DEVELOPING A PROCEDURE FOR SOFTWARE TESTING USING SYNTHETIC DATA

Shanna-Shaye Forbes

Academic Affiliation, Fall 2004: Junior, University of Texas at Austin

SOARS® Summer 4004

Science Research Mentors: David Rogers, Christopher Webster, Mike Daniels
Writing and Communication Mentor: Wendy Abshire
Community Mentor: Susan Cross
Peer Mentor: Melissa Burt

ABSTRACT

The Nimbus data processor was developed for the National Center for Atmospheric Research’s Research Aviation Facility to process the raw data collected during research flights and to compute derived variables dependent on these measured parameters. Over its years of use scientists accepted on faith that the derived variables produced by Nimbus were accurate, however, they lacked an automated tool to test consistency and accuracy. This project’s aim was to create a synthetic data injector (SDI) to test the consistency and accuracy of Nimbus’ derived variables. This project resulted in the development of a SDI that allows scientists to use synthetic data. They can observe how Nimbus handles raw data anomalies, and evaluate the sensitivity of derived variables. Synthetic data injection was accomplished when newly written code modified Nimbus’s existing code which opens an existing raw data file, decodes the information and stores the decoded information in memory before processing it. The newly written code replaced data in memory with synthetic data before processing, and scientists were then able to view the modified data. Modifications were made to a raw variable, and the changes in the raw variable became evident in the derived variables. The synthetic data injector demonstrated the consistency of Nimbus’ derived variables and allows scientists to modify aircraft data.

This work was done under the auspices of the Significant Opportunities in Atmospheric Research and Science (SOARS®) program of the University Corporation for Atmospheric Research, with funding from the National Science Foundation, the U.S. Department of Energy, the National Oceanic and Atmospheric Administration, and the Goddard Space Flight Center, NASA

SOARS® 2004, Shanna-Shaye Forbes, 1
Section 1: INTRODUCTION

As curiosity about the world around us increases, and as technological advances allow exploration of previously unexplored aspects of our world, the number of research projects conducted by scientists to collect information about different parts of the atmosphere increases. One may wonder what happens to the information that is collected. How is it later displayed and made available for comparison with models that simulate different aspects of our atmosphere? How accurate is the displayed information? What does a scientist do if an instrument malfunctions during a field experiment? This paper discusses the development of a software tool, known as the Synthetic Data Injector that helps address problems such as instrument performance, data collection over a computer network, and the consequences of incorrect calibration equations.

The newly created Synthetic Data Injector (SDI) enables testing of the consistency and accuracy of the data processor known as Nimbus, and also allows scientists to modify data recorded from malfunctioning instruments. Christopher Webster developed the Nimbus processor at the National Center for Atmospheric Research’s (NCAR) Research Aviation Facility (RAF) to process raw data from instruments aboard aircrafts such as RAF’s C-130. Nimbus’s output is used to test theoretical predictions, to improve the understanding of atmospheric processes, and to compare with forecast models.

The approach taken in this project was to inject synthetic data into memory after the raw data are decoded and before the data entered the Nimbus processor. This approach allows testing of the Nimbus processor’s derived variables, and also the removal of erroneous data before Nimbus processing. This method also allowed the synthetic data injector to be both instrument and frequency independent. An instrument and frequency independent SDI ensures that each time a new instrument is added to the aircraft, an instrument specific synthetic data injection procedure will not need to be created.

This paper is organized into five sections. Section 1 introduces the SDI, section 2 provides background information on the Nimbus processor, section 3 documents the implementation method for the SDI, section 4 discusses implications of the SDI’s contribution to Nimbus’s accuracy, and finally possible future enhancements to the Synthetic Data Injector are presented in section 5.

2. BACKGROUND

This section introduces Nimbus, discusses the file format in which Nimbus saves processed data, and briefly describes the program used to view Nimbus’s processed data.

2.1 The Nimbus Processor

Christopher Webster developed the Research Aviation Facility’s data processing software Nimbus (Figure 1), using the “C programming language and a Motif based Graphical User Interface” (Ruth, 2004). The Nimbus data processor processes raw data that have been collected on field experiments and “recorded onto removable hard drives by an onboard data system [known as an ADS].” Nimbus “read[es] raw RAF aircraft ADS data from a … file, uses an extensive standardized library of functions [to compute] derived variables, supports multi-rated output format [which allows] different variables to be output at different rates, and writes output files in netCDF format with data in engineering units suitable for analysis” (Ruth, 2004).
2.2 The netCDF file

"The netCDF software was developed at the Unidata Program Center in Boulder, Colorado." netCDF stands for "network Common Data Form" and "is an interface for array-oriented data access and a library that provides an implementation of the interface. The netCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data" (Unidata, 2004). The netCDF file produced by Nimbus can be viewed with a variety of display and analysis programs, including NCPILOT.

