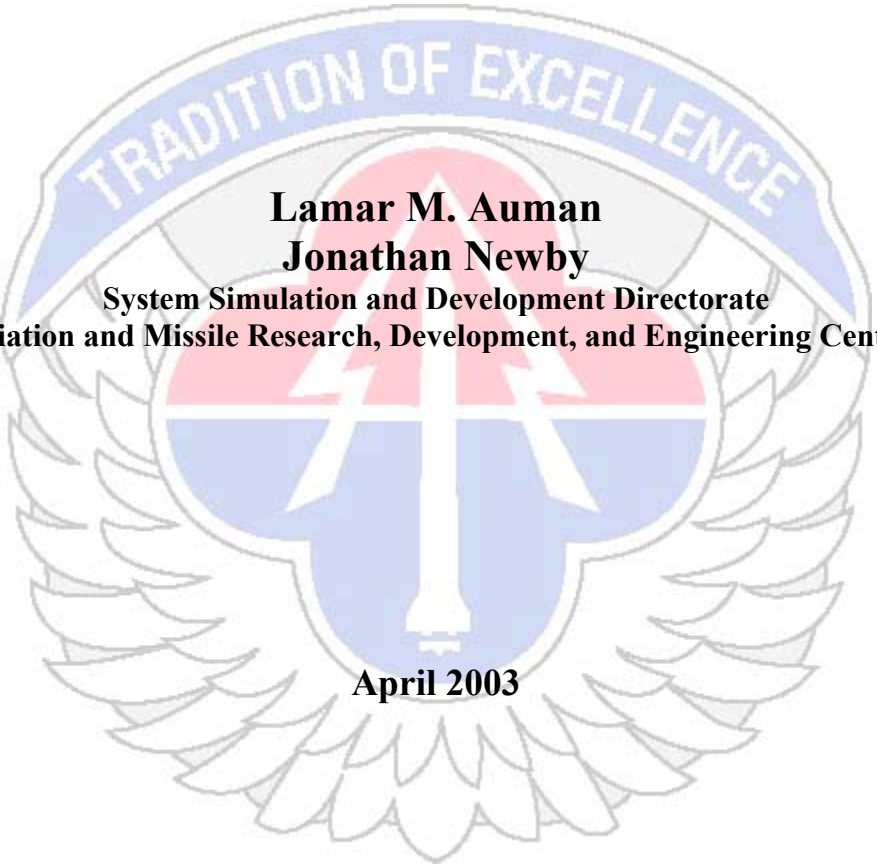
TECHNICAL REPORT RD-SS-03-10

A MATLAB/AERODYNAMIC ANALYZER SYSTEM TOOL

Lamar M. Auman
Jonathan Newby
System Simulation and Development Directorate
Aviation and Missile Research, Development, and Engineering Center



April 2003

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2003	Final

4. TITLE AND SUBTITLE
A MATLAB/Aerodynamic Analyzer System Tool

5. FUNDING NUMBERS

6. AUTHOR(S)
Lamar M. Auman and Jonathan Newby

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Commander, U. S. Army Aviation and Missile Command
ATTN: AMSAM-RD-SS-AT
Redstone Arsenal, AL 35898

8. PERFORMING ORGANIZATION REPORT NUMBER
TR-RD-SS-03-10

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT ***(Maximum 200 Words)***

A MATLAB script has been developed that interfaces with the U. S. Army Aviation and Missile Command's Aerodynamic Analyzer System (AAS). This tool has the capability of automatically removing data biases, forcing delta symmetry, and plotting 3-Dimensional (3-D) data.

14. SUBJECT TERMS
Wind Tunnel Testing, Aerodynamic Analysis

15. NUMBER OF PAGES
68

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONT)

LIST OF TABLES

I. INTRODUCTION

The Aerodynamic Analyzer System (AAS) consists of various programs and flat database files. The AAS is an Army developed tool that has its origins in the 1970's [1 through 5]. Over the last few years, the various FORTRAN codes have been consolidated into one Visual Basic program, dubbed AEROLAB [5]. The MATLAB scripts discussed herein, duplicate most of the capabilities of AeroLab, but add additional 3-Dimensional (3-D) plotting features and a means of incorporating an AAS database into MATLAB.

