FUN3D v14.0 Training
Aeroelasticity

Kevin Jacobson

Learning Goals

What we will cover:

- Updates to the FUN3D aeroelastic modal solver since version 13.4
  - Modal mesh deformation
  - GPU simulations with the modal solver
  - Aeroelastic generic gas simulations
- Python interfaces for aeroelastic analysis
  - Interacting with internal modal solver
  - Coupling to an external structural solver

What we will not cover:

- Material covered in the 13.4 training (December 2018 Workshop)
- Structural modeling or load and displacement transfers
- Linearized Frequency Domain (separate training)

Miscellaneous Updates since 13.4 Training (1/3)

- The 14.0 manual adds a section on the internal modal solver:

- Mode shape interpolation:
  - In previous trainings, we have recommended Jamshid Samareh's "Discrete Data Transfer Between Dissimilar Meshes" as an option for mode shape interpolation when using the internal modal solver.
  - Since the previous training, Steve Massey has released an open-source Radial Basis Function interpolation code:
    - https://github.com/nasa/rbf
    - This has become the primary tool of the Aeroelasticity Branch at Langley Research Center

Miscellaneous Updates since 13.4 Training (2/3)

- SLAT: a new linear solver for mesh deformation distributed with FUN3D 14.0
  - GMRES solver from NASA
  - Supports complex mode
  - SLAT is the new default **`linear_solver`** if you do not compile with SPARSKIT
    - SPARSKIT is still the default is FUN3D is configured with SPARSKIT

- Added an option to the prescribed motion of aeroelastic modes that allows a nonzero offset with periodic motion, **`moddfl=1`**
  - **`moddfl_offset(mode, body) = 0.0`** sets the mean or offset
  - **`moddfl_freq(mode, body) = 0.0`** sets the frequency
  - **`moddfl_amp(mode, body) = 0.0`** sets the amplitude
  - **`moddfl_t0(mode, body) = 0.0`** sets the start time

Miscellaneous Updates since 13.4 Training (3/3)

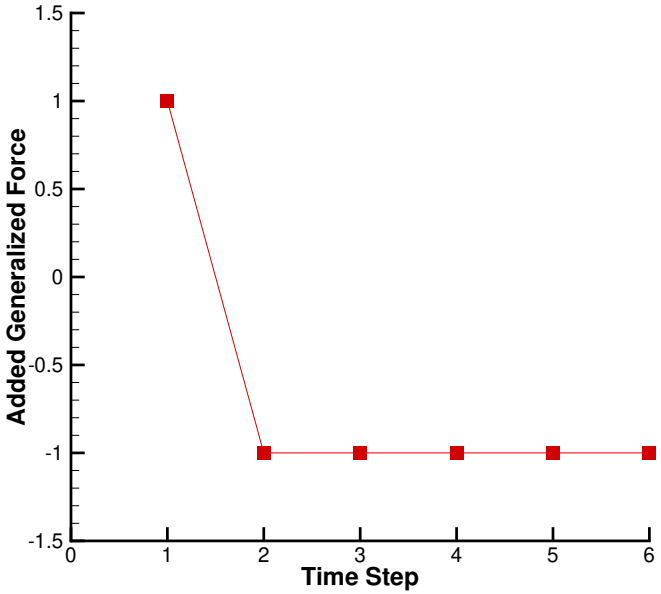
- A new option is available to ignore grid velocity terms in the flux computation:

```
&global
  ignore_grid_velocity = .false.
/
```

  - Can speed up static aeroelastic simulations
  - Only use **`ignore_grid_velocity = .true.`** for unsteady problems where you only care about the final steady state like static aeroelastic analysis, which must be run in unsteady mode with the internal modal structural solver
    - We will use this in the BSCW example later in this training

Internal Modal Solver - Static Generalized Forces

- Since the previous training, there has been one change to an existing input in the **`&aeroelastic_modal_data`** namelist.
- Prior to version 13.7, the modal solver input **`gforce0`** applied a *positive* perturbation on the first step, then a *negative* offset of the same magnitude during the rest of the simulation.
  - If using the offset to apply a non-aerodynamic load such as weight, this **`gforce0`** input would perturb the solution at each simulation restart.



`gforce0=1.0` behavior prior to 13.7

- In version 13.7 and later, this functionality was split into two inputs:
  - **`gforce0(mode,body) = 0.0`** applies a perturbation generalized force on the first step only.
  - **`gforce_static(mode,body) = 0.0`** applies a constant generalized force.
- When performing a restart, you should set **`gforce0`** to 0.0 if you do not want to re-perturb the solution.

