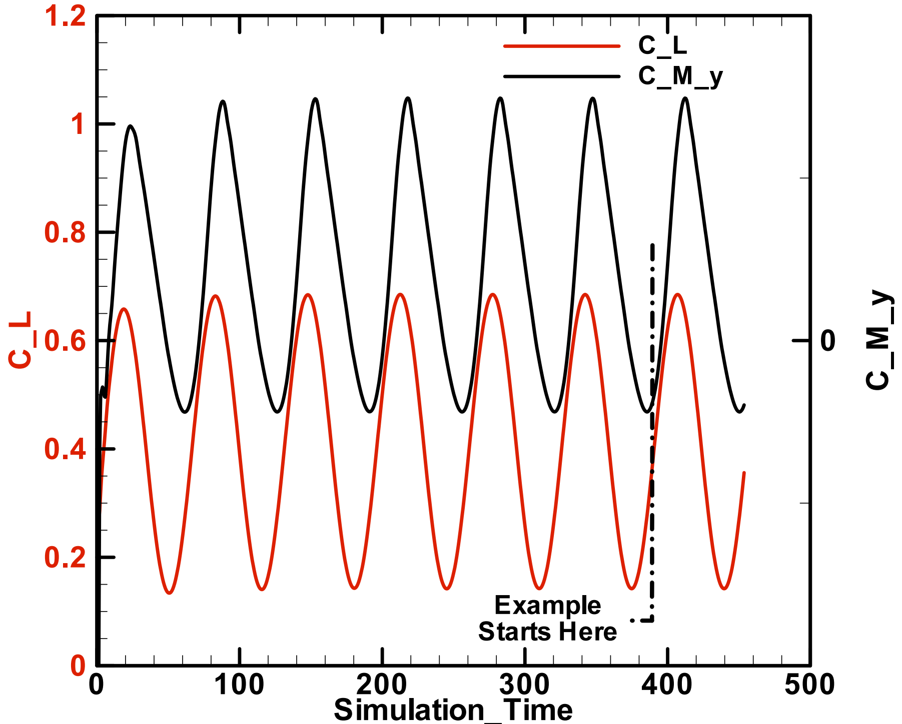
- Note: use sampling output here to illustrate what you might do in 3D to extract a plane data from the flow field, instead of, or in addition to, boundary output like we did in Example 1
- **`qsub qsub_pitching`** ! will take ~6 minutes to run
- Did it work? As always, last line or screen output should be: **`Done.`**
- Subiterations converge? **`grep "WARNING" screen_output | wc`** to find 16 occurrences – in this case 16 time steps don't *quite* reach the cutoff level in the max 30 subiterations we allowed

Example 2 - Pitching Airfoil (7/10)

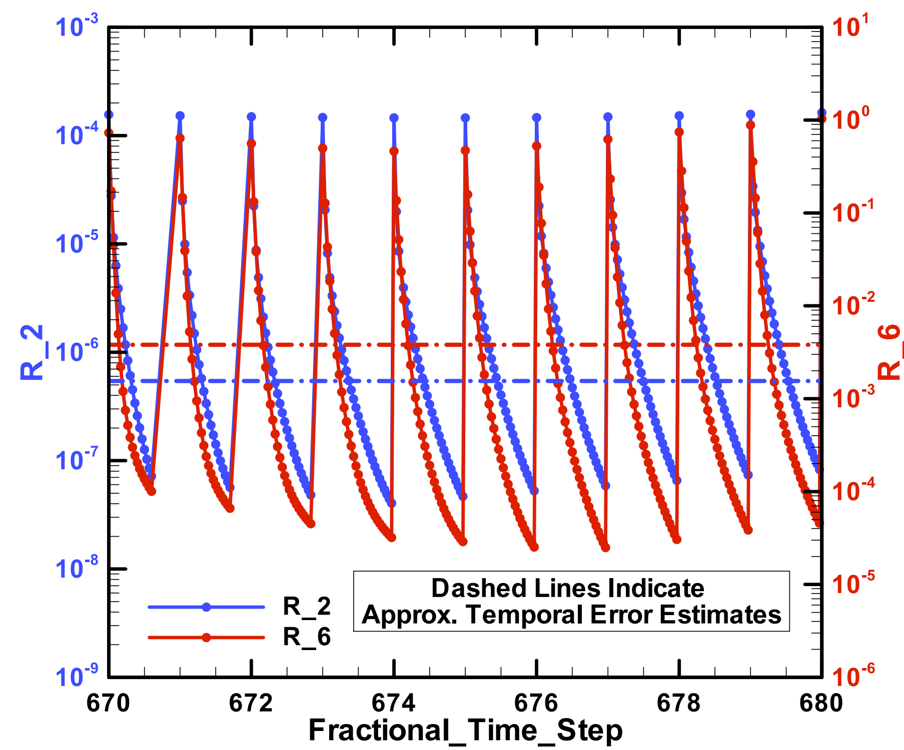
- Bring some files back for plotting…
- On cypher-work14:
  - **`tar -cvf output.tar *.lay *hist.tec n0012_i153_tec_sampling_geom1_timestep*.dat`**
- On your local machine :
  - **`mkdir Pitching_Airfoil`** and **`cd Pitching_Airfoil`**
  - **`scp cypher-work14:~/Unsteady_Demos/Pitching_Airfoil/output.tar .`**
  - **`tar -xvf output.tar`**
  - Should now have: **`time_history.lay, subit_history.lay, mach_animation.lay, cp_animation.lay, n0012_i153_hist.tec, n0012_i153_subhist.dat, n0012_i153_tec_sampling_geom1_timestep605.dat,`** … **`n0012_i153_tec_sampling_geom1_timestep700.dat`**

