Computational Aircraft Prototype Syntheses (CAPS) Training Session 1

John F. Dannenhoffer, III
Syracuse University

Bob Haimes
Massachusetts Institute of Technology
Introductions

- Dr. John Dannenhoffer
  - Syracuse University, jfdannen@syr.edu
- Bob Haimes
  - Massachusetts Institute of Technology, haimes@mit.edu
- Dr. Marshall Galbraith
  - Massachusetts Institute of Technology, galbramc@mit.edu
- Dr. Dean Bryson
  - Air Force Research Lab, dean.bryson@us.af.mil
- Dr. Nitin Bhagat
  - U. of Dayton Research Institute, nbhagat1@udayton.edu
- Dr. Ryan Durscher
  - Air Force Research Lab, ryan.durscher.1@us.af.mil
Overview

- CAPS Overview
  - mission
  - basic approach
  - system architecture

- CAPS Workflow

- Analysis of Simple Wing
  - basic assumptions (orientation, ...)
  - required Bodys
  - required attributes (naming vs. meta-data)
  - dissection of wing1.csm

- Aerodynamic Analyses
Muddy Cards

- Opportunity to provide immediate “feedback”
- Any questions about presentation material, critique of sample problems, . . .
- Questions will be answered at next session
CAPS Objectives

- To establish a geometry-centered design and analysis system
- To enable collaboration between conceptual design, multidisciplinary optimization, and high fidelity simulation efforts
- To be flexible enough to handle almost any aerospace configuration
CAPS Design Principles

- A user should only have to define a configuration once
  - should be driven by user-defined Design Parameters
  - should be attributed (tagged) during the construction process
- Must support multi-fidelity and multi-disciplinary analyses
  - Analysis tools should not have to be modified
  - The system must be expandable so that new analyses can be added to the system at any time
- CAPS should handle mechanics of transforming the user’s configuration into a “view” of the configuration that is needed by the analysis tool
- System execution must be flexible enough to support nearly any design or analysis task
Computational Aircraft Prototype Syntheses (CAPS)

- **Purpose**
  - Act as central repository and coordinator of all design information

- **Functionality**
  - Change a Design Parameter value (or values) and regenerate the geometry
  - Get geometric sensitivities with respect to the Design Parameters
  - Mesh (or setup the input for meshing) the geometry, specifically for the analysis at-hand
  - Setup for the execution of the specific analysis code (via an AIM)
Engineering Sketch Pad (ESP)

- Purpose
  - Provide geometric representations

- Functionality
  - User defines configuration with associated attributes (tags)
  - System creates “view” of configuration needed by each analysis
Analysis Interface Module (AIM)

- **Purpose**
  - Interface to integrate meshing and analysis tools into ESP
  - Automate high burden, error prone, repetitive touch labor tasks

- **Functionality**
  - Merges attributed geometry and external domain specific inputs
  - Produces input “file” for domain specific tools or mesh generators
  - Parses output data from domain specific tools
  - Exposes an interface for multi-disciplinary communication with other AIM’s
Logical collection of CAPS API sequences in single function calls

Enables rapid generation of CAPS problems in Python

```python
# Initiate a CAPS Problem
myProblem = capsProblem(libDir = "/ESP/Root/lib/dir")

# Load ESP input file describing the attributed geometry
myProblem.loadCAPS("./csmData/myGlider.csm")

# Load specific AIM
myProblem.loadAIM(aim = "avlAIM", analysisDir = "avl_Aviation_Test")

# Define the analysis conditions for a specific AIM
myProblem.analysis["avlAIM"].setAnalysisVal("Mach", 0.0667)

# Create the AVL Input files required for analysis
myProblem.analysis["avlAIM"].aimPreAnalysis()

# Run AVL ...

# Run AIM post-analysis to retrieve the output data
myProblem.analysis["avlAIM"].aimPostAnalysis()

# Report desired data back from the analysis
```
Target Configurations

- **wing1**
  - isolated wing: outer mold line (OML) only

- **wing2**
  - isolated wing: OML and controls

- **wing3**
  - isolated wing: OML and structure

- **wing4**
  - isolated wing: OML, structure, and controls

- **transport**
  - simple transport: OML, structure, and controls
Isolated Wing: Outer Mold Line (OML) Only
Isolated Wing: OML, Structures, and Controls
transport.csm

Isolated Transport: OML, Structures, and Controls
CAPS Workflow

- Define design parameters
- Select "possible" analyses (views) to be supported
- Build/examine the conceptual configuration (.csm file)
- Examine the views associated with each of the AIMs
- Write pyCAPS script to execute analyses
## Design Parameters for wing1