Modal Mesh Deformation - Process (1/2)

- Because the modal structural solver and mesh deformation are linear, the modal mesh deformation mode of the internal modal solver uses superposition to perform mesh deformation.
- The process:
  1. For each mode, compute volume deformation for each mode displaced by a reference amplitude, **`modal_ref_amp`**
  2. Store the volume_mode_shape = (volume node displacements from #1) / **`modal_ref_amp`**
  3. At each time step, the volume mesh = sum(modal_displacement x volume_mode_shape)

- The process is more efficient…
  - One linear elasticity solution per mode, instead of performing one linear elasticity solution per time step
  - But, it requires additional memory: need to store all the volume mode shapes in memory, 3 x (number of modes) x (nodes in the mesh)

Modal Mesh Deformation - Inputs (2/2)

- The modal mesh deformation is controlled by the **`&aeroelastic_modal_data`** namelist in **`moving_body.input`**:
  - It is activated with **`use_modal_deform = .true.`**
  - The input **`modal_ref_amp(mode,body) = 1.0`** sets the amplitude for each surface mode when computing the volume mode shape. The volume mode shapes are normalized (step 2 on previous slide), so this value can either be an estimate of the modal displacement expected during the simulation or a simply small value.

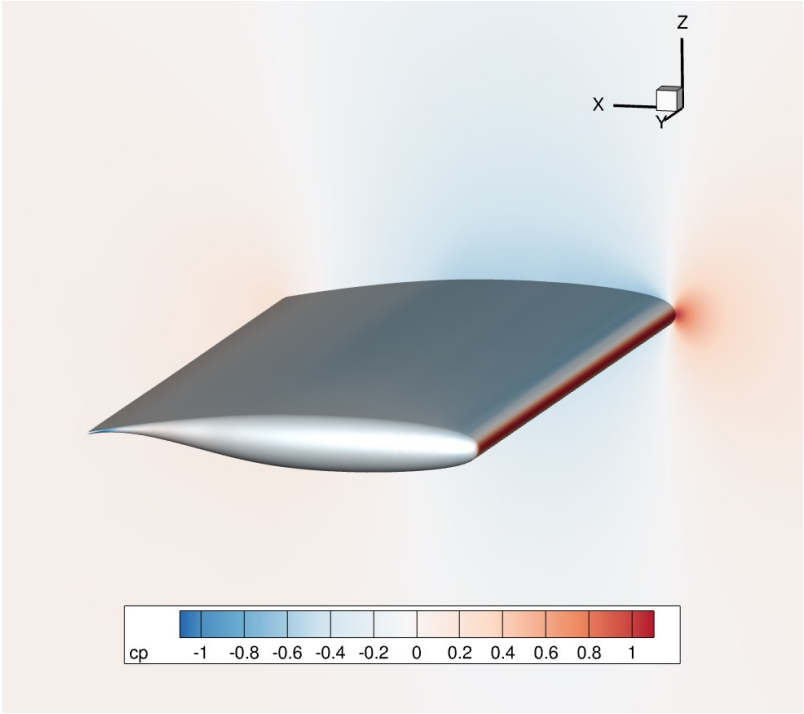
- If you are going to run a multiple simulations with the same mode shapes and the same number of processors, volume mode shapes can be written to files and read on subsequent simulations to skip computing the volume modes.
  - On the first simulation, set **`write_volume_modes = .true.`**
  - On the subsequent simulation(s), set **`read_volume_modes = .true.`**

GPU Simulations with the Modal Solver

- A full training session on GPU-based simulations will be given later in the series
- Aeroelastic simulations with the internal modal solver are supported in the GPU path of FUN3D
  - The GPU path requires using the modal mesh deformation, **`use_modal_deform = .true.`** in **`&aeroelastic_modal_data`** and **`recompute_turb_dist = .false.`** in **`&global`**
  - The other **`&aeroelastic_modal_data`** options are supported, such as **`moddfl`** perturbations for Reduced Order Model generation
- The initial mesh deformations in the modal mesh deformation process are performed on the CPU
- The standard mode of the GPU path is one MPI rank or CPU core per GPU, which can make the initial mesh deformation quite slow
  - It is highly recommended to use the **`write_volume_modes`** and **`read_volume_modes`** when performing GPU-based aeroelastic analysis
  - Another way to mitigate this bottleneck is to use the NVIDIA Multi-Process Service (MPS) to have multiple MPI ranks per GPU. See the GPU-based simulation training for more information

GPU and Modal Mesh Deformation Example (1/4)

- Example #1: Benchmark Supercritical Wing (Aeroelastic Prediction Workshop Series)
  - Mesh and mode shapes available from the AePW2 website
  - Other inputs in bscw directory of the tutorial tarball
  - Standard flutter process (see Dec. 2018 training)
    1. Steady jig shape analysis
    2. Static aeroelasticity
    3. Dynamic aeroelasticity
- Example features:
  - Using the **`ignore_grid_velocity`** flag for static aeroelasticity
  - Modal mesh deformation
  - GPU aeroelastic analysis
    - Remove the **`&gpu_support`** namelist if running on CPUs

