JSBSim Script Tutorial 1

Bill Galbraith
Holy Cows, Inc.
November 2010
Initial Release

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 additional purpose of this report is to demonstrate FAA-style test scripts for JSBSim.

2 Model Development

An aircraft model of a representative business-class turbojet aircraft was built using the Aeromatic utility found on the JSBSim web page. 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.

3 Scripts

The scripts produced for this tutorial correspond to the FAA Advisory Circulator 120-40B, a now-obsolete standard for aircraft simulator certification. AC 120-40B, as well as the similar AC 120-45A, has been superseded by FAA Part 60 regulations. The tests are the same, but the numbering schemes have changed.

3.1 Test Execution

The tests described below can be run with the JSBSim program. My development environment consists of the following

- Windows XP SP3
- JSBSim checked out from CVS about Nov 2, 2010 and compiled with Visual C++ 2008 Express Edition
- JSBSim execution was accomplished under CygWin with Linux-style scripts, for ease of command-line changes
- Plotting by GNUPlot is shown here, but there are other plotting routines out there. I use gnuplot version 4.4 patchlevel 0. it uses some command line options (-p, -e) not available in version 4.2.
- Criteria data that I used as a basis for testing is located outside the JSBSim directory tree, and is not included in this package

All files for the tutorial aircraft model and scripts are contained within the **aircraft/Tutorial_1** directory, with the exception of the engine model. The tutorial aircraft utilizes the pre-existing J85-GE-5.xml engine.

From the CygWin command line in the JSBSim directory, the following command line executes the appropriate tests

```
Debug/JSBSim –script=aircraft/Tutorial_1/scripts/2c6-a.xml
```

When this script is executed, data is recorded to the file **Tutorial_1.csv**, as specified in the output of the model file **Tutorial_1.xml**. This file is a time history of parameters, separated by commas. This file can be imported into Microsoft Excel easily for plotting, or you can use gnuplot. What I do is move the output file to a results directory under the aircraft/Tutorial_1 directory, via a command-line option. I also pipe the output messages to a file. This is done just to retain the data for future plotting and/or analysis.

```
Debug/JSBSim --script=aircraft/Tutorial_1/scripts/2c6-a.xml \
  --outputlogfile=aircraft/Tutorial_1/results/2c6-a.csv > JSBSim.out
```

To generate a plot of the results, you can use gnuplot. There is a plot scrip in the **aircraft/Tutorial_1/plots** directory to generate the plots. Information on setting up gnuplot for your system can be found on the internet, and varies too greatly between operating systems to be covered in this tutorial. On my system, I just use:

```
gnuplot –p ‘aircraft/Tutorial_1/plots/2c2-a.p’
```

In the gnuplot scripts in the **aircraft/Tutorial_1/plots** directory, I’ve shown how to output the plots to the screen, **.PNG**, or **.PDF** file. You can also specify these commands on the command line, as shown in the **runtest** script below.

Please note that I have generated the plots in the appendices with criteria data plotted as well, so there are two lines. On your plots, there will only be one line.

For testing convenience, I utilized the following Linux script, called **runtest**. To use it, just issue the command:

```
runtest 2c2-a
```

and a plot of the results is generated, both on the screen and in a .png file.

```
#! /bin/sh
set -e

# Remove the old results file
rm -f aircraft/Tutorial_1/results/$1.*

# Run the test
Debug/JSBSim --script=aircraft/Tutorial_1/scripts/$1.xml \
  --outputlogfile=aircraft/Tutorial_1/results/$1.csv > JSBSim.out

# Generate the GNUPlot to the default terminal (screen)
gnuplot -p aircraft/Tutorial_1/plots/$1.p

# Generate the gnuplot to the .PNG file
gnuplot -e "set terminal png size 1280,960; set output 'aircraft/Tutorial_1/results/$1.png'" \
aircraft/Tutorial_1/plots/$1.p

# Output the messages from JSBSim
tail -1000 JSBSim.out
```

‘runtest’ script for Cygwin

3.2 General Test notes

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. The default for the aircraft parameters was used.

