JSBSim Script
Tutorial 1

Bill Galbraith
Holy Cows, Inc.
December 2010
Rev 1.1

1 Purpose

The purpose of this report is to demonstrate scripting capabilities in JSBSim through example. Various scripting features are shown as scripts are built up from the simplest to more complex.

An aircraft model of a representative business-class turbojet aircraft was built using the Aeromatic utility found on the JSBSim web page (http://jsbsim.sourceforge.net/). An existing engine, the J85-GE-5 engine was used for propulsion. The Aeromatic program does an amazing job of building a stable aero model from just a few inputs. It generates all the file structure required for JSBSim, so it's a great place to start.

2 Environment Set-up

For this tutorial, we will assume that you are running on a Windows-based computer. The sections below will show you how to install **JSBSim** and **gnuplot**.

2.1 JSBSim installation

If you do not have JSBSim installed on your hard drive yet, you have two installation options. If you already have JSBSim installed, you can skip forward to the next section.

2.1.1 Install JSBSim Distribution Package

At the time of this writing (December 2010), JSBSim does not offer a precompiled version for Windows, nicely packaged into a ZIP file. There is one available here. This is a minimal directory structure of JSBSim, with the then-current version of JSBSim compiled for Windows. The source might not be the latest and greatest, bleeding-edge code, but it is sufficient to demonstrate the functions and features detailed in this tutorial, and gets you running quicker, without having to install multiple packages and have to compile the program yourself.

NOTE: This JSBSim distribution package has been SEVERELY reduced, to include just the barest minimal set of files necessary to run this tutorial. There are many aircraft and engines available in the JSBSim source package that have been removed from this distribution package to keep the size down.

2.1.2 Install and Compile JSBSim from source

If you wish to have the entire set aircraft and engines, as well as the latest source code, you can download it from CVS yourself. There are instructions on how to do this in the **JSBSim Reference Manual**, available at http://jsbsim.sourceforge.net. That manual also has instructions on building the program under Linux or Cygwin. To build the program under Windows using Visual C++ 2008 Express (a free C++ compiler available for Windows), consult the instructions here.

2.2 Installing gnuplot

gnuplot is a very flexible plotting routine. We are going to use it here just because it is free, and does what we need it to do. Download the installation package from the **gnuplot** website http://sourceforge.net/projects/gnuplot/files/, where you can download the latest gnuplot Zip file. At the time of this writing, the latest version was 4.4.2, so you want the file ***gn442win32.zip***.

Unzip the package, and move the **gnuplot** directory some place that you like. I liked it under the **c:\Program Files** directory on my Windows XP machines.

Once you have copied the **gnuplot** directories to a set place, you need to add the binary directory to the **PATH** environment variable. Press and hold the *Windows key*, and press the *break* key. Select the advance tab, and click on **Environment Variables**. Under

System variables, find the **Path** variable and add to it the path to the gnuplot binary directory. For my system, I added ;***c:\Program Files\gnuplot\binary***; at the end of the command line. Separate each item with a semicolon.

To test your installation, open a command line windows (Windows key-R, cmd <enter>), navigate to the ***gnuplot\demo*** directory (c:\Program Files\gnuplot\demo) and try one of the demo scripts, such as:

```
gnuplot textrotate.dem
```

This should produce a pretty picture on your screen.

3 Scripts

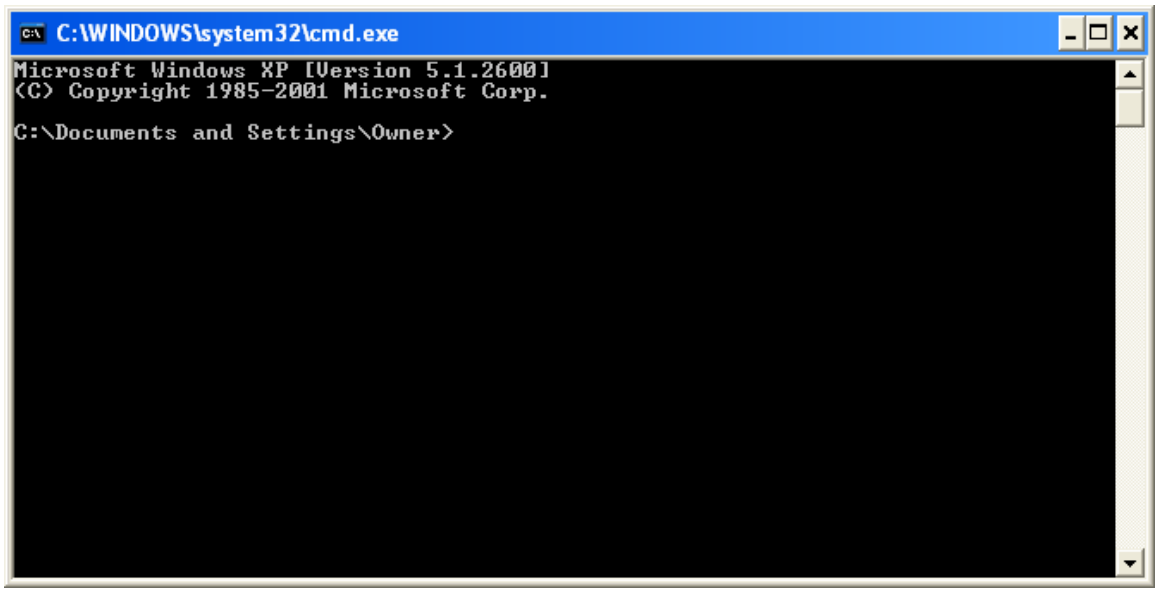
The scripts produced for this tutorial are typical of the type of tests required for aircraft simulator certification, either military or commercial. Not all test types are shown, but the fundamentals of building test scripts are shown.