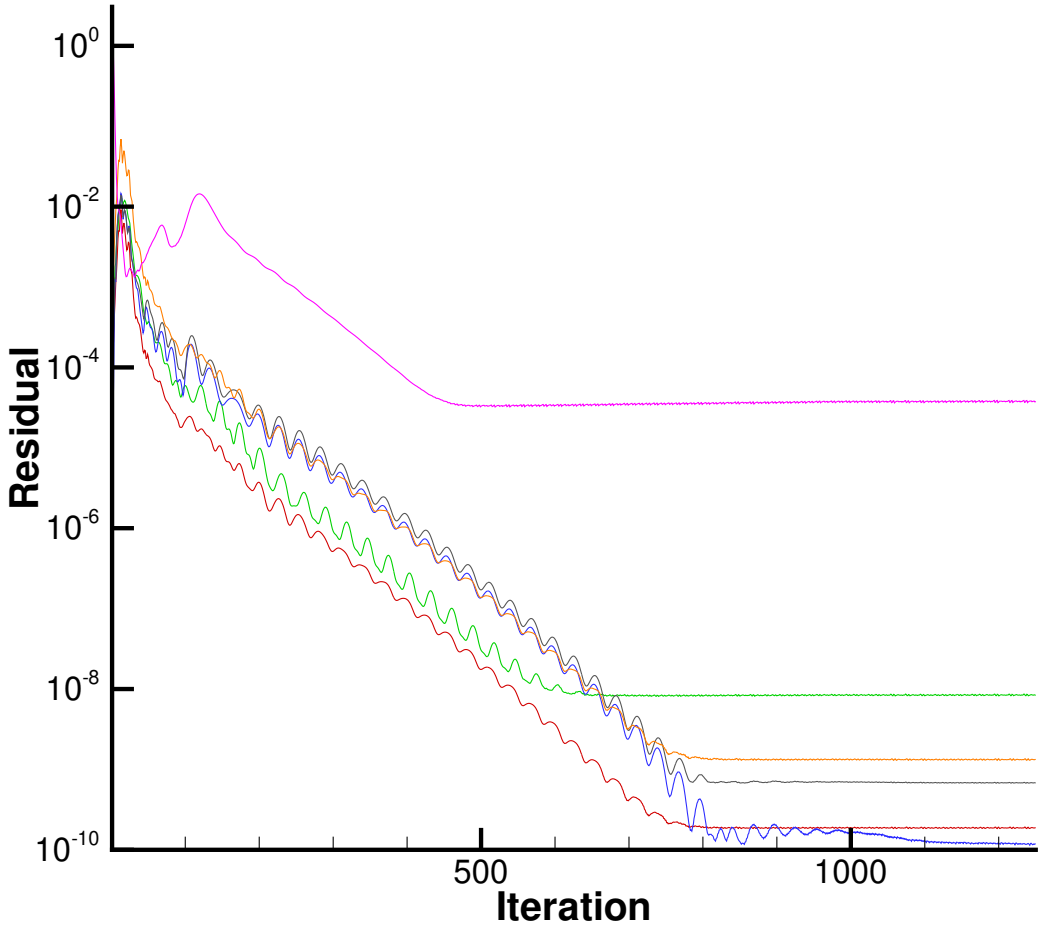


GPU and Modal Mesh Deformation Example (2/4)

Step 1: Steady aerodynamics

- **`fun3d.nml`**:

```
&reference_physical_properties
  temperature_units = "Kelvin"
  mach_number =        0.74
  reynolds_number =  278125.0
  temperature =     304.911111
  angle_of_attack = 0.0
  gamma = 1.136
/
&nonlinear_solver_parameters
  time_accuracy = "steady"
/
&code_run_control
  steps = 1250
  stopping_tolerance =    1.0E-15
  restart_read = "off"
/
```



Steady BSCW convergence history

GPU and Modal Mesh Deformation Example (3/4)

Step 2: Static aeroelastic analysis using the modal mesh deformation

- **`fun3d.nml`**:

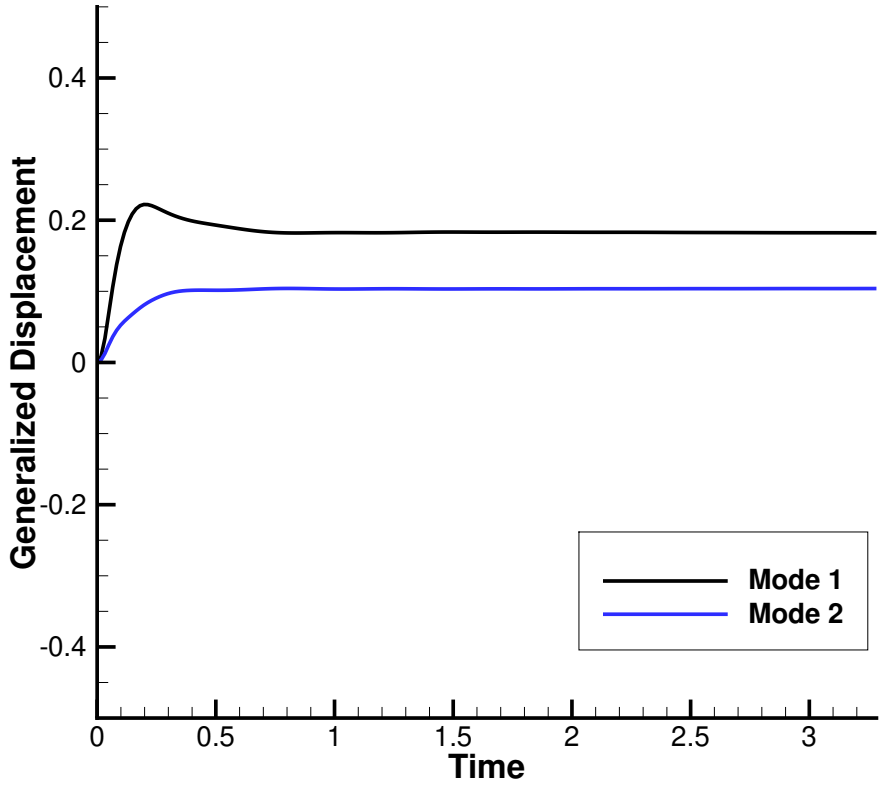
```
&global
  boundary_animation_freq = -1
  moving_grid = .true.
  ignore_grid_velocity = .true.
  recompute_turb_dist = .false. ! must be false for GPU aeroelastic cases
/

! remove this namelist for CPU usage
&gpu_support
  use_cuda = .true.
  cuda_start_mps = .true.
/
```

- **`moving_body.input`**:

```
&aeroelastic_modal_data
  nmode(1)   = 2
  uinf    = 4508.4
  grefl   = 1.0
  qinf    = 1.1736111
  freq(1,1) = 20.92
  freq(2,1) = 32.67
  gmass(1,1)   = 1.0
  gmass(2,1)   = 1.0
  damp(1,1)   = 0.99999
  damp(2,1)   = 0.99999

  use_modal_deform = .true.
  write_volume_modes = .true.
  modal_ref_amp(1,1) = 0.1
  modal_ref_amp(2,1) = 0.1
/
```



Static aeroelastic BSCW modal history

Run commands:

```
cp ../steady/*flow .
mpiexec nodet_mpi --aeroelastic_internal > static.out
```

GPU and Modal Mesh Deformation Example (4/4)

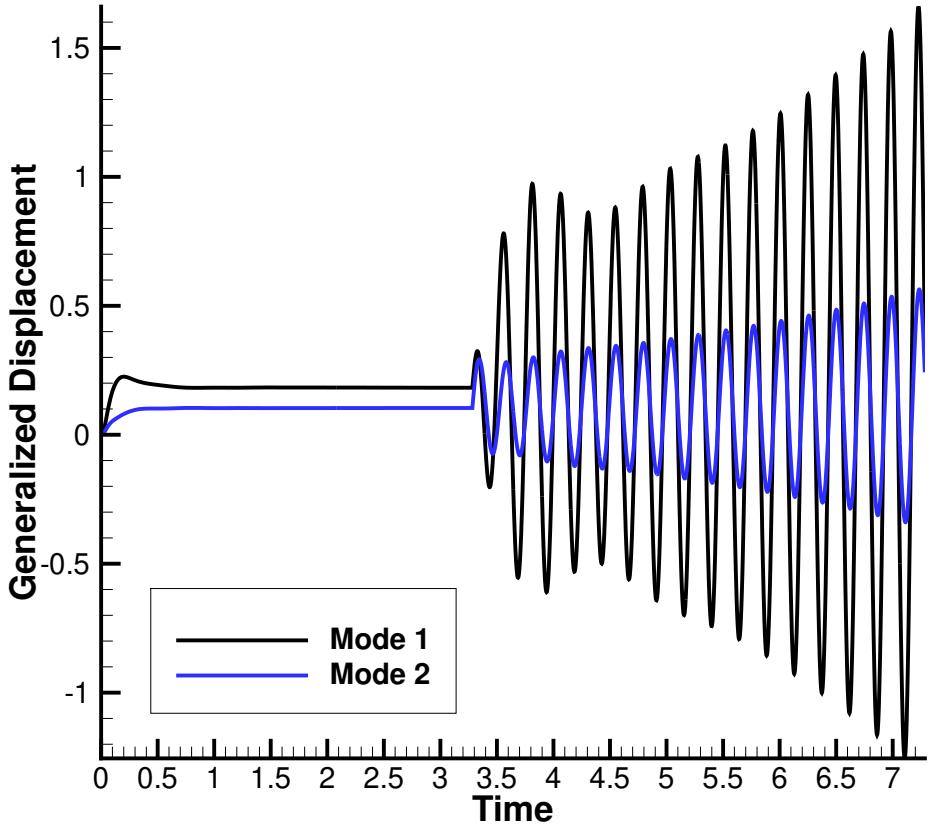
Step 3: Dynamic aeroelastic analysis

- **fun3d.nml:**

```
&nonlinear_solver_parameters
  time_accuracy = "2ndorderOPT"
  time_step_nondim = 12.1876
  subiterations = 15
/
&global
  boundary_animation_freq = -1
  moving_grid = .true.
  ignore_grid_velocity = .false.
  recompute_turb_dist = .false. ! must be false for GPU aeroelastic cases
/
```