The MATLAB/AAS program consists of a series of Graphical User Interfaces (GUIs) that allow the user to perform many different tasks that result in either the manipulation or visualization of the data in the database files. The capabilities included in the program are the ability to plot the data in a 2-Dimensional (2-D) XY plot, a cross plot, a carpet plot, or a 3-D plot; differencing runs; coordinate axis conversions; moment reference point shifting; biasing the data including an automatic bias and a forced delta symmetry bias; curve fitting; and exporting the data into Excel worksheets. The data in the program can also be saved into a MATLAB workspace file (.mat file) and loaded later into the program in the same state as when it was saved, meaning any manipulations done to the data can be saved. Also, the workspace file that is created can be later used by different MATLAB script functions.

II. MATLAB/AAS INTERFACE USER'S GUIDE

To begin the program, the MATLAB program must first be opened. Once it is open, the "Current Directory" must be changed to the folder containing the MATLAB/AAS scripts. This is done by clicking on the '…' button next to the Current Directory at the top of the MATLAB Command Window as shown in Figure 1. Once this is done, type in 'aas' at the prompt, and the window shown in Figure 2 will appear. The MATLAB/AAS script is now running. The user must now enter runs before the MATLAB/AAS plot and data manipulation options are functional. Run input is discussed in the following section.