3.1 Test Execution

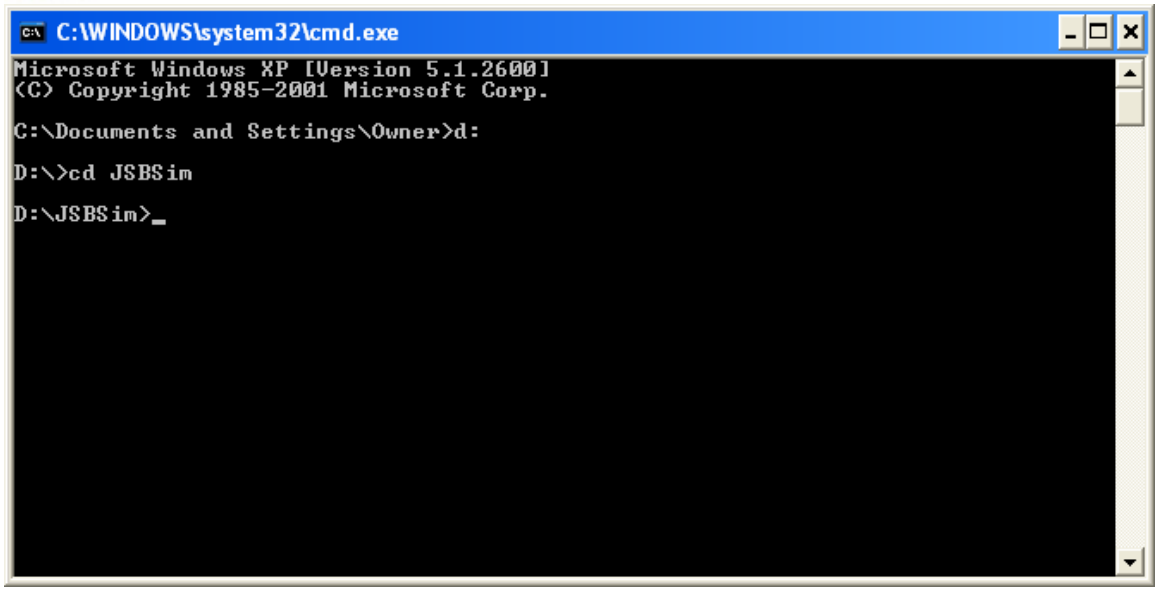
The tests described below can be run with the JSBSim program. For this tutorial, we will assume that you are running on a Windows-based computer. I used Windows XP during this writing, but commands are similar for Cygwin and Linux. Just substitue "/" for "\".

3.1.1 Running JSBSim

Open a command window. There are multiple ways to do this, but the quickest is pressing the Windows key and "R" to open up the **Run** windows, and typing in "cmd" (without the quotes). Your window should look something like this:



Navigate to the JSBSim directory. In my case, I have it installed on the D: drive, so I'll go to that drive and the correct directory as shown:



From the command line in the JSBSim directory, the following command line executes the JSBSim program:

```
Debug\JSBSim
```

Note that all commands to be typed into the command windows will be shown in a box in this tutorial.

This should spit out some usage information and exit the program, since we didn't specify any scripts to run. This message shows that we can specify a script file on the command line using the ***–script*** option. We can also specify the output log file on the command line with the ***–outputlogfile*** option. We will use both of these options below.

If you did not get this message, go back and figure out where you went wrong with the installation. In order to see the entire output, you will either have to scroll the CMD box, or maximize the window. Note that when maximized, it isn't the entire width of you screen.

3.1.2 Specific Test Execution

There are 5 tests included with this tutorial:

- trim-cruise
- trim-approach
- flap-change-up
- roll-response
- phugoid

Those tests are included in the directory ***aircraft\Tutorial\scripts***. They all end with an extension of ***.xml***.

When we run JSBSim, we have to specify on the command line what test we want to run. So, our command line will look like this:

```
Debug\JSBSim –script=aircraft\Tutorial_1\trim-cruise.xml
```

If you run this, the program will pause briefly while it runs, then return you to the command prompt. You will have a file called ***Tutorial_1.csv*** in the local directory, which contains the time history results of the test we just ran. That is described next.

3.1.3 Saving Results to a Specific File

When JSBSim executes, it would normally output the results to a file that is specified in the aircraft file (***aircraft\Tutorial_1\Tutorial_1.xml*** is the aircraft file in our case, and the output file is specified in that file as ***Tutorial_1.csv***, which gets placed in the directory where we are running JSBSim from. For this tutorial, we want to specify a unique output file, so that we can retain the outputs for plotting later. We use the ***–outputlogfile*** command line option for that.

We are also going to pipe the output to a text file that we can go back and look at if there are errors. For that, we use the redirection operator > and a file, such as **JSBSim.out**.

So, putting it all together, we end up with a command line that looks like this:

```
Debug\JSBSim –script=aircraft\Tutorial_1\trim-cruise.xml
–outputlogfile=aircraft\Tutorial_1\results\trim-cruise.csv > JSBSim.out
```

Note that this command should be on one line.

When this script is executed, the output message from JSBSim are recorded in the file ***JSBSim.out***, and data is recorded to the file ***results\Tutorial_1\results\trim-cruise.csv***, as specified. This file is a time history of parameters, separated by commas. (***CSV*** stands for **C**omma **S**eparated **V**alues) This file can be imported into Microsoft Excel easily for plotting, or you can use a program like **gnuplot**. We also pipe the output messages to a file. This is done to retain the messages from JSBSim in case there is an error.