- **moving_body.input:**

```
&aeroelastic_modal_data
  nmode(1)  = 2
  uinf    = 4508.4
  grefl   = 1.0
  qinf    = 1.1736111
  freq(1,1) = 20.92
  freq(2,1) = 32.67
  gmass(1,1)  = 1.0
  gmass(2,1)  = 1.0
  gvel0(1,1) = 5.0
  gvel0(2,1) = 5.0
  damp(1,1)  = 0.0
  damp(2,1)  = 0.0
  use_modal_deform = .true.
  read_volume_modes = .true.
/
```



Static and dynamic aeroelastic BSCW modal history

Run commands:

```
cp ../static/*flow .
cp ../static/*restart .
cp ../static/*volume_modes* .
cp ../static/*hist* .
mpiexec nodet_mpi --aeroelastic_internal > dynamic.out
```

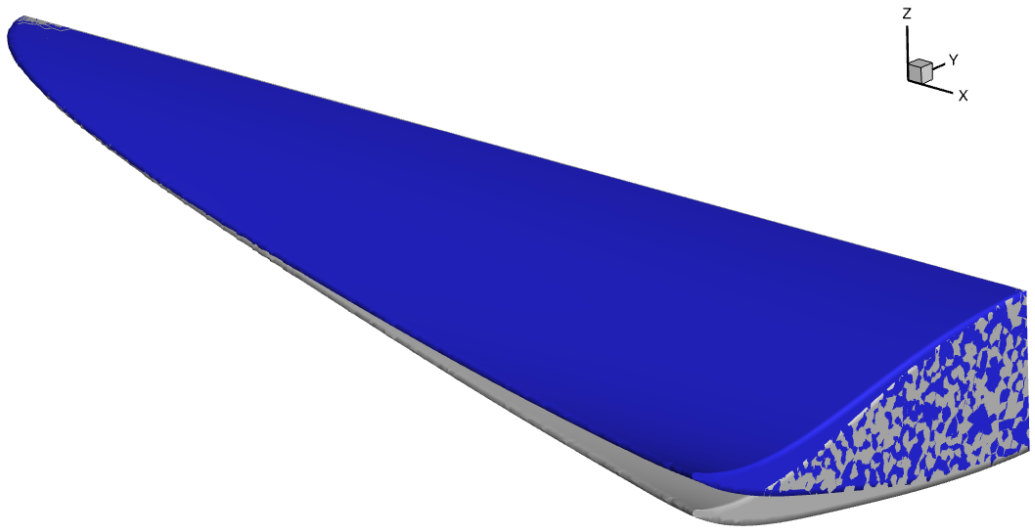
Aeroelastic Generic Gas Simulations (1/2)

- Version 14.0 introduces support for modal aeroelastic analyses with the generic gas path
  - Since modal solver inputs are dimensional, there is nothing special about the modal solver inputs for a generic gas case
  - Generic gas + modal aeroelastic analysis is supported on GPUs
- Example #2: generic waverider - static aeroelastic analysis
  - Using perfect gas model through the generic gas path
  - Step 1: Steady analysis
    - **fun3d.nml:**

```
&reference_physical_properties
  ! Mach 8, 30km
  dim_input_type    = 'dimensional-SI'
  velocity          = 2414.19760583   ! m/s
  density           = 0.0180153131776 ! kg/m^3
  angle_of_attack   = 5.0
  temperature       = 226.65          ! K
  temperature_units = "Kelvin"
/
```

    - **tdata:**

```
perfect_gas
```



Static aeroelastic displacements of the semispan waverider model

Aeroelastic Generic Gas Simulations (2/2)