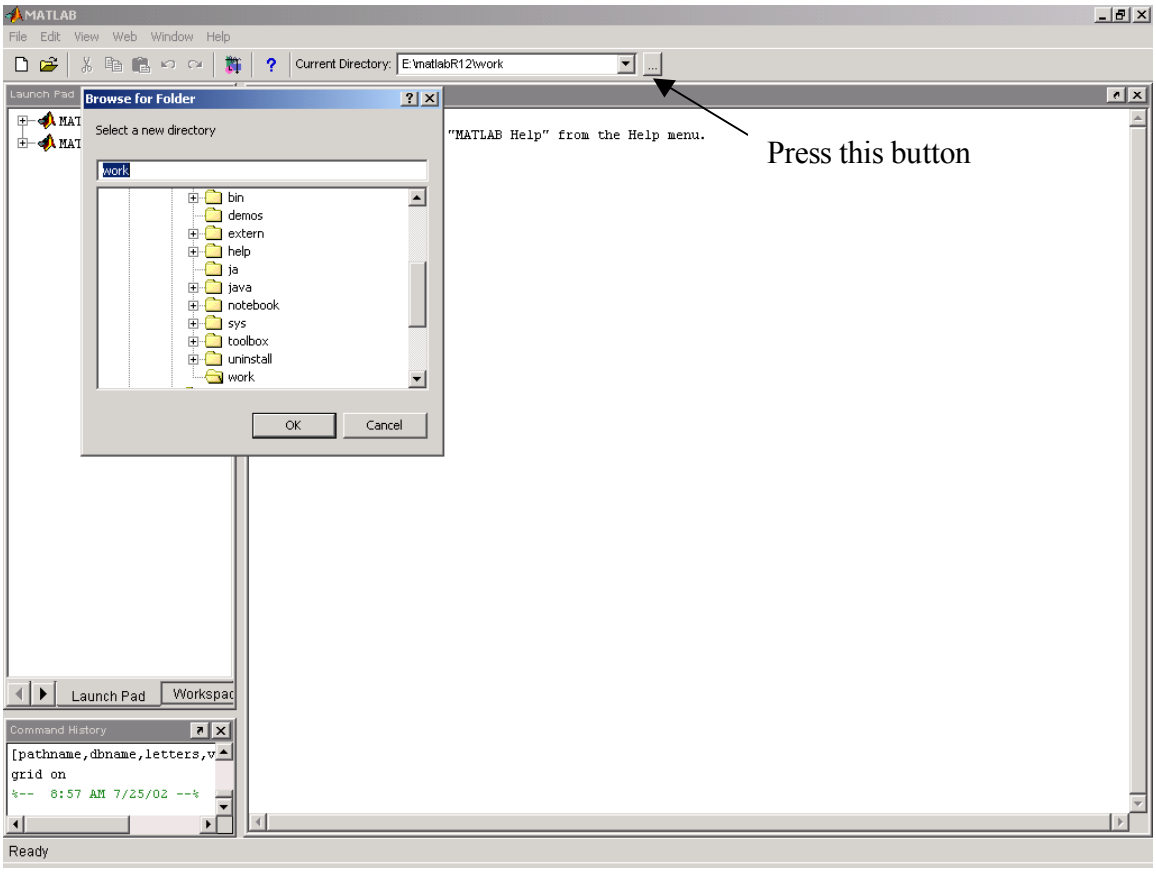


Figure 1. Changing the Directory in the MATLAB Workspace

Figure 2. MATLAB/AAS Control Screen at Start-up

A. Run Input

To select runs to analyze, click on File Menu in the menu bar shown in Figure 2 and select the Input Runs option. If there is no data already inputted in the program, an open file window will appear prompting the user to select the database file to input the runs from, as shown in Figure 4**.** Once the database is selected, the script will display the Input Runs dialogue screen shown in Figure 3. If some runs are already inputted in the program, the Input Runs dialogue screen will show up first with the Current Database set as the database from which the runs were found. The user may change the current database by clicking on the Change button shown in Figure 3.

To operate this window, the user must simply click the runs they want to input and click on the Add button, which will cause the chosen run or runs to appear in the Currently Loaded Runs box. If the Append to Current Runs box is checked, the runs added will be appended to the end of the list of runs in the Currently Loaded box. Otherwise, the runs added will replace those in the Currently Loaded box. To add more than one run at a time, the user must hold CTRL and right click the mouse on each of the runs to add. The Delete and Delete All buttons remove the runs in the Currently Loaded box. Choosing the OK button will cause the program to input the data from the selected runs. Choosing the Cancel button will close the window without inputting any new data.