2.3 NCPILOT

NCPLLOT is a computer program that was developed by Christopher Webster at RAF. It "allows users to view and analyze [processed] aircraft data, [and] supports both low rate and mixed rate data" (Webster, 2004).

3. METHODOLOGY

3.1 Pre Synthetic Data Injector Design

Before creating the synthetic data injector, it was imperative to ask the users what they wanted the SDI to feature. It was also important to find out how they might use the SDI for data modification after the consistency and accuracy of Nimbus's derived variables was assured. To accomplish this task, a survey of scientists (Appendix A) who generally use the Nimbus
processor was conducted. Their responses (Appendix B) were gathered and studied to aid in the creation of an SDI that would suit their needs. The survey showed that scientists wanted a SDI that could read data from a file, change all values of a variable to a constant value, and allow them to use a function to modify the measured values. In addition, they wanted the SDI to be frequency and instrument independent so that multiple synthetic data injectors would not be necessary. Scientists also wanted the SDI to be able to read synthetic data from a file containing one or many variables. Some of the features requested were more complicated and beyond the limited scope of this project. They can be addressed in future extensions of the SDI.

3.2 Synthetic Data Injector Design

To inject synthetic data into the Nimbus processor two possible methods were considered and only one was developed (Figure 2). Method 1 requires obtaining the synthetic data in engineering calculations from a user-created ASCII file; converting the engineering values to voltage, and then converting the voltage to raw data. Next, the ADSdata file is permanently replaced with synthetic data in raw form, and then sent through the Nimbus processor. Method 2 requires obtaining the synthetic data in engineering values from a user-created ASCII file, inserting the synthetic data into memory after the raw data has been decoded and converted to engineering calculations, and then sending that data through the Nimbus processor.

Method 2 is the preferred method for three reasons. First, method 1 unlike method 2 requires the creation of a new procedure to read the data from a user created ASCII file, and to convert the engineering values read from the file to its raw format each time a new instrument is added to the data collecting aircraft. Second, method 2 allows the user to modify different variables and allows the user to observe how a change to one or two variables affects a derived variable, without permanently changing the raw variable. This allows the user to run multiple Nimbus sessions to modify other measured variables and to observe how changes affect derived variables. This method enables the scientist to determine which measured variable has greater influences on derived variables. Third, method 2 saves time, because unlike method 1 it does not have to convert engineering calculations to raw binary data.
Figure 2. Schematic data flow through the Nimbus processor and two options for inserting synthetic data. Method 2 was chosen for this project.

3.3 Creating the Synthetic Data Injector Stage 1

The task of the SDI is to replace the flight data with information either from a file, a constant, or from a function; to find out whether the variables were collected by an analog/digital instrument or a block probe, to find out at which rate the user is processing information, and to inject information into the AveragedData, SampledData, or HighRateData arrays whenever appropriate. The destination arrays for synthetic data depend on which arrays are used to compute derived variables and which arrays are used to create the netCDF file.

After conducting the survey and determining what the general user wanted the SDI to feature, code was written in C++ and compiled with G++ to create a non-graphical user interface (non-GUI) from the command prompt. The non-GUI will be referred to as stage one of the SDI. C++ was “developed in the middle 1980s at AT&T, [as the C programming language’s] most popular descendant” (Indiana University Knowledge Base, 2004), and G++ is the “the traditional nickname of GNU C++, a freely redistributable C++ compiler” (Free Software Foundation, 2004).

Stage one of the SDI allowed the user to inject synthetic data from only one source (a file, a constant, or a function) during each Nimbus session. In addition, the user was only allowed to inject synthetic data during a low rate run in Nimbus because all derived variables and
the information output to the netCDF file are located in the AveragedData array. Nevertheless, stage one of the SDI was necessary to ensure that at the most basic level, synthetic data were being injected into the Nimbus processor.