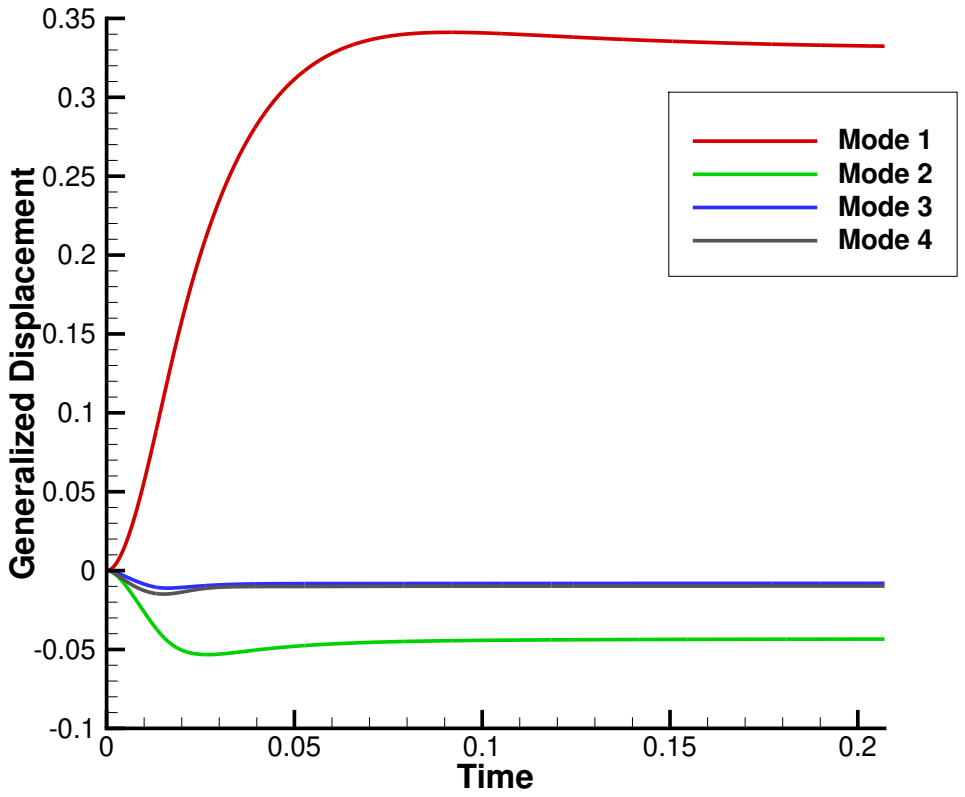
Step 2: Static aeroelastic analysis

- **fun3d.nml:**

```
&governing_equations
  eqn_type      = 'generic'
  viscous_terms = 'turbulent'
/
&reference_physical_properties
  ! Mach 8, 30km
  dim_input_type    = 'dimensional-SI'
  velocity          = 2414.19760583   ! m/s
  density           = 0.0180153131776 ! kg/m^3
  angle_of_attack   = 5.0
  temperature       = 226.65          ! K
  temperature_units = "Kelvin"
/
```

- **moving_body.input:**

```
&aeroelastic_modal_data
  nmode(1)  = 4
  uinf   = 2414.19760583
  qinf   = 52499.7759999
   .
   .
   .
/
```



Modal displacement histories

Run commands:

```
cp ../steady/*flow .
mpiexec nodet_mpi --aeroelastic_internal > static.out
```

- **uinf** is used in the flow-to-structural time step scaling and should be consistent with **fun3d.nml**
- **qinf** scales the forces and can be varied, although it is then not a match-point aeroelastic analysis

Python Interfaces

- The FUN3D python interfaces are for more coupling flexibility without the need for modifying the FUN3D source code
  - More flexibility means fewer guardrails (easier to break FUN3D)
  - This is an advanced topic. You should:
    - Feel very comfortable with modal aeroelastic analysis in FUN3D
    - Be willing to debug Python code with MPI operations
- General guidelines:
  1. Run a single FUN3D analysis per script
     - Much of FUN3D was written without reentry in mind
  2. Do not use multiple instances of the solver simultaneously
     - The FORTRAN variables in FUN3D are global data that will overwrite each other across instances
  3. Command line options are not passed to FUN3D when driven via Python
     - Some may be provided to the initialization as a Python dictionary
  4. FUN3D indices (body number, mode number, etc.) are 1-based

Compiling FUN3D Python Extension Modules

- Require Python packages: cython, numpy, mpi4py
- FUN3D external dependencies need to be compiled with the **`-fPIC`** (ParMETIS, Zoltan, etc.)
- FUN3D configuration options:
  - **`../configure --enable-python --enable-shared [other options]`**
- Getting FUN3D to use your Python setup:
  - The FUN3D **`configure`** script will search your PATH for python, cython, and python-config
  - If it cannot them or finds the wrong ones, environment variables can be used:

`PYTHON`	**`the Python interpreter`**
`PYTHON_CONFIG`	**`Path to python-config`**
`CYTHON`	**`Path to cython`**

- Example configuration for a Python build of FUN3D:

```
../configure --prefix=`pwd` \
  --with-mpi=/path/to/mpi \
  --with-parmetis=/path/to/parmetis \
  --enable-python \
  --enable-shared \
  PYTHON_CONFIG=/usr/local/pkgs-modules/Python_3.9.5/bin/python3-config \
  PYTHON=/usr/local/pkgs-modules/Python_3.9.5/bin/python \
  CYTHON=/usr/local/pkgs-modules/Python_3.9.5/bin/cython
```

