Awave Analysis Interface Module (AIM) Manual

Ed Alyanak and Ryan Durscher
AFRL/RQVC

August 18, 2020

Contents

1 Introduction **1**

1.1 Awave AIM Overview 1

1.2 Awave Modifications 1

1.3 Examples 2

2 AIM Attributes **2**

3 AIM Inputs **2**

4 AIM Outputs **2**

5 Awave AIM Examples **3**

5.1 Prerequisites 3

5.1.1 Script files 3

5.2 Creating Geometry using ESP 3

5.3 Performing analysis using pyCAPS 5

Bibliography **7**

1 Introduction

1.1 Awave AIM Overview

Awave provides an estimation for wave drag at supersonic Mach numbers at various angles of attack. Taken from the Awave manual [1] :

"Awave is a streamlined, modified version of the Harris far-field wave drag program described in the reference. It has all of the capabilities and accuracy of the original program plus the ability to include the approximate effects of angle of attack. It is an order of magnitude faster, and improvements to the integration schemes have reduced numerical integration errors by an order of magnitude. A formatted input echo has been added so that those not intimately familiar with the code can tell what has been input.

Reference: Harris, Roy V., Jr. An Analysis and Correlation of Aircraft Wave Drag. NASA TMX-947. March 1964. "

An outline of the AIM's inputs, outputs and attributes are provided in AIM Inputs and AIM Outputs and AIM Attributes, respectively.

Upon running preAnalysis the AIM generates a single file, "awaveInput.txt" which contains the input information and control sequence for Awave to execute. An example execution for Awave looks like (Linux and OSX executable being used - see Awave Modifications):

```
awave awaveInput.txt
```

1.2 Awave Modifications

The AIM assumes that a modified version of Awave is being used. The modified version allows for longer input and output file name lengths, as well as other I/O modifications. This modified version of Awave, awavemod.f, is not currently supplied with the AIM due to licensing issues, please contact the CAPS creators for additional details. Once this source code is obtained, it is automatically built with the AIM. During compilation, the source code is compiled into an executable with the name *awave* (Linux and OSX) or *awave.exe* (Windows)

1.3 Examples

An example problem using the Awave AIM may be found at Awave AIM Examples.

2 AIM Attributes

The following list of attributes drives the Awave geometric definition. Aircraft components are defined as cross sections in the low fidelity geometry definition. To be able to logically group the cross sections into wings, tails, fuselage, etc. they must be given a grouping attribute. This attribute defines a logical group along with identifying a set of cross sections as a lifting surface or a body of revolution. The format is as follows.

- **capsType** This string attribute labels the *FaceBody* as to which type the section is assigned. This information is also used to logically group sections together to create wings, tails, stores, etc. Because Awave is relatively rigid **capsType** attributes must be on of the following items:

  Lifting Surfaces: Wing, Tail, HTail, VTail, Cannard, Fin

  Body of Revolution: Fuselage, Fuse, Store

- **capsGroup** This string attribute is used to group like components together. This is a user defined unique string that can be used to tie sections to one another. Examples are tail1, tail2, etc.

- **capsReferenceArea** [Optional: Default 1.0] This attribute may exist on any *Body*. Its value will be used as the SREF entry in the Awave input.

3 AIM Inputs

The following list outlines the Awave inputs along with their default value available through the AIM interface. All inputs to the Awave AIM are variable length arrays. **All inputs must be the same length**.

- **Mach = double**
  OR
- **Mach = [double, ... , double]**
  Mach number.
- **Alpha = double**
  OR
- **Alpha = [double, ... , double]**
  Angle of attack [degree].

4 AIM Outputs

The main output for Awave is CDwave. This reports wave drag coefficient with respect to the AIM Inputs given. In addition, an echo of the Mach number and angle of attack inputs is provided. This allows the user to ensure that the CDwave value matches the expected Mach, AoA input pair. If a given pair does not execute then it will not appear in the results. Thus, it is always good practice to do a sanity check using the echo of input values.

- **CDwave =** Wave Drag Coefficient.
- **MachOut =** Mach number.
- **Alpha =** Angle of attack (degree).

5 Awave AIM Examples

This is a walkthrough for using Awave AIM to analyze a wing, tail, fuselage configuration.