During stage one of the SDI’s development the algorithm below was created to obtain data from the user at the command prompt and to inject user-provided synthetic data into the Nimbus processor. The steps in the algorithm include:

- Find out if the scientist wishes to use synthetic data
- If the scientist wants to use synthetic data find out how they wish to create/use synthetic data (file, function, constant)
  - If file option, then
    - Read from the file if the file exists
    - Find out where the variables listed in the file are located in memory
    - If the time in memory matches the time in the file replace the value in memory with the synthetic value
      - If the times do not match advance the file on memory until times match
  - If constant option, then
    - Ask the scientist for the variable name(s) and value(s)
    - Find out where the variable(s) entered can be found in memory
    - Replace the value(s) in memory with the synthetic value(s)
  - If function option, then
    - Ask the scientist for the variable(s) to be modified, and the function(s) to be applied
    - Find out where the variable(s) can be found in memory
    - Apply the appropriate function to each variable for every time in memory
- Send the data into the Nimbus processor
- Generate netCDF file
- Observe the results in NCPILOT

3.4 Creating the Synthetic Data Injector Stage 2

Stage two of the SDI began after the three types of data injection (file, constant, function) were tested and functioning properly. In Stage two Nimbus’s existing GUI was modified to include the SDI options. To include the SDI’s options an item was added to the Nimbus file menu to allow synthetic data insertion from a file (Figure 3). A box was added to the edit variable pop-up window to allow variable modification by a constant, and a drop down menu was added to the edit variable pop-up window to allow variable modification by a function (Figure 4). Although the most basic methods of injection were being done in stage one of the SDI, stage two is more user friendly. The GUI in stage two allows all three types of synthetic data injection during the same session. In addition, stage two of the SDI allowed the user to inject synthetic data synthetic during both the low rate and the high rate runs of Nimbus. In stage two of the SDI synthetic data are injected into the appropriate array(s) based on which array(s) are used to compute derived variables and which array(s) are used to create the netCDF file. Although the GUI stage of the SDI was difficult to create (it required keeping track of which variable was to be modified, and how the variable was to be modified) it was determined that simultaneous synthetic data injection from multiple sources would be more beneficial to the user.
To accomplish this task a SyntheticData class was created to track variables being modified, record the corresponding values for each variable, and store a synthetic value in memory.

Figure 3. Nimbus user interface with new menu option for Synthetic data.

Figure 4. Pop-up menu has new Functions option.

During stage two of the SDI's development the algorithm below was developed to obtain data from the user via the GUI, and inject synthetic data into the Nimbus processor. The steps in the algorithm include:

- If the user selects synthetic from the file menu, then
  - Read from the file if it exists
  - Use the SyntheticData Class to find out where the variables listed in the file are located in memory

SOARS® 2004, Shanna-Shaye Forbes, 7
• If the user is processing high rate data, then
  • Check the file to ensure that twenty five milliseconds worth of data is provided for every second, if yes then continue processing normally, if no then close Nimbus and display a message to the user letting them know that twenty five milliseconds of data were not provided in the file
    • If the time in memory matches the time in the file, then use the synthetic data class to replace values in memory with synthetic values
      • If the times do not match advance the file in memory until times match
  • If the user enters a constant from the Edit Variable Window, then
    o Use the SyntheticData Class to locate the variable in memory.
    o Use the SyntheticData Class to replace the values in memory with the synthetic values.
  • If the user selects a function from the Edit Variable Window, then
    o Use the SyntheticData Class to locate the variable in memory
    o Use the Synthetic Data Class to replace the values in memory with the function modified values
  • Send the data into the Nimbus Processor
  • Generate a netCDF file
  • Observe the results in NC PLOT

3.5 Testing the Synthetic Data Injector

Before providing the Synthetic Data Injector for general use with Nimbus, it was necessary to test the SDI to ensure that the SyntheticData Class was handling data injection appropriately from a file, a function, and a constant for both the high rate and low rate run of Nimbus. To facilitate this testing, research flight number 5, flown on July 5, 2002, as a part of the Soil Moisture/Sea Salinity field experiment was randomly chosen. This ADS file contained nearly five hours of data.