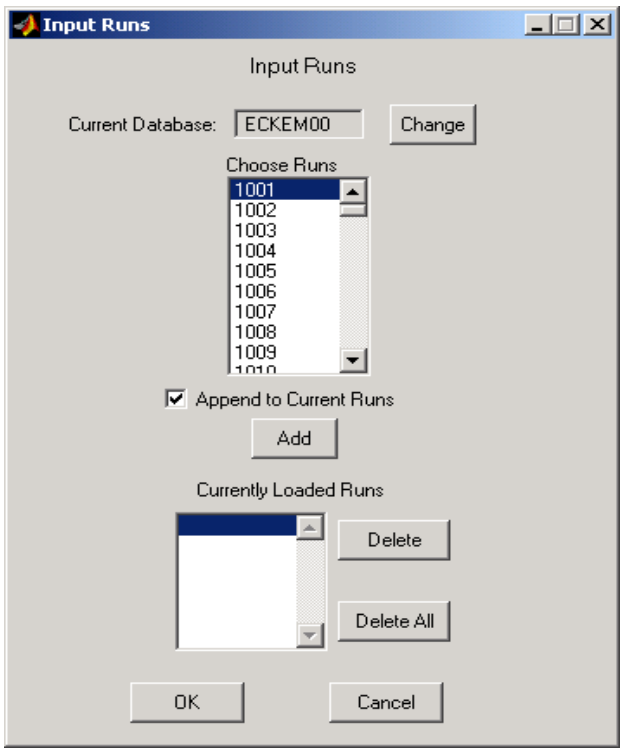


Figure 3. MATLAB/AAS Edit Run Dialogue Screen

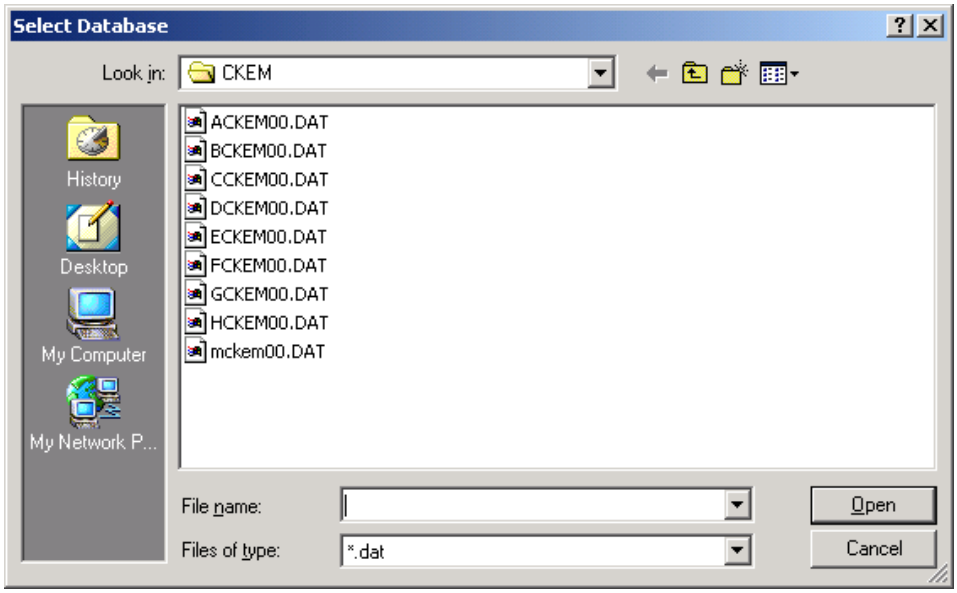


Figure 4. MATLAB/AAS Select Database Screen

1. Choose Runs

Once a set of runs has been selected, the user may specify a sub-set of those runs for plotting by choosing the Choose Runs option under the Runs menu. The window for choosing runs has a box containing the runs loaded in the program with the current runs selected highlighted. An example of this window is shown in Figure 5. Specifying a sub-set of runs will delete any manipulations performed on the data to that point.

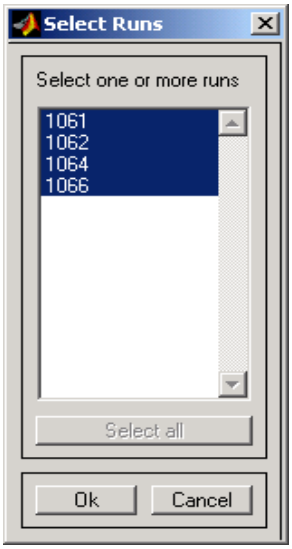


Figure 5. MATLAB/AAS Edit Run Dialogue Screen

2. Save Run Set

At any time in the program, the current data in the program can be saved by choosing the Save Run Set menu option under the File menu. This command saves the data in a MATLAB formatted workspace data file (*.mat). The user is asked whether they want to save the data manipulations or the original data. For example, these include biases, moment reference point shifts, coordinate conversions, or differenced runs. Loading a run set with the manipulations saved will cause the program to be exactly in the same state as when the user saved the data. If the user does not wish to do so, the original data from the database is saved. The Save Run Set window is shown in Figure 6.