5.1 Prerequisites

It is presumed that ESP and CAPS have been already installed, as well as Awave. Furthermore, a user should have knowledge on the generation of parametric geometry in Engineering Sketch Pad (ESP) before attempting to integrate with any AIM. Specifically this example makes use of Design Parameters, Set Parameters, User Defined Primitive (UDP) and attributes in ESP.

5.1.1 Script files

Two scripts are used for this illustration:

1. awaveWingTailFuselage.csm: Creates geometry, as described in the following section.
2. awave_PyTest.py: pyCAPS script for performing analysis, as described in Performing analysis using pyCAPS.

5.2 Creating Geometry using ESP

First step is to define the analysis intention that the geometry is intended support.

```
attribute capsAIM $awaveAIM
```

Next we will define the design parameters to define the wing cross section and planform.

```
despmtr   thick     0.12      frac of local chord
despmtr   camber    0.04      frac of local chord
despmtr   tlen      5.00      length from wing LE to Tail LE
despmtr   toff      0.5       tail offset

despmtr   area      10.0
despmtr   aspect    6.00
despmtr   taper     0.60
despmtr   sweep     20.0      deg (of c/4)

despmtr   washout  -5.00      deg (down at tip)
despmtr   dihedral  4.00      deg
```

The design parameters will then be used to set parameters for use internally to create geometry.

```
set       span      sqrt(aspect*area)
set       croot     2*area/span/(1+taper)
set       ctip      croot*taper
set       dxtip     (croot-ctip)/4+span/2*tand(sweep)
set       dztip     span/2*tand(dihedral)
```

Next the Wing, Vertical and Horizontal tails are created using the *naca* User Defined Primitive (UDP). The inputs used for this example to the UDP are Thickness and Camber. The naca sections generated are in the X-Y plane and are rotated to the X-Z plane. They are then translated to the appropriate position based on the design and set parameters defined above. Finally reference area can be given to the Awave AIM by using the **capsReferenceArea** attribute. If this attribute exists on any body that value is used otherwise the default is 1.0.

In addition, each section has a **capsType** attribute. This is used to define the type of surface being create into a lifting surface or a body. The other attribute found on the first wing section is **capsGroup**. This is used to logically group cross section of a give **capsType** type together. More information on this can be found in the AIM Attributes section.

```
# right tip
udprim    naca      Thickness thick     Camber    camber
attribute capsReferenceArea area
attribute capsType   $Wing
attribute capsGroup    $Wing
scale     ctip
rotatex   90        0         0
rotatey   washout   0         ctip/4
translate dxtip    -span/2    dztip

# root
udprim    naca      Thickness thick     Camber    camber
attribute capsType   $Wing
attribute capsGroup    $Wing
rotatex   90        0         0
scale     croot

# left tip
udprim    naca      Thickness thick     Camber    camber
attribute capsType  $Wing
attribute capsGroup    $Wing
scale     ctip
rotatex   90        0         0
rotatey   washout   0         ctip/4
translate dxtip     span/2    dztip
```

Vertical Tail definition

```
# tip
udprim    naca      Thickness thick
attribute capsType  $VTail
attribute capsGroup   $VertTail
scale     0.75*ctip
translate tlen+0.75*(croot-ctip) 0.0 ctip+toff

# base
udprim    naca      Thickness thick
attribute capsType  $VTail
attribute capsGroup   $VertTail
scale     0.75*croot
translate tlen 0.0 toff
```

Horizontal Tail definition

```
# tip left
udprim    naca      Thickness thick
attribute capsType  $HTail
attribute capsGroup   $Stab
scale     0.75*ctip
rotatex   90        0         0
translate tlen+0.75*(croot-ctip) -ctip toff

# tip left
udprim    naca      Thickness thick
attribute capsType  $HTail
attribute capsGroup   $Stab
scale     0.75*ctip
rotatex   90        0         0
translate tlen+0.75*(croot-ctip) 0.0 toff

# tip right
udprim    naca      Thickness thick
attribute capsType  $HTail
attribute capsGroup   $Stab
scale     0.75*ctip
rotatex   90        0         0
translate tlen+0.75*(croot-ctip) ctip toff
```