3.1.4 Plotting the Results

Now, we want to plot the results. To generate a plot of the results, you can use the **gnuplot** program. There are plot scripts in the **aircraft/Tutorial_1/plots** directory to generate the plots. To plot to the screen we use the command:

```
gnuplot  aircraft\Tutorial_1\plots\trim-cruise.p
```

What if we want to plot to a file, so that we can include the pictures in a report, such as was done for this tutorial? To plot to a file, there are some slightly different commands required inside the ***trim-cruise.p*** file. In that file, I've shown how to output the plots to the screen, **.PNG**, or **.PDF** file. For your convenience, I've made those changes and named the file ***trim-cruise.pf***. You can generate the plots to a .PNG file with:

```
gnuplot  aircraft\Tutorial_1\plots\trim-cruise.pf
```

That command outputs the results to the file ***aircraft\Tutorial_1\results\trim-cruise.png***.

3.1.5 Putting it all together with the RUNTEST Batch File

The JSBSim command line is pretty long, as well as the plot command, and you can easily make a mistake typing that out every time, so lets build a batch file to handle all of this. If you used the JSBSim distribution package, this batch file already exists. Otherwise, in your favorite text editor such as **edit** under the command prompt, build a file in the ***JSBSim*** home directory called ***runtest.bat*** that looks like this:

RUNTEST.bat for Windows

```
rem Remove the old results file
del /Q aircraft\Tutorial_1\results\%1.csv

rem Run the test                          (note: this command is all on one line)
Debug\JSBSim --script=aircraft\Tutorial_1\scripts\%1.xml
--outputlogfile=aircraft\Tutorial_1\results\%1.csv > JSBSim.out

rem Generate gnuplot to the screen
gnuplot aircraft\Tutorial_1\plots\%1.p

rem Generate gnuplot to a PNG file
gnuplot aircraft\Tutorial_1\plots\%1.pf
```

Now, in the command window, you can just type a simple command like:

```
runtest trim-cruise
```

And it deletes the old results, runs the test, and plots out the results to the screen and to a file in the ***aircraft\Tutorial_1\results*** directory. Pretty sweet, huh?

4 General Test notes

In this section, we will examine the test scripts in detail, starting from the simplest test and building on it to come up with more sophisticated tests.

For the initial conditions of these tests, I only set the altitude and calibrated airspeed to that from my criteria data. There was no attempt made to match outside air temperature, winds or barometric pressure, nor aircraft-specific parameters such as weight, center of gravity location, or inertias, such as you would do if you were running real flight test. The default for the aircraft parameters was used.

There are 5 tests which are discussed:

- trim-cruise
- trim-approach
- flap-change-up
- roll-response
- phugoid

4.1 Trim-cruise

What this script demonstrates:

- Basic trimming at altitude and airspeed
- Initialization with engines running

This is the simplest of tests possible. All it does is trim the aircraft at an altitude and airspeed, and run the simulation for 1 second to verify trim has been obtained.

File: aircraft\Tutorial_1\scripts\trim-cruise.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
    name="Trim in Cruise configuration">

  <use aircraft="Tutorial_1" initialize="scripts/airborne"/>
  <run start="0" end="1" dt="0.00833333">

   <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
         <set name="ic/vc-kts"  value="191.0"/>
         <set name="ic/h-sl-ft" value="4779.0"/>
         <set name="simulation/do_simple_trim" value="1"/>
    </event>
  </run>

</runscript>
```

In this script, you will see a header, including xml specifications on the first 5 lines. After that, you will see a ***use*** statement. It specifies the aircraft model to use (without the .xml extension), which JSBSim knows to find in the **aircraft** directory. The subdirectory below ***aircraft*** is the same as the aircraft name, so the main aircraft file is in ***aircraft\Tutorial_1\Tutorial_1.xml***. This line also specifies the initialization file ***scripts/airborne.xml***, which JSBSim knows to look in the directory **aircraft/Tutorial_1** directory.

Let's take a quick look at the ***airborne.xml*** initialization file. I use it just for airborne initialization, with a default airspeed and altitude, and setting all the engines running. Since each script file requires an initialization file, I used the same one for every script, then set the test-specific initial conditions inside the script. You could create a separate initialization file for each script, and in there specify the altitude, airspeed and other parameters.

File: airborne.xml

```
<?xml version="1.0" encoding="utf-8"?>
<initialize name="airborne">

  <running>               -1   </running>
  <altitude unit="FT">  5000.0  </altitude>
  <vc unit="KTS">        150.0  </vc>