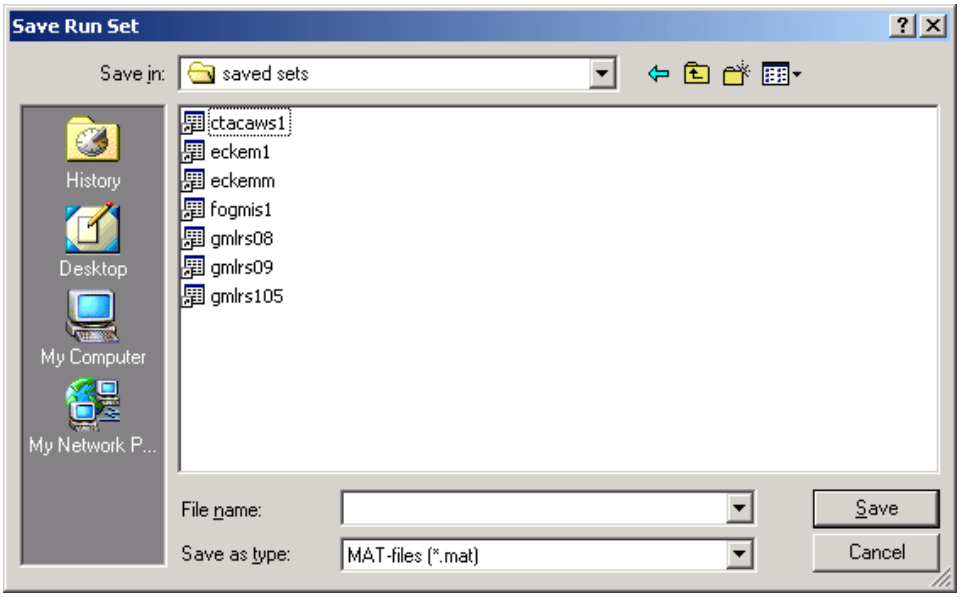


Figure 6. MATLAB/AAS Save Run Dialogue Screen

3. Load Run Set

The data from the program is saved into a MATLAB workspace file, or a .mat file. This data can be later loaded into the program by choosing the Load Run Set menu option under the File menu. This will open a Load Run Set window, as shown in Figure 7. The window will show the MAT-files found in the folders chosen. The user must choose a .mat file that contains data saved by the MATLAB/AAS program.

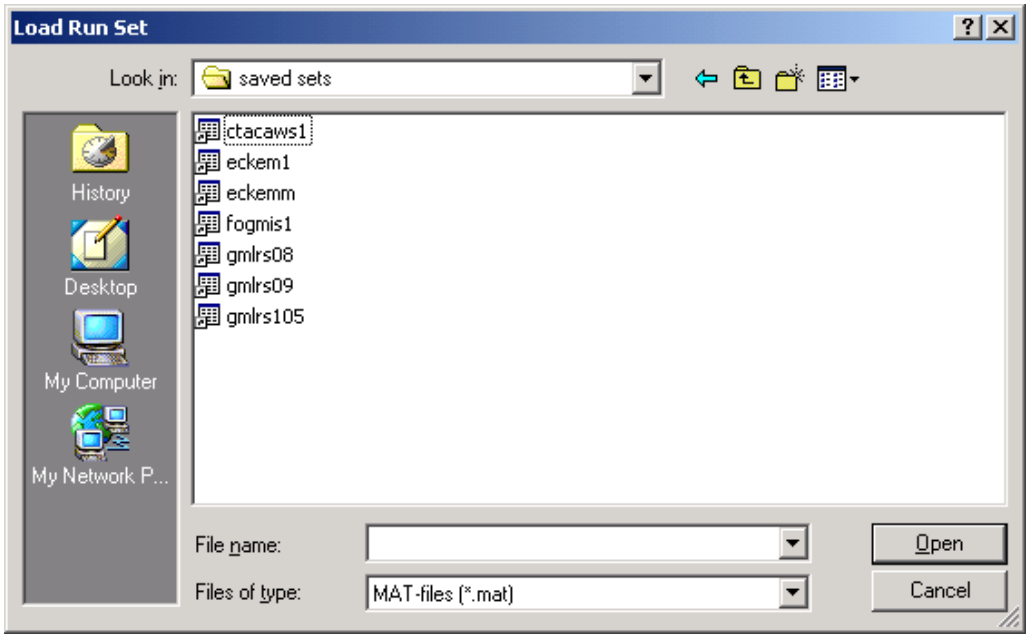


Figure 7. MATLAB/AAS Load Run Dialogue Screen

B. Plot Types

Four types of plots are available to the user, and are listed below.