The SDI was tested with a low rate run of Nimbus and three randomly selected variables were modified with SDI options. The variable DPB (dew/frost point temperature) was set to constant 0.539 from the Edit Variable Window, the variable ROLL (aircraft roll angle) was set to a constant value of 12.523 from a file, and the cosine function was applied to the variable ADIFR (vertical differential pressure) from the Edit Variable Window. The same process was repeated for a high rate run of Nimbus, and a comparison was made between the low rate and high rate runs with and without synthetic data injection. As expected both the high rate run and the low rate run with synthetic data produced the same output (Figure 5) and the high rate and low rate runs without synthetic data produced the same output (Figure 6). Synthetic data injection was successful because the variable DPB which originally fluctuated did in fact contain its constant value or the entire time period, the variable ROLL contained its constant value for the entire time period, and variable ADIFR had a sinusoidal pattern (Figure 6).
Figure 5. Approximately five hours of data showing three variables unmodified low rate and high rate output from Nimbus

Figure 6. Similar to Figure 5, low rate and high rate output from Nimbus, but this time the values are replaced by synthetic data

4. RESULTS AND DISCUSSION

Because of the time constraints of the research it was not possible to perform a full test of all Nimbus’s derived variables; however, a scenario was proposed and run to show that the SDI facilitates testing of Nimbus’s derived variables. The question “what would happen if there was
a leak in the piping of the static pressure port, resulting in a 1 mb increase in the static pressure readings?" was posed. The variable PSDF is measured from the C130’s static pressure port (Figure 7) and it represents atmospheric pressure. As the C130 goes higher into the atmosphere the static pressure decreases and altitude above sea level increases. A modification to the raw variable PSDF should cause changes in its derived variables.

To facilitate this test the variable for raw static pressure PSDF was increased by 1 mb, and we expected to see three affects: a change in variable PALT (altitude above sea level and derived from the variable PSDF), a change in the true air speed variable TASX (indirectly derived from PSDF), and also a change in wind speed and wind direction because they are the difference of ground speed vector (GPS data) and airspeed vector (magnitude TASX).

As demonstrated (Table 1), TASX is not directly derived from PSDF; however, a modification in PSDF should be evident in TASX.
Table 1. The derivation of the variable TASX and its indirect dependency on the raw value PSDF.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Derived From</th>
<th>How Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSC- raw static pressure</td>
<td>PSDF- measured by the c130’s static pressure port</td>
<td>PSX=PSDF</td>
</tr>
<tr>
<td>QCXC- dynamic pressure</td>
<td>PSX</td>
<td></td>
</tr>
<tr>
<td>PSXC- corrected static pressure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTRX- measured by the radome, total temperature that has not been corrected for airspeed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECFRA- recovery factor radome true airspeed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMACH2- the radome mach # (speed of the plane vs. the speed of sound)</td>
<td>QCXC PSXC</td>
<td>$5\left(\frac{q_{cxc}}{psxc} + 1\right)^{0.28571} - 1$</td>
</tr>
<tr>
<td>TASR- aircraft true speed from the radome</td>
<td>TTRX RECFRA RMACH2</td>
<td>$(TTRX + 273.16)^{\left(\frac{401.86}{RMACH2 + RECFRA}\right)}$</td>
</tr>
<tr>
<td>TASCH- aircraft true speed with humidity corrections</td>
<td>TASR and another dependency</td>
<td></td>
</tr>
<tr>
<td>TASX- aircraft’s true airspeed</td>
<td>TASCH</td>
<td></td>
</tr>
</tbody>
</table>

To test this scenario test flight number 1 flown on January 22, 2004 from the Ocean Wave field experiment was selected. From this data file an eight minute time segment was selected because the C130 performed a porpoising maneuver, where it pitched up and down very quickly and then performed a side slip maneuver, where the C130 flies side ways on the left and changes quickly to fly sideways on its right. The results are shown in Figure 8. An increase of 1 mb in the variable PSDF as expected had no effect on the plane’s maneuver-related variables. There was a decrease in the derived variable PALT’s. PALT is inversely related to PSDF. There was an increase in the derived variable TASX that led to modifications in wind speed and wind direction parameters. Although the change in PSDF was small (1 mb), it caused significant changes in derived variables that were consistent with expectations.
As the scenario above suggested a simple equipment malfunction or incorrect calibration can significantly affect the values of many derived variables as well as others that are not directly related to the measured raw value. Thus if a scientist becomes aware that an instrument malfunctioned during an experiment, it may be possible to use the SDI to insert a synthetic value to diagnose the effects of the error or to substitute a suitable replacement value.