Example 2 - Pitching Airfoil (8/10)

Time History
(time_history.lay)

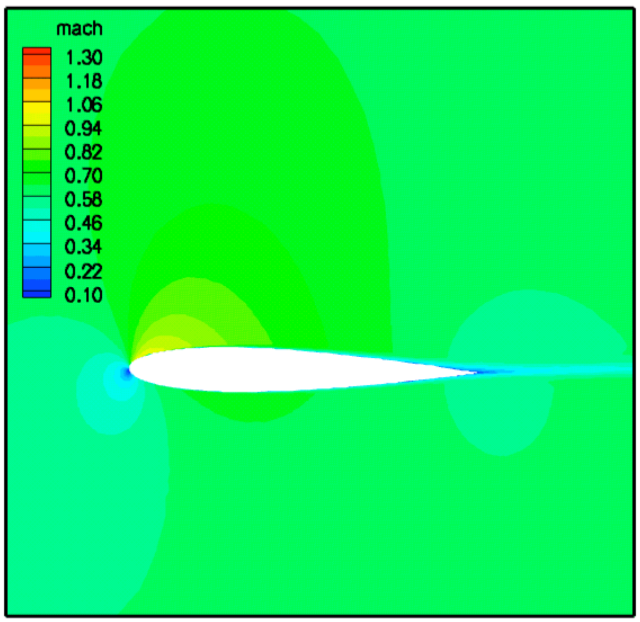


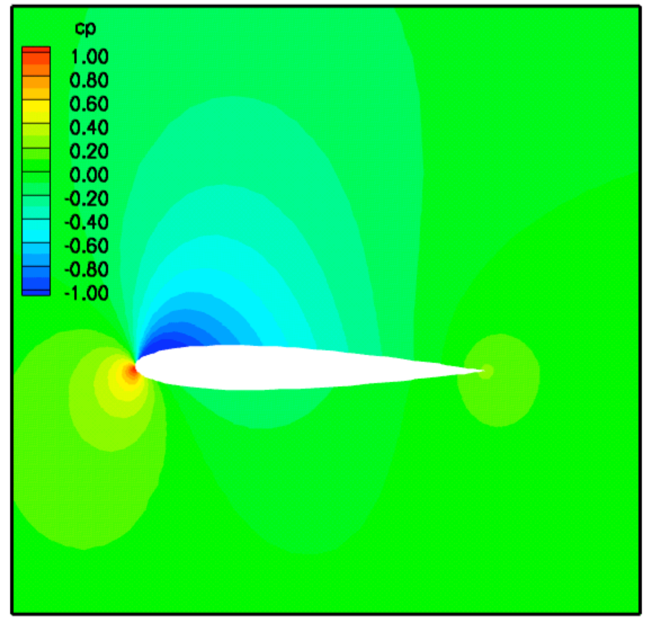
Sample Subiteration Convergence
(where mean flow just misses tolerance)
(subit_history.lay)



Example 2 - Pitching Airfoil (9/10)

Mach Number
(mach_animation.lay)





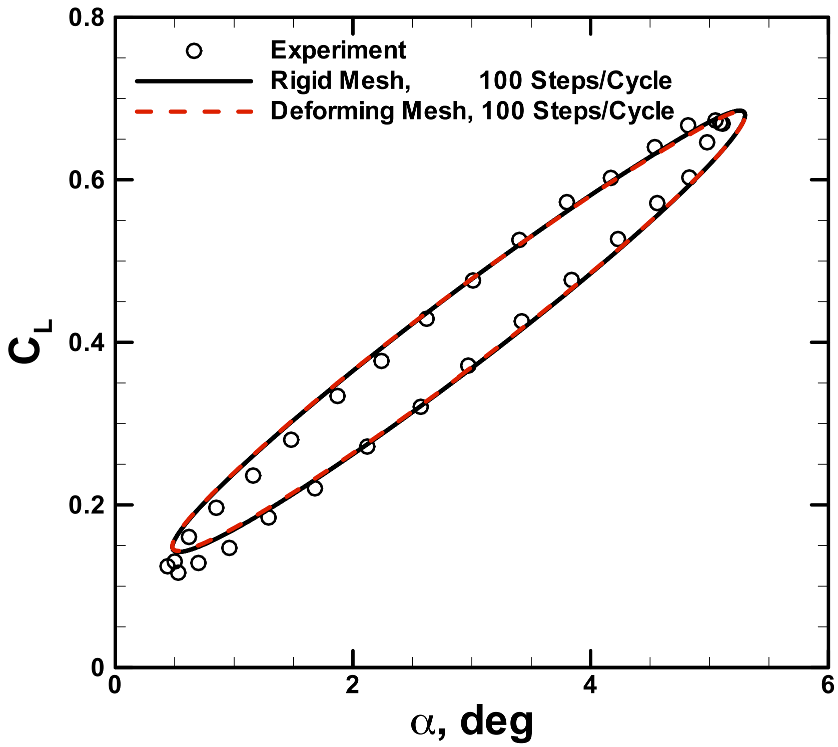
Pressure Coefficient
(cp_animation.lay)