- 2-D Plot
  - Plots any coefficient versus any coefficient
- 3-D Plot
  - Plots any coefficient against any coefficient and parameter
- Cross Plot
  - Plots any coefficient at given values of the PIV versus any stored parameter. Used for plotting data versus Mach number, fin deflection angle, and so on.
- Carpet Plot
  - Plots stability maps for sets of control deflection runs.

1. 2D Plot

This allows the user to plot any coefficient versus any other coefficient for all of the current runs in the program. When 2-D Plot is selected from the main window, the Plot 2-D dialogue box, shown in Figure 8 is displayed.

To select the desired X and Y variables to be plotted, the user clicks on coefficients from the lists in the X and Y variable boxes. The user is also able to select multiple Y variables (by holding the CTRL button and right clicking on the coefficients), which will produce multiple 2-D plots.

MATLAB/AAS also allows the user to select the values to go in the legend of the X-Y plots. The first box allows the user to select from the list of coefficients to show the values for constant coefficients in the legend. This is provided since some of the independent variable coefficients may be held constant for specific runs, so that value can be seen on the graph. The second box allows the user to select the parameters to show in the legend for each run. Also, the option below the boxes lets the user choose whether to show the configuration codes for the runs in the legend. After pressing OK once the relevant choices are made, the plots will appear on the screen, one plot window for each Y variable selected. An example of a MATLAB/AAS 2-D plot is shown in Figure 9.

The options for the plots produced are discussed in the Plot Options section.