Common Issues Compiling Python

1. Mismatch of mpi for mpi4py and FUN3D
   - Check the mpi backend of your mpi4py build with this command:
     - `python -c "import mpi4py; import os; a = mpi4py.__path__[0]; print(a); os.system(f'cat {a}/mpi.cfg')"`
   - If the output of this does not match the mpi your FUN3D installation was configured with, either reinstall mpi4py or reconfigure and reinstall FUN3D to use your mpi4py's mpi

2. Mismatch of python-config (or cython) with python
   - Sometimes you'll have python3 in your path, but the first python-config found is a system default path (often a python 2 version)
   - Use the **`PYTHON_CONFIG`** environment variable to specific the proper python-config

Post Installation Setup and Running

Post installation setup:

- The python module is installed in **`[prefix]/lib/python[version]/site-packages/fun3d`**
- Add [prefix]/lib/python{version}/site-packages to your PYTHONPATH

```
export PYTHONPATH=[prefix]/lib/python[version]/site-packages:${PYTHONPATH}
```

Running a python script with FUN3D:

- Set up a **`[script].py`** that calls the FUN3D API
- FUN3D python simulations do not use **`nodet_mpi`**:

```
mpiexec python [script].py
```

- Apart from command line options, inputs and outputs for FUN3D python drivers are identical to the normal executable
- Sometimes python jobs hang with a runtime failure occurs at the python level.
  - Keep an eye on your python jobs as you debug the script

Python Interfaces – Basic Flow Solve

```
from mpi4py import MPI
from fun3d.solvers import Flow
```

1. Instantiate the FUN3D flow solver

```
flow = Flow()
```

2. Initialize the solver
   - Read the mesh and namelist, allocate and initialize the solutions, etc.

```
flow.initialize(comm=MPI.COMM_WORLD)
```

3. Loop over the iterations
   - Call coupling interfaces in this loop
   - Do not iterate more than the number of steps in **`fun3d.nml`**

```
for _ in range(200):
    flow.iterate()
```

4. Call the post function
   - Write the restart file, final visualization

```
flow.post()
```

Python Interfaces - Internal Modal Solver (1/2)

- **`moddfl = 6`** in **`&aeroelastic_modal_data`** set for modes to be controlled from python
- Use a python dictionary to set **`aeroelastic_internal`**
  - Remember FUN3D won't read command line arguments when driven by python
- Modal input and output is done with **`interface`** from the **`fun3d`** module:
  - **`from fun3d import interface`**
  - **`interface.aeroelastic_push_modal_displacement(body, mode, gdisp)`**
    - Set the modal displacement for a specific mode on this time step. Call before **`iterate()`**
    - **`body`** and **`mode`** are 1-based indices (match **`moving_body.input`** indices)
    - **`gdisp`** is a scalar. Loop over this interface function to drive multiple modes
    - Generalized velocity and acceleration internally are backed out using BDF2.
  - **`state = interface.aeroelastic_pull_modal_state(body, mode)`**
    - Get the modal state. Called after **`iterate()`**
    - **`state`** is size 4: [**`gdisp, gvel, gaccel, gforce`**]

Python Interfaces - Internal Modal Solver (2/2)

- Basics of using the modal solver interfaces:

```
from mpi4py import MPI
from fun3d.solvers import Flow
from fun3d import interface

command_line_options = {"aeroelastic_internal": True}

flow = Flow()
flow.initialize(kwargs=command_line_options, comm=MPI.COMM_WORLD)

body = 1
for step in range(nsteps):
    mode = 1
    gdisp = 0.5 * step / nsteps
    interface.aeroelastic_push_modal_displacement(body, mode, gdisp)

    flow.iterate()

    mode = 2
    state = interface.aeroelastic_pull_modal_state(body, mode)
```

Coupling to External Solvers with Python

FUN3D parts of the process:

1. Instantiate the **`Flow()`** object and call **`initialize()`**
2. Get the initial coordinates (and connectivity) out of FUN3D
3. Time loop
    1. Set **`rigid`** and/or **`deform`** motion into FUN3D
    2. Call **`iterate()`**
    3. Get nodal forces on surface
4. Call **`post()`**

- Surface motion and forces are distributed but all inputs and outputs are in the same order
    - FUN3D inputs and output on each rank of python will only be the nodes owned by that rank
    - Some ranks will not contain surface patches
    - Synchronization of halo data is performed inside FUN3D
- This mode of FUN3D is not compatible with the GPU flow solver

General Aeroelastic Interfaces - Surface Nodes

```
num_nodes = flow.extract_surface_num(body)
```

- Get the number of surface nodes on this rank

```
if num_nodes > 0:
```

- Only call the next two functions if nodes are on this rank

```
x, y, z = flow.extract_surface(num_nodes, body)
```

- Get the surface node coordinates on this rank

```
ids = flow.extract_surface_id(num_nodes, body)
```

- Global ID numbers of surface nodes on this rank

- The `run_with_connectivity.py` example shows how to get the surface connectivity if your load and displacement transfer scheme needs it