3.3 **2c6-a.xml** – Longitudinal Trim, Cruise Configuration

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.

```
<?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="FAA AC 120-40B test 2c6-a.">

  <use aircraft="Tutorial_1" initialize="scripts/2c6-a_init"/>
  <run start="0" end="1" dt="0.00833333">

    <!-- Perform a full trim -->
    <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="simulation/do_simple_trim" value="1"/>
    </event>

  </run>

</runscript>
```

File: 2c6-a.xml

Look at the **2c6-a.xml** files. In it, you will see a header, including xml specifications on the first 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/Tutorial_1** directory, in the **Tutorial_1.xml** file. This line also specifies the initialization file **scripts/2c6-a_init.xml**, which JSBSim knows to look in the directory **aircraft/Tutorial_1** directory, in the **scripts/2c6-a_init.xml** file, since we specified the directory **scripts**. We will look at the initialization script below.

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 (<set name="simulation/do_simple_trim" value="1"/>). 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 it sees the *do_simple_trim* statement, it goes off and trims the aircraft to the initial conditions specified, without updating *sim-time-sec*. After the trim is complete, a report is issued to *stdout* (the default is the screen) showing the results of the trim. JSBSim then executes for 1 second, and 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/2c6-a.csv**.

```
<?xml version="1.0" encoding="utf-8"?>
<initialize name="2c6-a_init">

  <!-- This file sets up the aircraft initial conditions. -->
  <altitude unit="FT">  4779.0  </altitude>
  <vc unit="KTS">        191.0  </vc>
  <phi unit="DEG">         0.0  </phi>
  <psi unit="DEG">         0.0  </psi>
  <beta unit="DEG">        0.0  </beta>
  <gamma unit="DEG">       0.0  </gamma>
  <running>               -1   </running>
</initialize>
```

File: 2c6-a_init.xml

Let’s look at the initialization file **2c6-a_init.xml**. After the header on the first line and a comment for the description, it sets the items that make up the initial condition. It sets the altitude and airspeed (in this case I use calibrated airspeed), as well as default values for phi (roll angle), psi (heading), beta (sideslip), and gamma (flight path angle). The last line, *running* specifies that all engines should be running. In the file in the distribution package, below that in comments, are all of the parameters that can be set in the initial condition. Some of them are mutually exclusive. For example, if you set calibrated airspeed *vc*, you don’t have to set true airspeed *vt*.

The JSBSim routine does such as wonderful job of trimming. I initially question whether the simulation was indeed running freely after the trim. After 120 seconds, the aircraft has only climbed 20 feet, and lost 0.4 knots airspeed. Pitch angle and alpha of attack were almost unchanged. The plotted results of this test are not shown, as there is nothing exciting to see.

3.4 2c6-b – Longitudinal Trim, Approach Configuration

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.

```
<?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="FAA AC 120-40B test 2c6-b.">

  <use aircraft="Tutorial_1" initialize="scripts/2c6-b_init"/>
  <run start="0" end="4" dt="0.00833333">

    <event name="Configure aircraft">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="fcs/flap-cmd-norm" value="0.5"/>
    </event>

    <!-- Perform a full trim -->
    <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="simulation/do_simple_trim" value="1"/>
    </event>

  </run>

</runscript>
```

File: 2c6-b.xml

Looking at the **Tutorial_1.xml** file, we find that there are settings for 15 and 30 degrees. Since maximum deflection is 30 degrees, 15 deg deflection normalized is 0.5. The **2c6-b.xml** script shows this being set before the trim is performed. Note that the flaps MUST be set before the trim is initialized. This test is run for 4 seconds.

As the **2c6-b.xml** script stands now, we have direct control of the flap position and do not have to wait for it to ramp down. If the modeling for the flaps system had been more thorough, we would have to set the cockpit control to the approach setting, and wait for the flaps to come down before trimming. We might do that with a condition in the event to wait until the flaps are down, since the time could vary as the flap model matures. As an error check, we also might check to see if the time exceeds a large value (30 seconds) and the flaps aren’t down yet, and issue a warning message.

No plot of these results is shown, as there is nothing exciting to see there.

3.5 **2c2-a –** Flap Change Dynamics – Retraction

What this script demonstrates:

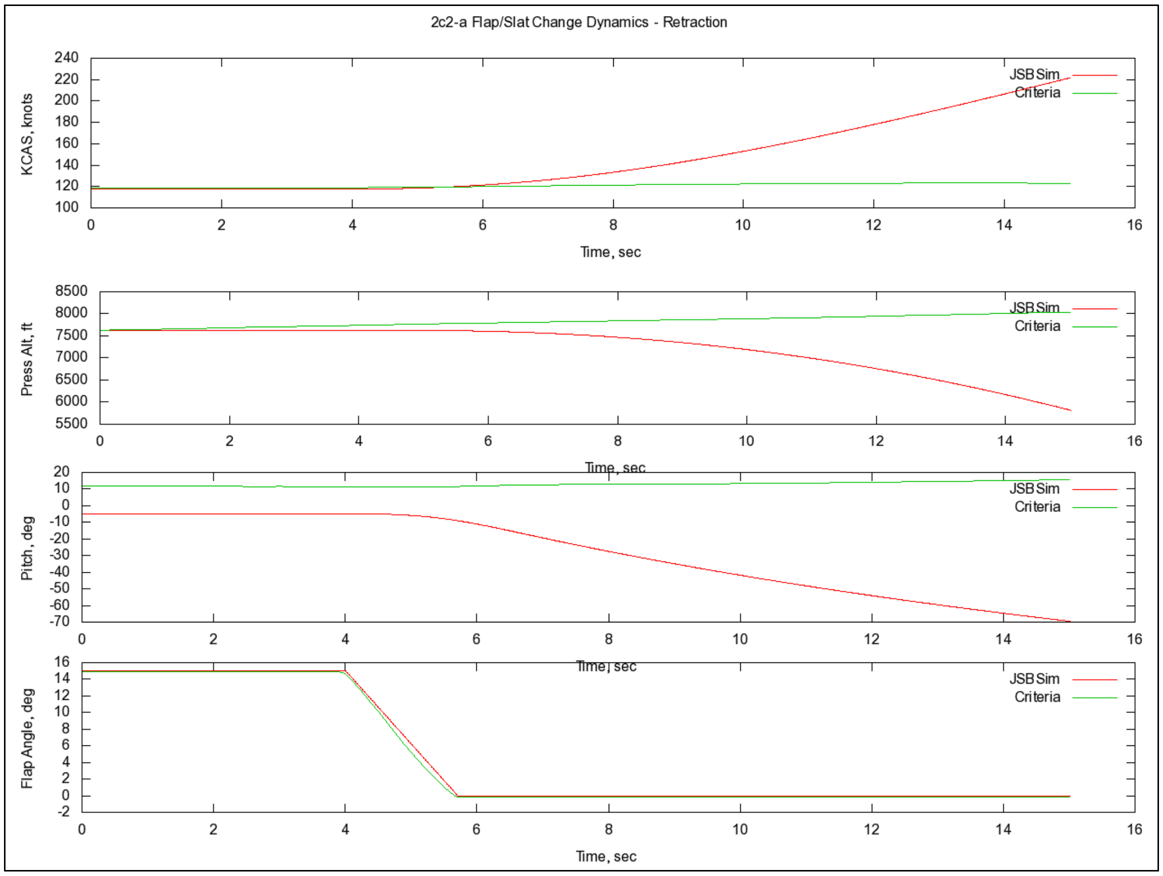
- Ramped input of a control

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 retract the flaps in 1.7 seconds, using the *ramp* function. 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.

```
<?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="FAA AC 120-40B test 2c2-a.">

  <use aircraft="Tutorial_1" initialize="scripts/2c2-a_init"/>
  <run start="0" end="15" dt="0.00833333">

    <event name="Configure aircraft">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="fcs/flap-cmd-norm" value="0.5"/>
    </event>

    <!-- Perform a full trim -->
    <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <set name="simulation/do_simple_trim" value="1"/>
    </event>

    <!-- Retract the flaps 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>
```

File: 2c2-a.xml

The time history plot of the results plotted against the criteria data is shown below.



Test 2c2-a Results

3.6 **2d2-a** – 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 bank angle *phi* is specified in the file **2d2-a_init.xml**, as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<initialize name="2d2-a_init">