</initialize>
```

Okay, back to the specific test file ***trim-cruise.xml***

The next line in the script specifies the run times, such as start and end times, and the iteration rate *dt*. A delta time *dt* of 0.00833 seconds is 120 hz. So, this causes JSBSim to execute from times 0.0 to 1.0 at a rate of 120 hz, then exits.

The next block is an event. It specifies the condition for when it should run (simulation/sim-time-sec ge 0.0), and what it should do when the condition specified is met. When JSBSim executes this script, it checks the condition every frame until the condition is met. We specified that sim-time is greater or equal to 0.0, just to make sure the event is executed. When that condition is met, we override the initial condition (IC) airspeed and altitude set in the ***airborne.xml*** by setting new a IC altitude and airspeed here that are the specific conditions from the criteria data, then trim the aircraft to the initial conditions specified, without updating *sim-time-sec*. JSBSim then executes for 1 second. Since there is an output section in the **Tutorial_1.xml** file, it dumps the specified parameters to a data file, also specified in that output section of **Tutorial_1.xml**, the output file being **Tutorial_1.csv**. However, on the command line, we overrode that output name, and set it to **aircraft\Tutorial_1\results\trim-cruise.csv**.

The test results plot is shown in **Figure 4.1-1**. This is generated by running the **gnuplot** program. Note that on the top plot for **KCAS, kts**, you can barely see the green line at 191.0. It is there, just hard to see. That happens sometimes with **gnuplot**.

Something that needs to be mentioned is that **gnuplot** commands contained in the ***trim-cruise.p*** and ***trim-cruise.pf*** files use a column number to decide what information to plot. For example, in the ***trim-cruise.csv*** file, it plots column 1 versus column 74 as the **KCAS, kts** value. If you change the parameters recorded in **Tutorial_1.xml**, it could change the column order, and KCAS might not be the 74th column any more. Just a little something to keep in mind. These are reasons I don't like **gnuplot** too much.

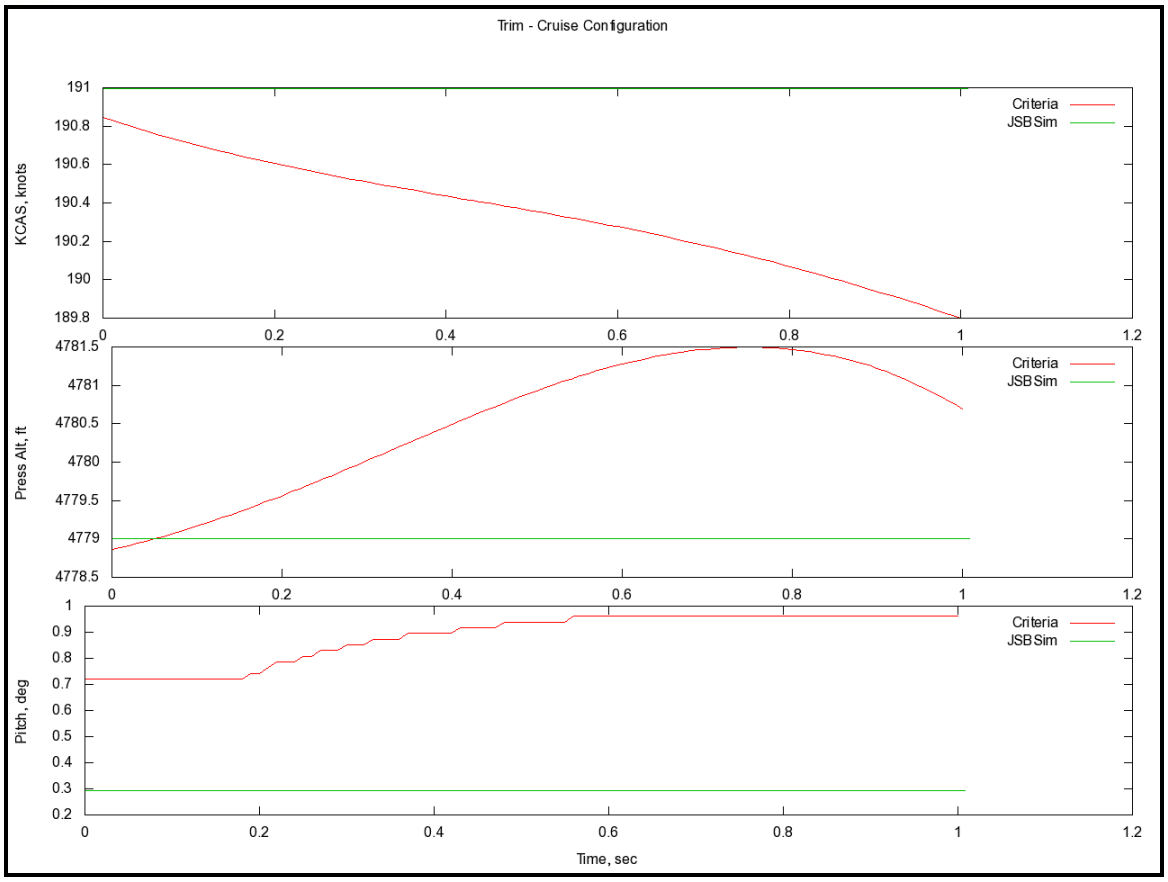


Figure 4.1-1. Test results for trim-cruise

4.2 Trim-approach

What this script demonstrates:

- Aircraft configuration for trim (Setting flaps before trimming)

The next step is to show a secondary control surface deflection, such as the flaps. In this case, we are setting the flaps at 15 degs before trimming the aircraft. In this case, we are setting the commanded flap position and the actual flap position.

There was been a lot of discussion on the JSBSim Developer's list recently (December 2010) as to the proper way to handle the initialization of something like the flaps. If you just set the commanded value and it is different than the default position, the flaps may transition during the trim, or it may step to the new value. At the time of this writing, it was not decided which way it should be, and is always subject to change. This method should cover the issue completely, but it is still something that you should be aware of.

File: aircraft\Tutorial_1\scripts\trim-appraoch.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
    name="Trim - Approach">

  <use aircraft="Tutorial_1" initialize="scripts/airborne"/>
  <run start="0" end="4" dt="0.00833333">

    <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
        <set name="ic/vc-kts"  value="113.0"/>
        <set name="ic/h-sl-ft" value="8776.0"/>
        <set name="fcs/flap-cmd-norm" value="0.5"/>
        <set name="fcs/flap-pos-deg" value="15.0"/>
        <set name="simulation/do_simple_trim" value="1"/>
    </event>

  </run>

</runscript>
```

There is nothing exciting about this test, other than showing how to trim with the flaps set to half (The total flap deflection in the **Tutorial_1.xml** file is 30 degrees). The time history results are plotted in **Figure 4.2-1**.