Example 2 - Pitching Airfoil (10/10)

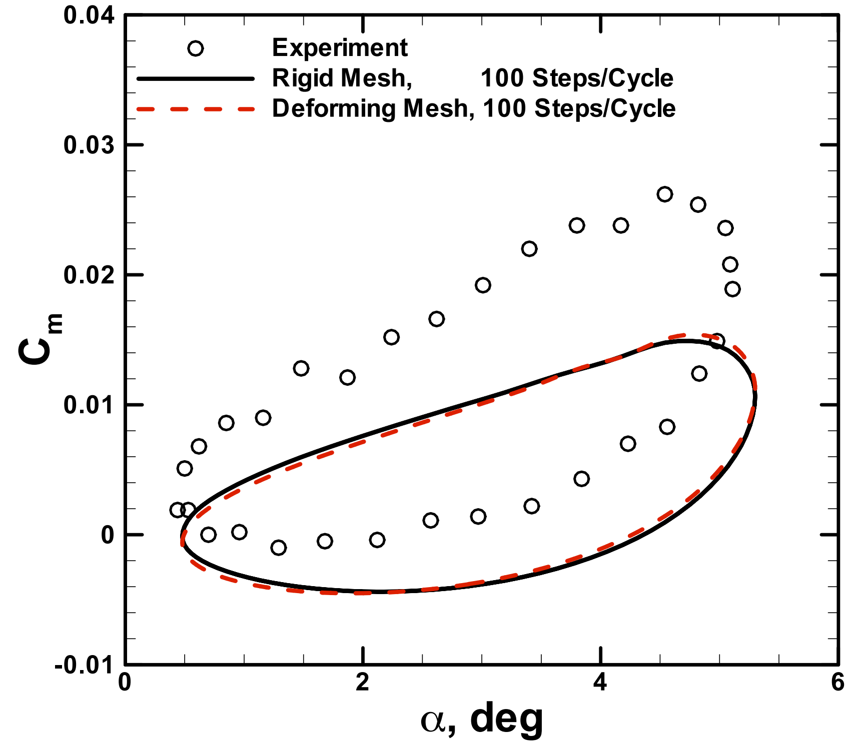
Comparison with Landon, AGARD-R-702, Test Data,1982

Note: comparison typical of other published CFD results

Lift vs. Alpha



Pitching Moment vs. Alpha



We ran rigid mesh: deforming mesh produces nearly identical results

Troubleshooting Body / Grid Motion

- When first setting up a dynamic mesh problem, ***strongly*** suggest using one or both of the CLO's **`--body_motion_only`** and **`--grid_motion_only`**
- Both options are used in conjunction with **`--moving_grid`**, and turn off the solution of the flow equations for faster processing
  - **`--body_motion_only`** also turns off the grid motion; especially useful for 1st check of a deforming mesh case since the elasticity solver is also bypassed; cannot restart from this
  - **`--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; restart is possible
  - 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`**, **`[project]_hist.tec`**, etc.:
- Input
  - **`moving_body.input`** (dynamic grids only)
- Output
  - **`[project]_subhist.dat`**
  - **`PositionBody_N.dat`** (dynamic grids only)
  - **`VelocityBody_N.dat`** (dynamic grids only)
  - **`AeroForceMomentBody_N.dat`** (dynamic grids only)

FAQ's

- Most frequent questions arise regarding how to set the time step… covered at great length here
- The second-most (maybe the first) asked question is how much CPU time does it take?
  - If you have to ask you can't afford it !
  - Really depends on how small a time step is used, and how many subiterations are used/needed
- Any special considerations for *incompressible* time dependent / moving grid cases? Yes, for *moving grids*:
  - Must use CLO **`--roe_jac`** in order to use correct linearization routines
  - ~~However, incompressible flow on moving grids is currently not functional - hope to have fixed soon~~ Fixed in v11.2
  - Use BC 5050 or 5025 instead of 5000

What We Learned

- Overview of governing equations for unsteady flows with moving grids
- Time discretization and the subiteration scheme
  - Must drive subiteration residual toward zero to recover design order
  - Temporal error controller
  - How to assess subiteration convergence
- Nondimensionalization of time and motion parameters
  - Determining the time step
  - Typically more involved than steady-state cases where all you usually have to consider are the familiar Re and Mach numbers
- Body and mesh motion options
  - Primarily focused on specified (“forced”) motion
  - Other options available; some covered in subsequent sessions
- Animation as a visualization and troubleshooting tool