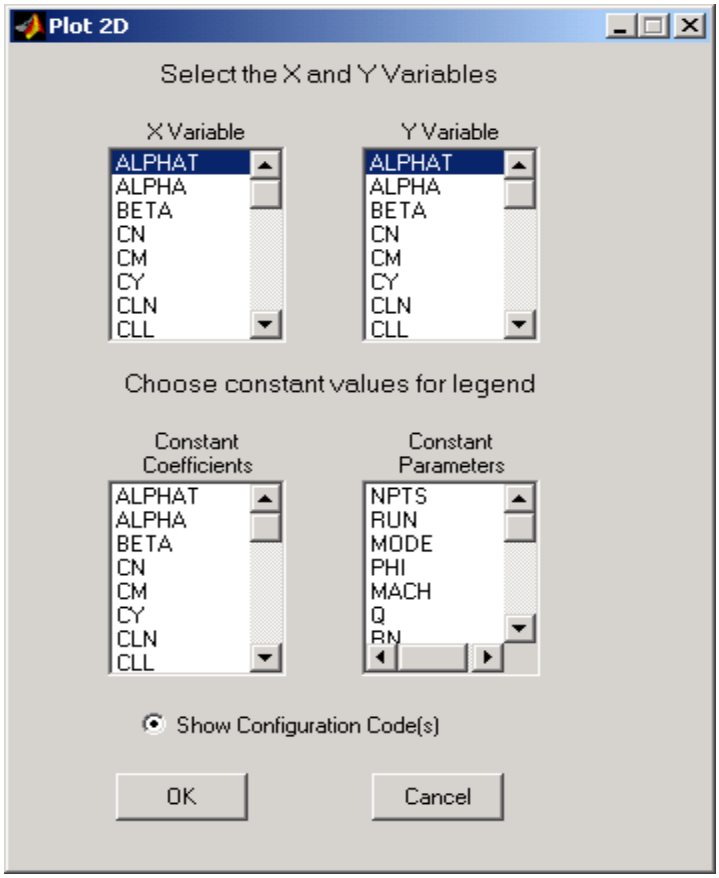


Figure 8. X-Y Plot Variable Input Dialogue Box

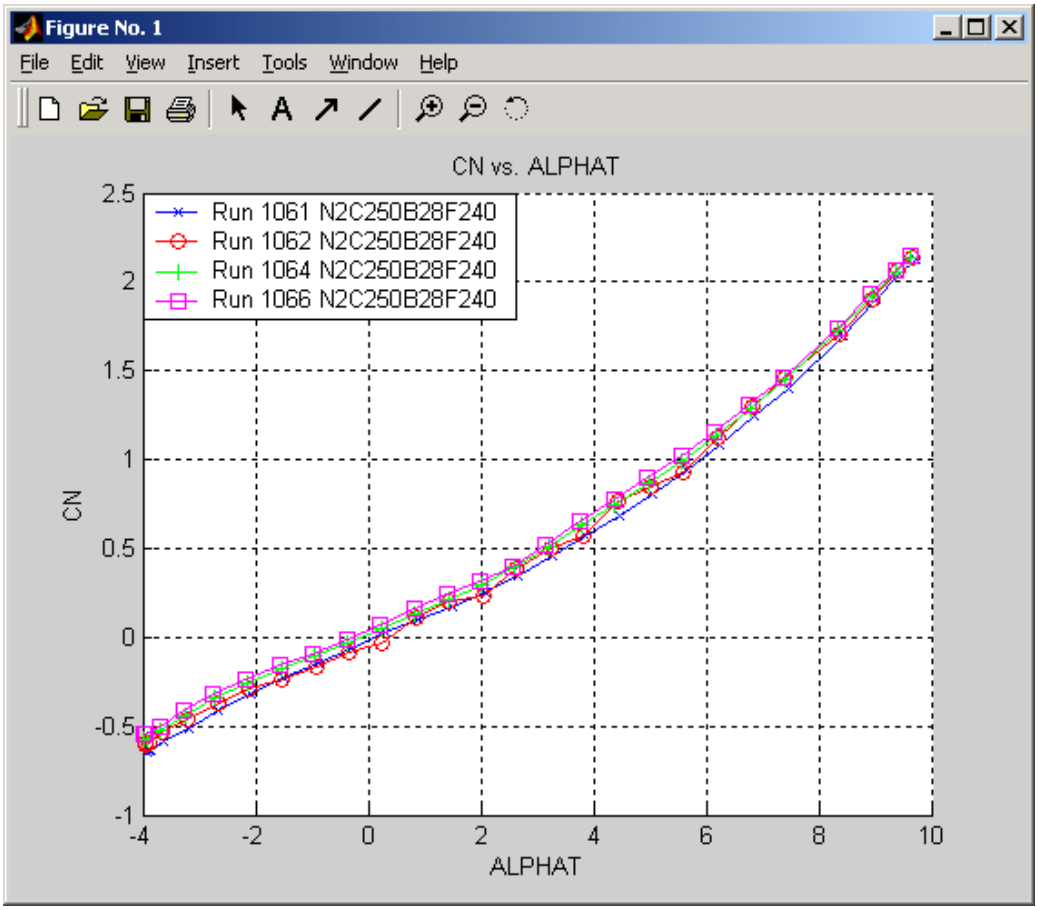


Figure 9. Typical 2-D Plot

2. Cross Plot

The Cross Plot option allows the user to plot any coefficient stored in the database versus stored parameters. Figure 10 shows the Cross Plot dialogue window, and Figure 11 presents a typical Cross Plot. In this example, 4 runs were entered that had fin pitch deflection angles of 0.00, -10.00, -20.00, and -30.00 degrees. The user chose to plot Pitching Moment Coefficient versus Fin2 deflection angle at discrete Angles-of-Attack (AoA) of 0, 5, and 10 degrees. To enter the values for the PIV, they must be entered with a space between each value as shown in the text box at the bottom of Figure 10. AoA is the PIV for each run. MATLAB/AAS first interpolates the run data to determine the values at the user specified PIV values. MATLAB/AAS then plots the data as a function of the Fin2 deflection angle that is stored as a parameter.

Other common uses for cross plot include axial force versus Mach number plots as shown in Figure 12.

The options for the plots produced are discussed in the Plot Options section.