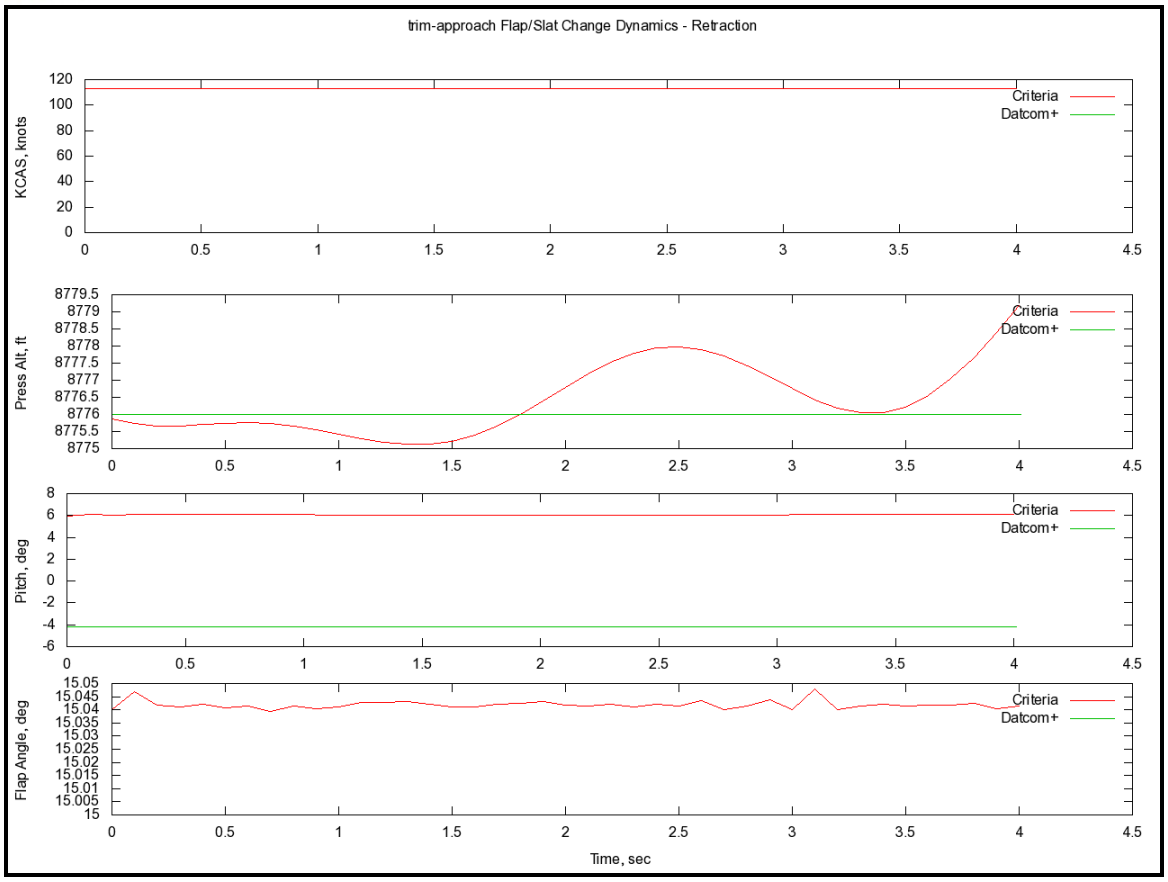


Figure 4.2-1. Test results for trim-approach

4.3 Flap-Change-up

What this script demonstrates:

- Ramped input of a control

Now that the flaps are down and we've trimmed the aircraft, let's demonstrate how to retract flaps and show the time history of the aircraft reaction. This script does that. As below, we will initialize the flaps at 15 deg (normalized value of 0.5) before trimming. A new event has been added to move the flap control up in 0.25 seconds, but the flaps retract in 1.7 seconds in keeping with the model in **Tutorial_1.xml**. Note that the **Tutorial_1.xml** file was modified from the original Aeromatic output to change the time to move the flaps from 15 deg to 0. The original time was 4 seconds. This was changed to 1.7 seconds to match the criteria data.

File: aircraft\Tutorial_1\scripts\flap-change-up.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
    name="Flap Change - up">

  <use aircraft="Tutorial_1" initialize="scripts/airborne"/>
  <run start="0" end="15" dt="0.00833333">

    <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
        <set name="ic/vc-kts"  value="118.0"/>
        <set name="ic/h-sl-ft" value="7629.0"/>
        <set name="fcs/flap-cmd-norm" value="0.5"/>
        <set name="fcs/flap-pos-deg" value="15.0"/>
        <set name="fcs/flap-cmd-norm" value="0.5"/>
        <set name="simulation/do_simple_trim" value="1"/>
    </event>

    <!—Move the flap control in 0.25 seconds. The flaps retract in 1.7 seconds -->
    <event>
      <condition> simulation/sim-time-sec ge 4.0 </condition>
      <set name="fcs/flap-cmd-norm" value="0.0" action="FG_RAMP" tc="1.7"/>
    </event>