  <!-- This file sets up the aircraft initial conditions. -->
  <altitude unit="FT"> 10592.0  </altitude>
  <vc unit="KTS">       216.0  </vc>
  <gamma unit="DEG">      0.0  </gamma>
  <phi unit="DEG">      -42.0  </phi>
  <psi unit="DEG">        0.0  </psi>
  <beta unit="DEG">       0.0  </beta>
  <running>              -1  </running>
</initialize>
```

File: 2d2-a_init.xml

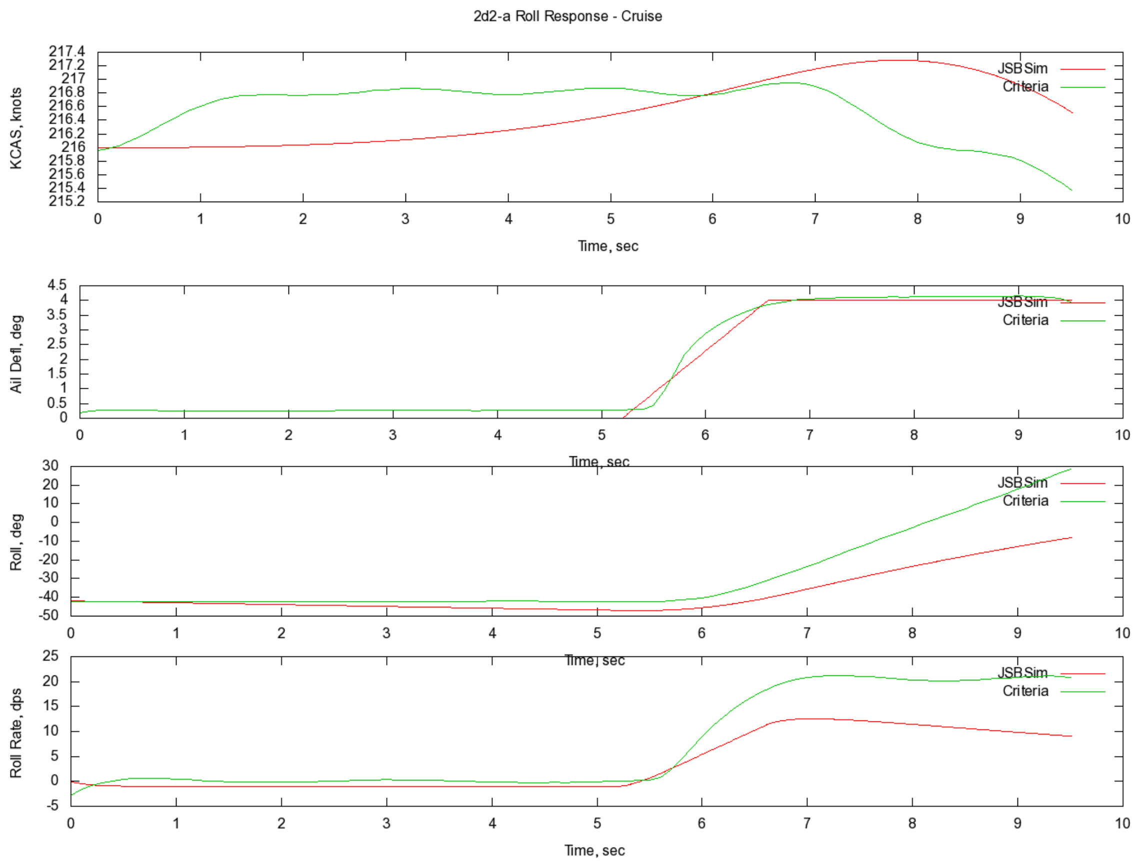
The test was first written with just a trim to a bank angle and allowed to run. The aircraft eventually rolled out wings level as expected. Next, a simple control input was introduced which was close to the criteria data input. Note the use of the *function table* construct.

```
<?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="FAA AC 120-40B Test 2d2-a">

  <use aircraft="Tutorial_1" initialize="scripts/2d2-a_init"/>
  <run start="0" end="9.5" dt="0.00833333">

    <!-- Trim in a turn -->
    <event name="Trim">
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <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>
```

File: 2d2-a.xml

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 below.



Test 2d2-a Results

3.7 **2c10-a** - 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 significant. 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.

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.

```
<?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="FAA AC 120-40B Test 2c10-a">

  <use aircraft="Tutorial_1" initialize="scripts/2c10-a_init"/>
  <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>