File can be found at `$ESP_ROOT/training/CAPS/data`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wing:area</td>
<td>10.0</td>
<td>wing area</td>
</tr>
<tr>
<td>wing:aspect</td>
<td>6.00</td>
<td>aspect ratio</td>
</tr>
<tr>
<td>wing:taper</td>
<td>0.60</td>
<td>taper ratio</td>
</tr>
<tr>
<td>wing:sweep</td>
<td>20.0</td>
<td>deg (of leading edge)</td>
</tr>
<tr>
<td>wing:thick</td>
<td>0.12</td>
<td>thickness ratio, frac of local chord</td>
</tr>
<tr>
<td>wing:camber</td>
<td>0.04</td>
<td>maximum camber, frac of local chord</td>
</tr>
<tr>
<td>wing:washout</td>
<td>5.00</td>
<td>deg (down at tip)</td>
</tr>
<tr>
<td>wing:dihedral</td>
<td>4.00</td>
<td>deg</td>
</tr>
</tbody>
</table>
“Possible” Analyses (Views) for wing1

- VIEW:Concept — conceptual design
- VIEW:Avl — Athena Vortex Lattice
- VIEW:SansLIP — Linearized full potential
- VIEW:Su2Inviscid — inviscid CFD analysis
- VIEW:Su2Viscous — viscous CFD analysis
Basic Assumptions

- Configuration files define the necessary Bodys.
- Bodys are oriented such that:
  - $x$ points out the tail
  - $y$ points out the right wing
  - $z$ points up
Required Bodys (for Aerodynamic Analyses)

- Outer Mold Lines (OMLs) for each component
  - FuseOml (a SolidBody)
  - WingOml (a SolidBody)
  - HtailOml (a SolidBody)
  - VtailOml (a SolidBody)
Required Attributes on WingOml

- **Body**
  - tagComp with value $leftWing or $riteWing

- **Faces**
  - tagComp with value $leftWing or $riteWing
  - tagType with value $root, $tip, $upper, $lower, or $trailingEdge

- **Edges**
  - tagType with value $root, $leadingEdge (with supporting tagComp), or $trailingEdge (with supporting tagComp)
Dissection of `wing1.csm`

File can be found at `$ESP_ROOT/training/CAPS/data`

- Definition of VIEWs to be supported
- Definition of COM Ponents that are defined
- Definition of Design Parameters
- Call to capsHeader
- Construction of WingOml (with attributes)
- Call to capsViews
Dissection of template_avl.py

File can be found at $ESP_ROOT/training/CAPS/data

- Import pyCAPS class
- Initialize capsProblem object
- Load geometry (.csm) file
- Set geometry variables to enable Avl mode
- Set other geometry specific variables
- Load AVL AIM
- Set analysis specific variables
- Run AIM pre-analysis
- Run AVL (via system call)
- Run AIM post-analysis
Start with configuration file (`wing1.csm`)
Start with `template_avl.py` script for AVL
Change directory into `$ESP_ROOT/training/CAPS/data`
Run: `python template_avl.py wing1.csm`
Automatically-created View of wing1 for AVL
Second Aerodynamic Analysis

Drag polar using AVL on wing1

- Start with configuration file (wing1.csm)
- Make a copy of the original template_avl.py script (and call it exercise_1_1.py)
- Modify exercise_1_1.py to create a lift curve ($C_L$ vs $\alpha$) and a drag polar ($C_D$ vs $C_L$)
  - you will have to import numpy and matplotlib
  - you will have to add a loop to run the various analysis
  - you will have to generate the plots (next page)
- Run: python exercise_1_1.py wing1
Second Aerodynamic Analysis

Drag polar using AVL on wing1
Third Aerodynamic Analysis

Design sweep using AVL on wing1

- Start with configuration file (wing1.csm)
- Make a copy of the original template_avl.py script (and call it exercise_1_2.py)
- Modify exercise_1_2.py to plot the spanwise efficiency factor (e) vs. taper ratio, for various sweep angles (next page)
- Run python exercise_1_2.py wing1
Third Aerodynamic Analysis

Design sweep using AVL on wing1
Start with SAME configuration file (wing1.csm)

Start with template_sansLIP.py script for sansLIP
- enable VIEW:SansLIP instead of VIEW:AVL
- load avlAIM instead of sansLIPAIM
- set sansLIP-specific meshing parameters

Run: python template_sansLIP.py wing1.csm
Automatically-created View of wing1 for sansLIP
Start with SAME configuration file (wing1.csm)

Start with `template_su2inviscid.py` script for SU2 in inviscid mode
- similar changes to above
- SU2 also requires generation of a volume mesh (via `tetgen` by default)

Run `python template_su2inviscid.py wing1.csm`
Automatically-created View of wing1 for SU2
Muddy Cards

- Opportunity to provide immediate “feedback”
- Any questions about presentation material, critique of sample problems, . . .
- Questions will be answered at next session