  </run>

</runscript>
```

For the test purist, this is a closed loop or stick-fixed response, since we are not allowing the elevator to move during the test.

The time history plot of the results plotted against the criteria data is shown in **Figure 4.3-1**. Note that the JSBSim model pitches down pretty quickly, loses altitude and picks up airspeed. That's not desirable in an aircraft, but that is what happened with the JSBSim model produced by Aeromatic.

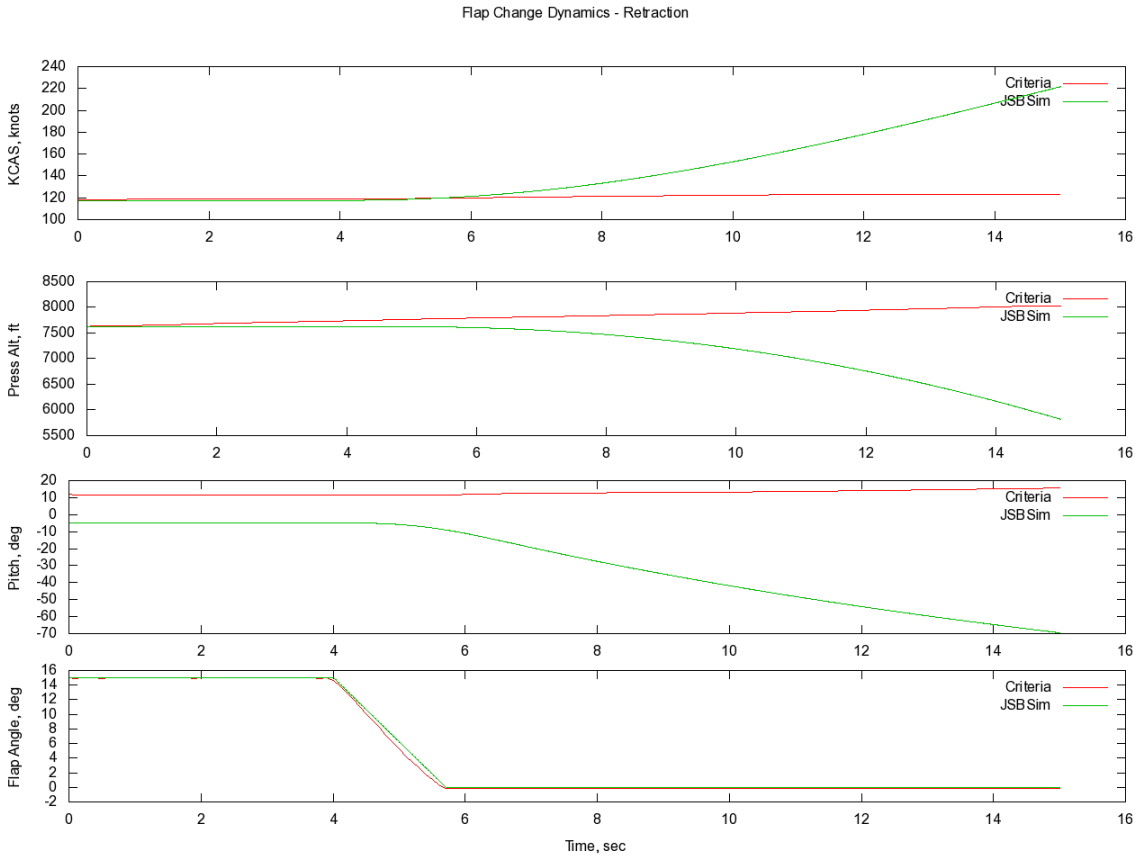


Figure 4.3-1. Test results for flap-change-up

4.4 Roll Response, Cruise Configuration

What this script demonstrates:

- Trimming with a bank angle
- LFI table for time history input of control (math function – table)
- Math function - product

The next evolutionary step was to have a test with a control input. A roll response test was chosen. This test shows two features not present in the previous test, trimming in a bank and a time history of a control input. The test was first written with just a trim to a bank angle and allowed to run. The aircraft eventually rolled out wings level as expected. Next, a simple control input was introduced which was close to the criteria data input. Note the use of the *function table* construct.

File: aircraft\Tutorial_1\scripts\roll-response.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
    name="Roll Response">

  <use aircraft="Tutorial_1" initialize="scripts/airborne"/>
  <run start="0" end="9.5" dt="0.00833333">

    <event name="Trim">
       <condition> simulation/sim-time-sec ge 0.0 </condition>
         <set name="ic/vc-kts"  value="216.0"/>
         <set name="ic/h-sl-ft" value="10592.0"/>
         <set name="ic/phi-deg" value="-42.0"/>
        <set name="simulation/do_simple_trim" value="5"/>
    </event>

    <event name="Time Notify" continuous="true">
      <description>Provide a time history input for the aileron</description>
      <condition> simulation/sim-time-sec ge 0 </condition>
      <set name="fcs/aileron-cmd-norm" >
          <function>
          <table>
            <independentVar lookup="row">simulation/sim-time-sec</independentVar>
            <tableData>
                 0.0     0.0
                 5.2     0.0
                 6.6     0.2
                10.0     0.2
            </tableData>
          </table>
        </function>
      </set>
    </event>

  </run>

</runscript>
```

This script shows how to set up a table for control input. Between the data points listed, an interpolation is performed to arrive at the desired value. The plotted results of this test are shown in **Figure 4.4-1**.