    <!-- Perform a full trim -->
    <event name="Trim">
      <description>Trim at the initial conditions state</description>
      <condition> simulation/sim-time-sec ge 0.0 </condition>
      <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
                  <!—Lots of data points removed →
                  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>
```

File: 2c10-a.xml

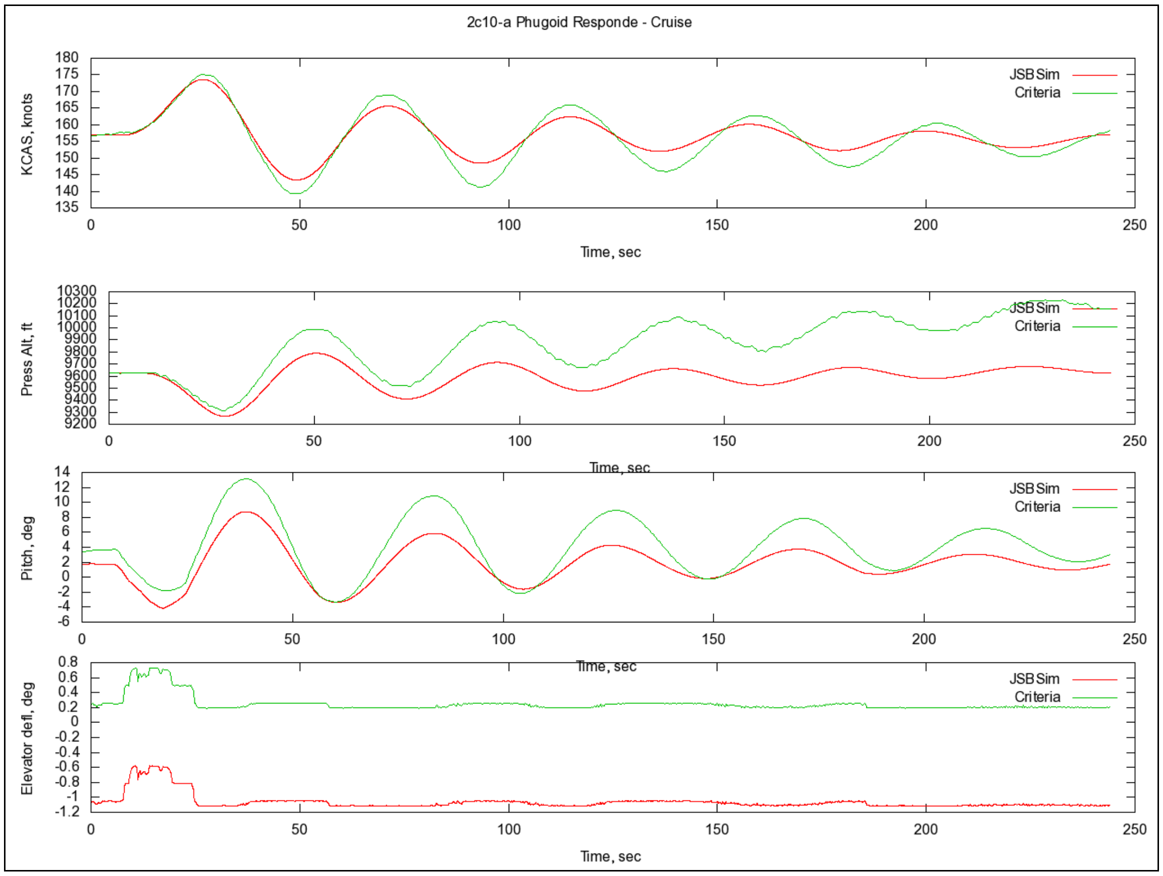
There is a control modeling issue that is worth discussing at this point. The Aeromatic model produced is intended for use on a computer, probably flying FlightGear though a joystick. The joystick is a spring-centering input device. There are also trim inputs, which on some joysticks, are small sliders 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, one 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 is shown below. Note that the elevator deflection mimics that of the criteria data, but is offset from the criteria.



Test 2c10-a Results

4 Future Plans

4.1 DATCOM+ model

The author supports the software package DATCOM+, which is an extension of the US Air Force Digital Datcom computer program. DATCOM+ . One of the next steps will involve generating a JSBSim model of an aircraft. The DATCOM+ package has a number of example aircraft already, so generation of a JSBSim model will be fairly easy, Implementation of this model in JSBSim should help reveal any flaws in the DATCOM+ output, and should help improve DATCOM+.

4.2 Aircraft-specific configuration

Once a more representative JSBSim model is built, I can start improving the scripts to include more aircraft-specific configuration items, such as weight, center of gravity, and inertias, as well flap and gear dynamics. At the same time, atmospheric parameters will be refined such as OAT (Outside Air Temperature), barometric pressure, and winds.

4.3 Aircraft-specific control system

The control system for the Tutorial_1 aircraft as generated by Aeromatic is designed for joystick input, with the control and trim being parallel parameters which are summed to obtain the surface deflection. A control system more representative of an aircraft will allow the scripts to be more representative of that required to use the JSBSim model and scripts on an actual airplane simulator.

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 before. I support the DATCOM+ aero estimation tool, and one of the outputs is in JSBSim model format.