Fuselage definition. Notice the use of the *ellipse* UDP. In this case, only translation is required to move the cross section into the desired location.

```
skbeg -0.4*tlen 0.0 0.0
skend
attribute capsType $Fuse
attribute capsGroup  $Fuselage

udprim  ellipse ry 0.5*croot rz 0.2*croot
attribute capsType $Fuse
attribute capsGroup  $Fuselage
translate 0.0 0.0 0.0
```

```
udprim  ellipse ry 0.4*croot rz 0.1*croot
attribute capsType $Fuse
translate croot 0.0 0.0

udprim  ellipse ry 0.1*croot rz 0.1*croot
attribute capsType $Fuse
attribute capsGroup  $Fuselage
translate tlen 0.0 toff

udprim  ellipse ry 0.01*croot rz 0.01*croot
attribute capsType $Fuse
attribute capsGroup  $Fuselage
translate tlen+0.75*croot 0.0 toff
```

Store definition. This addition is to demonstrate the addition of a wing tip store in the Awave representation.

```
udprim  ellipse ry 0.1*ctip rz 0.1*ctip
attribute capsType $Store
attribute capsGroup  $RightWingTank
translate dxtip    -span/2    dztip

udprim  ellipse ry 0.1*ctip rz 0.1*ctip
attribute capsType $Store
attribute capsGroup  $RightWingTank
translate dxtip+ctip    -span/2    dztip

udprim  ellipse ry 0.1*ctip rz 0.1*ctip
attribute capsType $Store
attribute capsGroup  $LeftWingTank
translate dxtip    span/2    dztip

udprim  ellipse ry 0.1*ctip rz 0.1*ctip
attribute capsType $Store
attribute capsGroup  $LeftWingTank
translate dxtip+ctip    span/2    dztip
```

5.3 Performing analysis using pyCAPS

An example pyCAPS script that uses the above ∗.csm file to run Awave is as follows.

First the pyCAPS and os module needs to be imported.

```
# Import capsProblem from pyCAPS
from pyCAPS import capsProblem

# Import os module
import os
```

Once the modules have been loaded the problem needs to be initiated.

```
myProblem = capsProblem()
```

Next local variables used throughout the script are defined.

```
workDir = "AwaveAnalysisTest"
```

Next the ∗.csm file is loaded and design parameter is changed - area in the geometry. Any despmtr from the awaveWingTailFuselage.csm file is available inside the pyCAPS script. They are: thick, camber, area, aspect, taper, sweep, washout, dihedral...

```
geometryScript = os.path.join("..","csmData","awaveWingTailFuselage.csm")
myGeometry = myProblem.loadCAPS(geometryScript, verbosity=args.verbosity)
myGeometry.setGeometryVal("area", 10.0)
```

The Awave AIM is then loaded with:

```
myAnalysis = myProblem.loadAIM( aim = "awaveAIM",
                                analysisDir = workDir )
```

After the AIM is loaded the Mach number and angle of attack (Alpha) are set as aimInputsAwave. The Awave AIM supports variable length inputs. For example 1, 10 or more Mach and AoA pairs can be entered. The example below shows two inputs. The length of the Mach and Alpha inputs must be the same.

```
myAnalysis.setAnalysisVal("Mach" , [ 1.2, 1.5])
myAnalysis.setAnalysisVal("Alpha", [ 0.0, 2.0])
```

Once all the inputs have been set, preAnalysis needs to be executed. During this operation, all the necessary files to run Awave are generated and placed in the analysis working directory (analysisDir)

```
myAnalysis.preAnalysis()
```

At this point the required files necessary run Awave should have been created and placed in the specified analysis working directory. Next Awave needs to executed such as through an OS system call (see Awave AIM Overview for additional details) like,

```
print ("\n\nRunning Awave......")
currentDirectory = os.getcwd() # Get our current working directory

os.chdir(myAnalysis.analysisDir) # Move into test directory
os.system("awave awaveInput.txt > Info.out"); # Run Awave via system call

os.chdir(currentDirectory) # Move back to top directory
```

A call to postAnalysis is then made to check to see if Awave executed successfully and the expected files were generated.

```
myAnalysis.postAnalysis()
```

Similar to the AIM inputs, after the execution of Awave and postAnalysis any of the AIM's output variables (AIM Outputs) are readily available; for example,

```
CdWave = myAnalysis.getAnalysisOutVal("CDwave");
```

Printing the above variable results in,

```
CdWave =  [0.484423786, 0.0935611948]
```

References

[1] L. A. McCullers. *AWAVE: User's Guide for the Revised Wave Drag Analysis Program*, Apr. 1992. 1