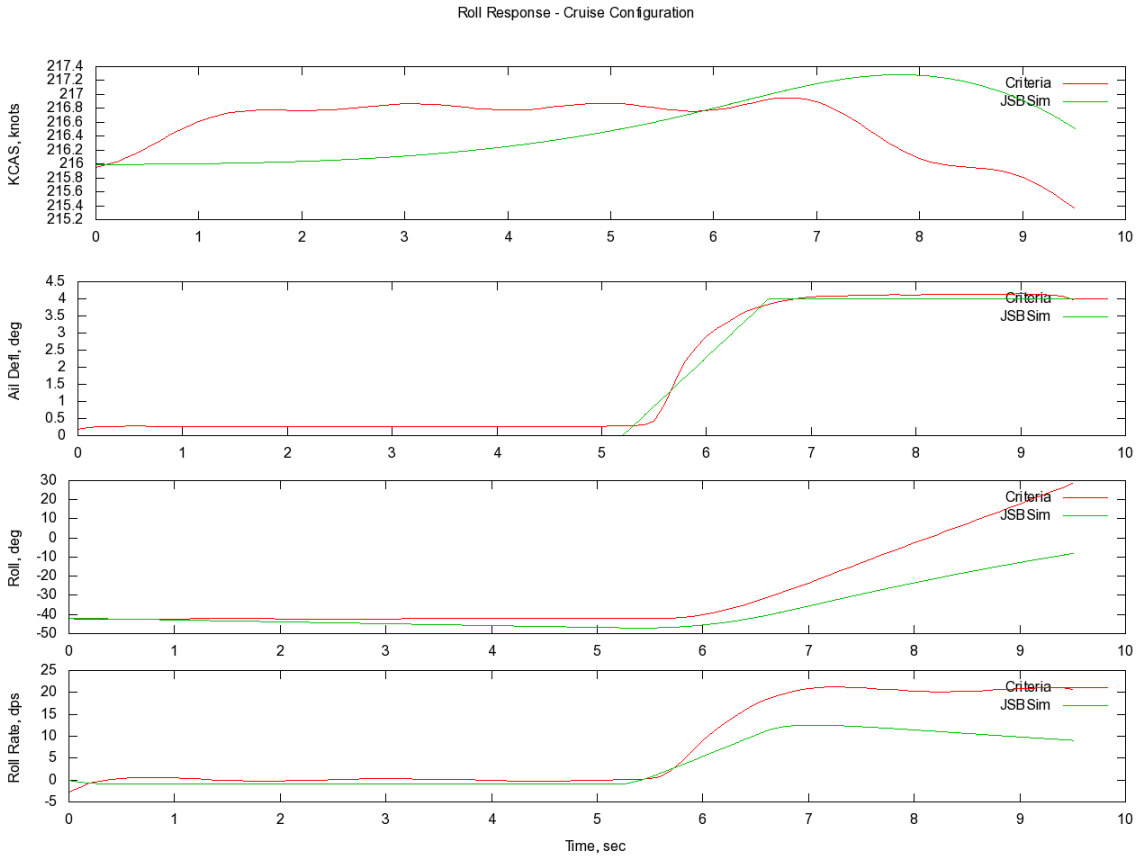


Figure 4.4-1 Test results for roll-response

4.5 Phugoid Dynamics – Cruise

What this script demonstrates:

- Declaration of local parameters
- Multiple events
- Notification within an event
- Math functions (sum, difference, table)

The next step was to introduce a complexity in the control playback. A detailed explanation is required for this one.

When the aircraft was tested, it achieves trim at a particular elevator position. When the simulation is trimmed, it arrives at an elevator position that is probably different than the criteria. In order to excite the Phugoid dynamics, a pulse is introduced into the elevator. If we attempted to input the elevator angles directly from the criteria data, on the very first frame the elevator would jump from the simulation trimmed position to the aircraft trimmed position, which might be a significant jump. This will result in a response when none was expected. Instead, we want to play back the movement relative to the trim position, not the absolute position. As such, I needed to determine the position of the elevator when trimmed, and calculate the offset from the criteria. This offset is then removed from the criteria position for the duration of the maneuver to arrive at the elevator position that we need to command, or a delta command.

This script shows how to do some of the math functions, as well as how to declare and use new parameters. These parameters are visible only within this script, and are destroyed when the script exists. With a little examination, you can probably see what it is doing.

File: aircraft\Tutorial_1\scripts\phugoid.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
    name="Phugoid">

  <use aircraft="Tutorial_1" initialize="scripts/airborne"/>
  <run start="0.0" end="244" dt="0.0166667">

    <!-- Declare the local parameters -->
    <property> fcs/elevator-offset-deg </property>
    <property> fcs/elevator-playback-cmd-deg </property>
    <property> fcs/elevator-cmd-deg </property>

    <event name="Trim">
      <description>Trim at the initial conditions state</description>
      <condition> simulation/sim-time-sec ge 0.0 </condition>
        <set name="ic/vc-kts"  value="157.0"/>
        <set name="ic/h-sl-ft" value="9624.0"/>
        <set name="simulation/do_simple_trim" value="1"/>
    </event>

    <event name="Calculate the elevator offset in degrees">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
```

```
      <set name="fcs/elevator-offset-deg">
        <function>
              <difference>
                <!--<property> fcs/elevator-pos-deg </property> -->
                <value> 0.0 </value>
                <value> 0.244034 </value>  <!-- the first value in the playback table -->
              </difference>
        </function>
      </set>
      <notify>
         <property> fcs/elevator-pos-deg </property>
         <property> fcs/elevator-offset-deg </property>
         <property> fcs/pitch-trim-cmd-norm </property>
      </notify>
    </event>


    <!--
    Playback an actual time history of the elevator position. Here, we are just looking
    up the elevator position from the criteria data
    -->
    <event name="Calculate the cmd playback elev pos in deg" continuous="true">
      <description>Provide a time history input for the elevator</description>
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="fcs/elevator-playback-cmd-deg" >
          <function>
              <table>
                <independentVar lookup="row">simulation/sim-time-sec</independentVar>
                <tableData>
                    0.00    0.2440
                    0.25    0.2497
                    0.50    0.2578
                    0.75    0.2341
                    1.00    0.2310
                    1.25    0.2124
                    1.50    0.2150
                    1.75    0.2363
                    2.00    0.2320
                    2.25    0.2291
                    2.50    0.2387
                    2.75    0.2631
                    3.00    0.2538
                    …….          (Lots of values snipped out)
                    243.50    0.1979
                    243.75    0.1962
                    244.00    0.2048
                </tableData>
              </table>
        </function>
      </set>
    </event>

    <!--
    Here we are just calculating the new commanded elevator position, from
    the position in the criteria data, and the offset of the JSBSim trimmed
    position from the criteria
    -->
    <event name="Calculate the cmd elev pos in deg" continuous="true">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="fcs/elevator-cmd-deg">
        <function>
          <sum>
             <property> fcs/elevator-playback-cmd-deg </property>
            <property> fcs/elevator-offset-deg </property>
          </sum>
        </function>
      </set>
    </event>

    <!--
    Calculate the normalized position for the elevator. Note that this reflects
```

```
    the FCS section in the Tutorial_1.xml file, although we used degrees here
    instead of radians, just to eliminate one conversion.
    -->
    <event name="Calculate the normalized cmd elevator position" continuous="true">
      <condition> simulation/sim-time-sec ge 0.2 </condition>
      <set name="fcs/elevator-cmd-norm">
        <function>
            <table>
            <independentVar lookup="row"> fcs/elevator-cmd-deg </independentVar>
            <tableData>
                -20.0    -1.0
                 20.0     1.0
            </tableData>
          </table>
        </function>
      </set>
    </event>