5. Future modifications

As scientists begin using the synthetic data injector a list of the attributes that they would like the synthetic data injector to feature will be developed and later, if possible incorporated into the SDI. Meanwhile, in the near future, I hope to develop an algebraic expressing parser to allow users to enter equations from the Edit Variable Window. Additionally, there are many other functionalities I hope to develop including:

- increasing the number of file formats the SDI is able to read
- allowing time jiggering (move values forward or backward in time relative to other data)
- allowing multiple timestamp formats
- allowing the SDI to read from multiple synthetic data files during a single Nimbus session
- developing an extensive library of functions, and
- allowing the users to save and reuse functions they’ve previously entered in this library.

These enhancements will significantly improve upon the current functionality of the SDI, and will enable the SDI to perform many of the tasks requested by users in the survey responses.
References
Free Software Foundation, cited 2004: G++. [Available online at
http://www.gnu.org/directory/GNU/gpp.html ]

Hayhurst, Kelly J. / Langley Research Center Hampton Virginia, cited 2004. Framework for
Small-Scale Experiments in Software Engineering: Guidance and Control Software
Project: Software Engineering Case Study. [Available online at

Indiana University Knowledge Base, cited 2004: What are the C and C++ programming
languages? [Available online at
http://kb.indiana.edu/data/aewf.html?cust=449615.93553.131]

[Available online at http://www.atd.ucar.edu/atd/instruments/raf/ads/]

Data Products From The NCAR Research Aviation Facility. [Available online at
http://raf.atd.ucar.edu/Bulletins/bulletin9.html]

http://my.unidata.ucar.edu/content/software/netcdf/index.html]
Appendix A

This is a copy of the survey sent out to scientists who use the nimbus processor, before the SDI was created.

Hello,

My name is Shanna-Shaye Forbes and I'm a SOARS Protégé working with RAF this summer. My project is entitled "Developing A Procedure For Software Testing Using Synthetic Data". I'll be putting Synthetic Data into the Nimbus Processor which processes the information collected on research flights. Before I start the project I would love to get feedback from scientists who may possibly choose to use the "synthetic data injector" after it has been completed. Initially the "synthetic data injector" will be used to test the nimbus processor for accuracy. The test will be done by inserting different datasets (created by one of my mentors) in ASCII for which the outputs are known into the processor and observing whether or not the expected outputs are received. These datasets will be in calibrated/engineering units and not in raw data form. If the nimbus processor is accurate, the "synthetic data injector" could be used by scientists to modify one or more variables and observe how that affects the results they obtain from the nimbus processor. If per chance a scientist wishes to know what effect an increase in temperature would have on the conclusions they've drawn from a flight data analysis they can simply opt to modify the temperature with "synthetic data" and observe the results. It could also be used to test the sensitivity of a derived variable to its inputs.

Which would you prefer to do if a user created data file is used?
Would you prefer to create one file with all the variables in it along with time stamps?

<table>
<thead>
<tr>
<th>Time</th>
<th>Temperature</th>
<th>Pitch</th>
<th>Roll</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:01:01</td>
<td>23.5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12:01:02</td>
<td>23.501</td>
<td>5.001</td>
<td></td>
</tr>
</tbody>
</table>
Would you prefer multiple files each with a time stamp?

<table>
<thead>
<tr>
<th>Time</th>
<th>Temperature</th>
<th>Time</th>
<th>Pitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:01:01</td>
<td>23.5</td>
<td>12:01:01</td>
<td>5</td>
</tr>
<tr>
<td>12:01:02</td>
<td>23.501</td>
<td>12:01:02</td>
<td>5.001</td>
</tr>
</tbody>
</table>

What do you think the user interface should feature?
Thus far these are the features I've come up with.

Ask which variable the user would like to replace.
Ask the user how they would like the synthetic data created, either from a user created existing file in a specific format or from a modification of the existing file (i.e. with a ramp function, a sinusoidal function, increase the value by 10%, etc).

Do you have any suggestions, would you like to do more? Please let me know what you think. If you're not sure please feel free to jot down anything that comes to mind.

Please feel free to respond until June 24, 2004

Shanna-Shaye Forbes
National Center for Atmospheric Research
Research Aviation Facility: JEFFCO
Office: 121
Phone: 303-497-1023
Email: forbes@ucar.edu

Appendix B

These are some of the survey responses I received from Nimbus users.

Survey Response 1.

Shanna-

In reply to your survey email, here are a few of my thoughts on what I'd like to do with synthetic data.