General Aeroelastic Interfaces - Surface Connectivity (1/2)

The connectivity interfaces return the node ids that make up the triangular and quadrilateral surface elements

- Only the owned faces are included for each rank
- The node ids are the global ids from the **volume** mesh (match the ids in the original mesh file)
- One of the tutorials shows how to create connectivity based on surface node numbering if necessary

```
num_tris = flow.extract_tri_face_num(body=ibody)
if num_tris > 0:
    tri_conn = flow.extract_tri_face_connectivity(num_tris, body)
```

```
num_quads = flow.extract_quad_face_num(body)
if num_quads > 0:
    quad_conn = flow.extract_quad_face_connectivity(num_quads, body)
```

General Aeroelastic Interfaces - Surface Connectivity (2/2)

In version 14.0, there is a bug in the surface connectivity interfaces that will be corrected future versions:

In fun3d/Python/extension/fun3d/_flow/funtofem_wrapper.pyx make the following changes:

```
def extract_tri_face_connectivity(int nfaces, int body):
    cdef np.ndarray conn = np.zeros([nfaces,3], dtype=~~int~~ np.int32, order="F")
    extract_tri_face_connectivity_f2f(nfaces, <int *> conn.data, body)
    return conn

def extract_quad_face_connectivity(int nfaces, int body):
    cdef np.ndarray conn = np.zeros([nfaces,4], dtype=~~int~~ np.int32, order="F")
    extract_quad_face_connectivity_f2f(nfaces, <int *> conn.data, body)
    return conn
```

General Aeroelastic Interfaces - Motion and Forces

```
if "deform" in mesh_movement:
    if num_nodes > 0:
        flow.input_deformation(dx, dy, dz, body)
```

- Set x, y, z displacements for the surface nodes on this rank

```
if "rigid" in mesh_movement:
    transform = np.eye(4, order="F") # example matrix
    flow.input_rigid_transform(transform, body)
```

- Set the standard 4x4 transformation matrices for rigid motion. See section 7.3 of the manual

```
if num_nodes > 0:
    fx, fy, fz = flow.extract_forces(num_nodes, body)
```

- Get arrays of x, y, z force coefficients for the surface nodes on this rank
- Multiply by the dynamic pressure, q_∞, to get the dimensional loads

Python Thermal Interfaces

```
if num_nodes > 0:
    flow.input_wall_temperature(temperature, body)
```

- Set the wall temperature distribution for the surface nodes on this rank
- **`temperature`** should be normalized: T/T_{ref}

```
if num_nodes > 0:
    cqx, cqy, cqz = flow.extract_heat_flux(num_nodes, body)
```

- Get the nondimensional area-weighted heat flux at the surface nodes
  - Integrated over the surface faces as normal to the surface then distributed to the nodes of the face
- Multiply by the reference power $\frac{1}{2}\,\rho_{ref}\,U_{ref}^3$ to get dimensional heat flow (Watts for SI)

- Thermal coupling only supported with viscous wall boundaries (4000)

Python Coupling Examples (1/2)

- The tutorial tarball for this session contains examples to show different aspects of using the python interfaces
  - **`external_motion`** subdirectory:
    - **`run_basic_motion.py`** – demonstrates the basics of driving the motion and getting forces for aeroelastic problems
    - **`run_collect_on_root.py`** – extends the **`run_basic_motion.py`** to show how to couple to solvers that need the surface data on a single rank
    - **`run_with_connectivity.py`** – extends the **`run_basic_motion.py`** to show how to extract the surface connectivity
    - **`run_with_fsi_subiterations.py`** – extends the **`run_basic_motion.py`** to show how to perform fluid-structure interaction subiterations
  - **`external_motion_two_bodies`** subdirectory:
    - **`run.py`** – external motion with multiple bodies in the same problem
  - **`thermal`** subdirectory:
    - **`run.py`** – using the thermal coupling interfaces

Python Coupling Examples (2/2)

- `internal_aeroelastic` subdirectory:
  - `run.py` – demonstrates the basics of using the interfaces for internal modal solver interactions
- `bc_interface` subdirectory:
  - `run.py` – demonstrates how to drive a 7011 (subsonic inflow) boundary

- `mpiexec python [script].py` to run any of the examples
- The examples have detailed comments in the python scripts
- If you have questions about any of the python interfaces or examples:
  - Email fun3d-users@lists.nasa.gov for questions that can be discussed among the FUN3D community
  - Email fun3d-support@lists.nasa.gov for questions involving proprietary matters

What We Learned

- Updates to FUN3D modal aeroelastic capabilities since version 13.4
  - Modal mesh deformation
  - GPU simulations with the modal solver
  - Aeroelastic generic gas simulations
- How to use the FUN3D Python interfaces for coupled or externally driven analysis
  - Interacting with internal modal solver
  - Coupling to external structural solvers