  </run>

</runscript>
```

There is a control modeling issue that is worth discussing at this point. The Aeromatic model produced is intended for use on a computer, probably flying FlightGear though a joystick. The joystick is a spring-centering input device. There are also trim inputs, which on some joysticks, are small sliders on the side of the joystick. The two inputs are considered in parallel, and are summed together to calculate the elevator position.

In most general aviation aircraft and many older generation commercial and military aircraft, the controls are considered reversible. This means that the yoke or stick is connected directly to the elevator through a linkage system of cables, pushrods, turn cranks, gears, and other devices. When the cockpit control is moved, the surface moves, and when the surface moves, the cockpit control moves. When the trim is activated, the surface moves, which moves the cockpit control. This system is considered a series connection, so you can't move the trim independent of the cockpit control.

So, in the JSBSim simulation, when the aircraft is trimmed, the trim position is used, and the elevator is maintained at zero, while in the aircraft, the trim is used to relocate the elevator to a non-zero location. There is a notation in the script for this test, indicating that the offset is only the offset from 0.0 of the criteria data, without considering the simulation elevator, since the elevator is zero.

Since the script has control positions for the entire duration of the test, this Phugoid test is a stick-fixed response. This would be equivalent to the pilot moving the stick to input the elevator pulse, then return the stick to the original position and holding it there. There is another way to perform this maneuver, called stick-free. As the name suggests, once the pulse is input and the control is returned to neutral, the pilot would release the stick and allow the elevator to float as it wishes, which would move the stick. This level of modeling is not present in the JSBSim Tutorial_1 model now, and probably won't be for several generations of improvements, as this is some pretty involved stuff.

The plots of the test results are shown in **Figure 4.5-1**. Note that the elevator deflection mimics that of the criteria data, but is offset from the criteria.

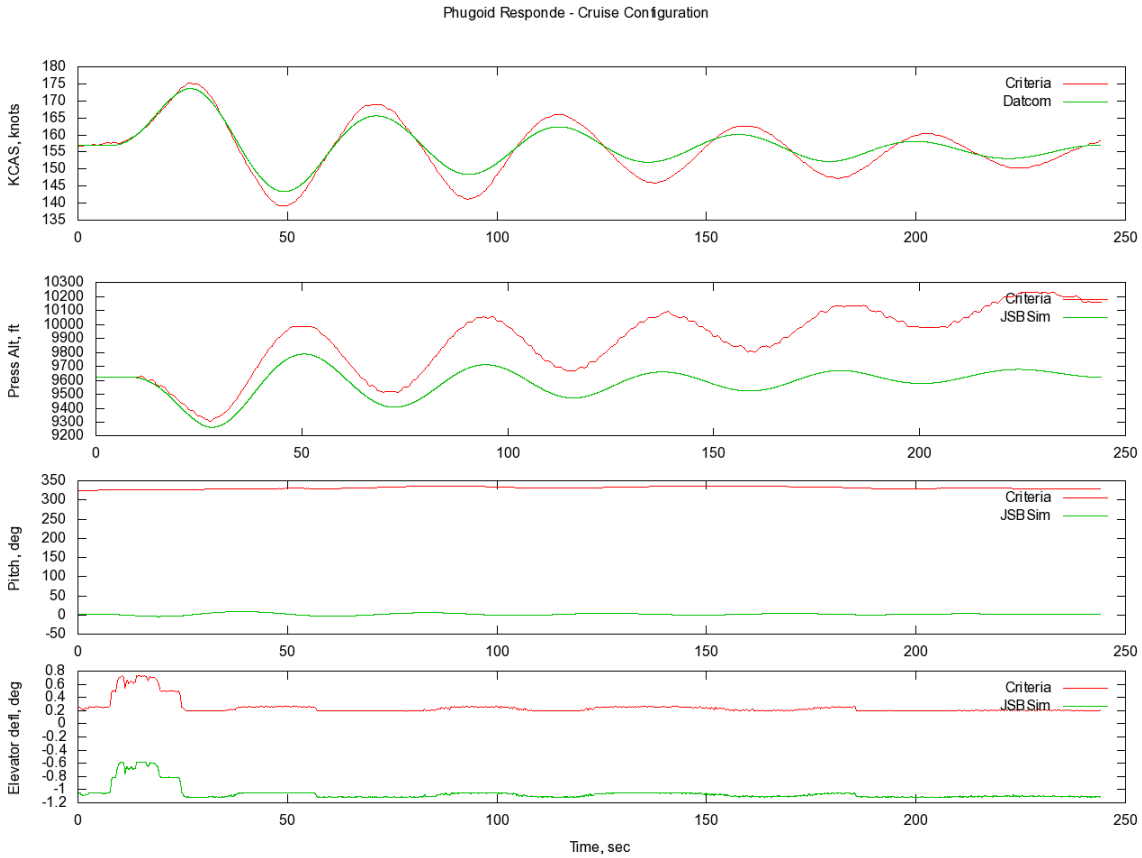


Figure 4.5-1. Test results for phugoid

5 About the Author

I have over 26 years experience building flight simulators for military and commercial applications. I have written and/or used several Automatic Fidelity Test (AFT) systems in the past, and am well versed in testing requirements for flight simulators. I wrote my first plotting routine in 1984, and my first AFT system in 1986. I was familiar with the operation and concepts of JSBSim, but had never written a model or scripts until a couple weeks before I started this tutorial. I support the DATCOM+ aero estimation tool, and one of the outputs is in JSBSim model format.

You can reach me at billg@holycows.net.