1. Size distribution measurements of cloud and aerosol particles are stored as histogram data. Statistical metrics such as the mean and standard deviation are calculated from those data, along with the total surface area, total volume, water mass in a volume of air, radar reflectivity, dispersion and other things. It would be worthwhile to test those calculations by using numbers from histograms that I specify.

2. Search for sources of error in wind calculations. The wind is calculated by subtracting the air and ground vector velocities of the aircraft. The ground vector comes from GPS and/or inertial navigation system. The air motion comes from pressure sensors on the nose and the heading. Biases in any of those measurements can affect the winds. It would be useful to see how the ultimate calculations are affected by substituting pressure values or headings.

3. Time jigging. Another useful thing to look at is the time response of different sensors. For example, if the GPS position is lagged by a fraction of a second, how does that affect the wind calculation?
4. ncplot has built-in Fourier spectral analysis tools. It would be useful to test those, for example, by adding a small sine wave to a variable such as temperature, and then see if the spectral analysis finds it and if the amplitude (power) is correct.

5. Similarly, co-spectra software could be tested by adding small sine waves to two independent variables and looking at the coherence and phase.

I'll keep pondering and send some more ideas later. Hope this helps.

Survey Response 2.

Which would you prefer to do if a user created data file is used?
Would you prefer to create one file with all the variables in it along with time stamps?
or

Would you prefer multiple files each with a time stamp?

<table>
<thead>
<tr>
<th>Time</th>
<th>Temperature</th>
<th>Time</th>
<th>Pitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:01:01</td>
<td>23.5</td>
<td>12:01:01</td>
<td>5</td>
</tr>
<tr>
<td>12:01:02</td>
<td>23.501</td>
<td>12:01:02</td>
<td>5.001</td>
</tr>
</tbody>
</table>

One file is better, however, multiple timestamp formats really help, such as fractional hour, fractional day, and seconds since 1970 ... (unix timestamp). Also something in your files indicating a comment so the parser does not get confused. The HH:MM:SS timestamp is hard for computers to work with.

What do you think the user interface should feature?
Thus far these are the features I've come up with.

Ask which variable the user would like to replace.
Ask the user how they would like the synthetic data created, either from a user created existing file in a specific format or from a modification of the existing file (i.e. with a ramp function, a sinusoidal function, increase the value by 10%, etc).

I like the idea of a user creating their own file with the data that they would like to use. Remember to state the format that your program desires, so it is easy to make input files for you program. With user created input files the input functions to the program can be endless. However, if you have time this summer, an interface that enables the user to create a function, and graphically see the function graphed with another dimension (time) would be really nice. Being able to specify which columns would be very helpful, but I don't think as helpful as, specifying the number of rows to use.

Do you have any suggestions, would you like to do more? Please let me know what you think. If you're not sure please feel free to jot down anything that comes to mind.

I would like the program to be able to decimate and/or interpolate to different sampling frequencies, which nimbus can currently do. So if the input file is sampled at 10 Hz, have nimbus output 5 Hz data, so that the effect of the decimation can be seen, such as the impulse and step response. This would be very helpful especially to the sceptical scientists that do flux measurements using ADS.

Survey Response 3.

Which would you prefer to do if a user created data file is used?
Would you prefer to create one file with all the variables in it along with time stamps?
or

Would you prefer multiple files each with a time stamp?

Multiple Files

What do you think the user interface should feature?
Thus far these are the features I've come up with.
Ask which variable the user would like to replace. Ask the user how they would like the synthetic data created, either from a user created existing file in a specific format or from a modification of the existing file (i.e. with a ramp function, a sinusoidal function, increase the value by 10%, etc).

- Superimpose an equation on an already existing value
- Instead of asking the user what they want to modify, remember the last thing modified, or allow the user to pick from a menu of things they’ve already done.
- Have a structure under the program that handles the users’ changes in a specific way (may use a database for this)
- Ask if the user wants to save the changes for later, if yes store it in a database.
- Come up with a library of functions
  - Step change function
  - Sinusoidal
  - White noise
  - Increase the current value by a percentage, offset the value with a constant, or increase the value by 1 or 2 degrees.
- Multiple variable changes could be a possibility for each value
- Put the same offset into the variables pitch and angle of attack to test Nimbus software.
- When Nimbus is run for high rate data some variables are still displayed at a low rate in the high rate output, you may want to put a function to reinterpolate the low rate data to high rate for output.
- Consider adding a function that zips and unzips the users data file.