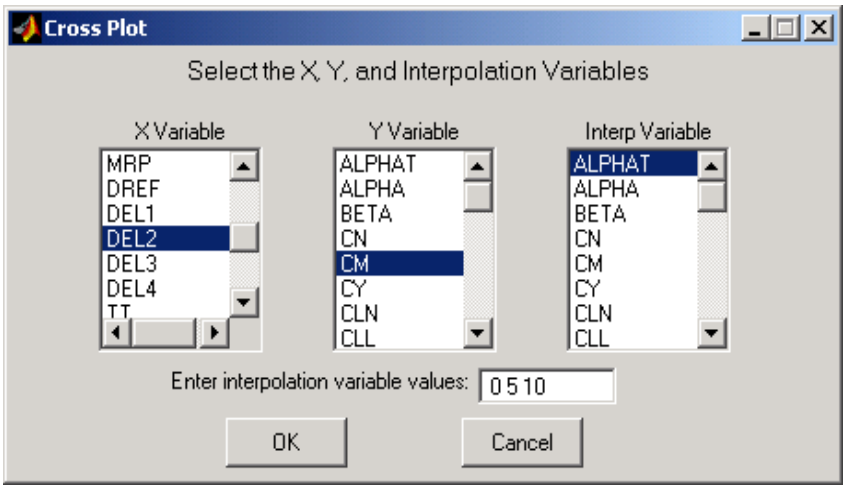


Figure 10. Cross Plot Input Dialogue Box

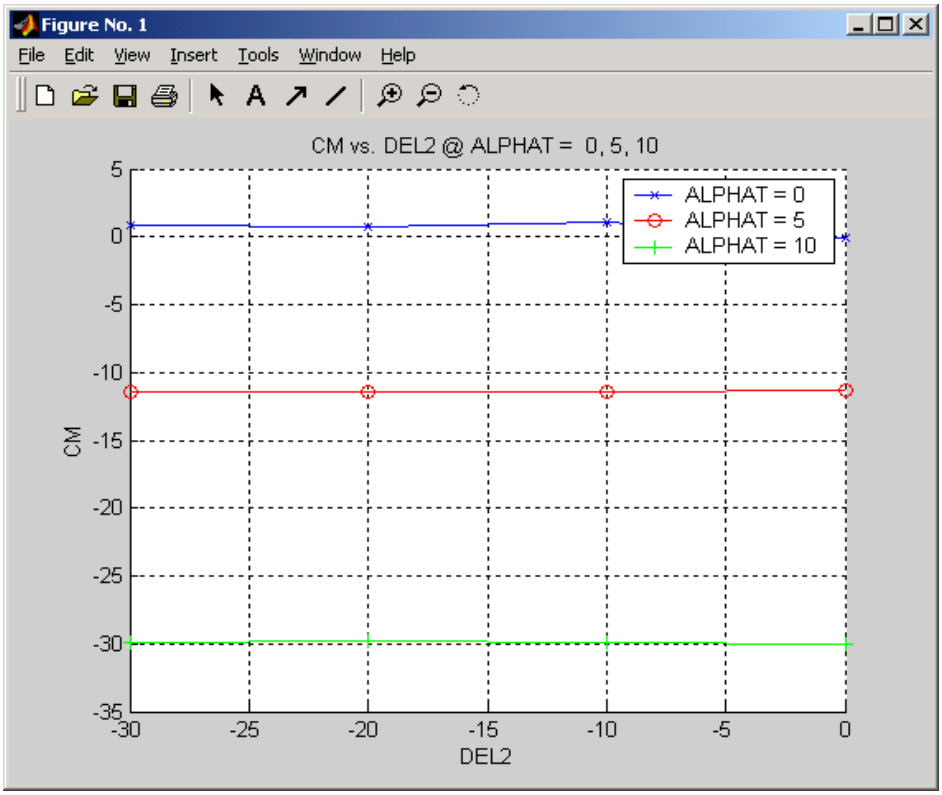


Figure 11. Typical Cross Plot, Pitching Moment Versus Fin Deflection Angle

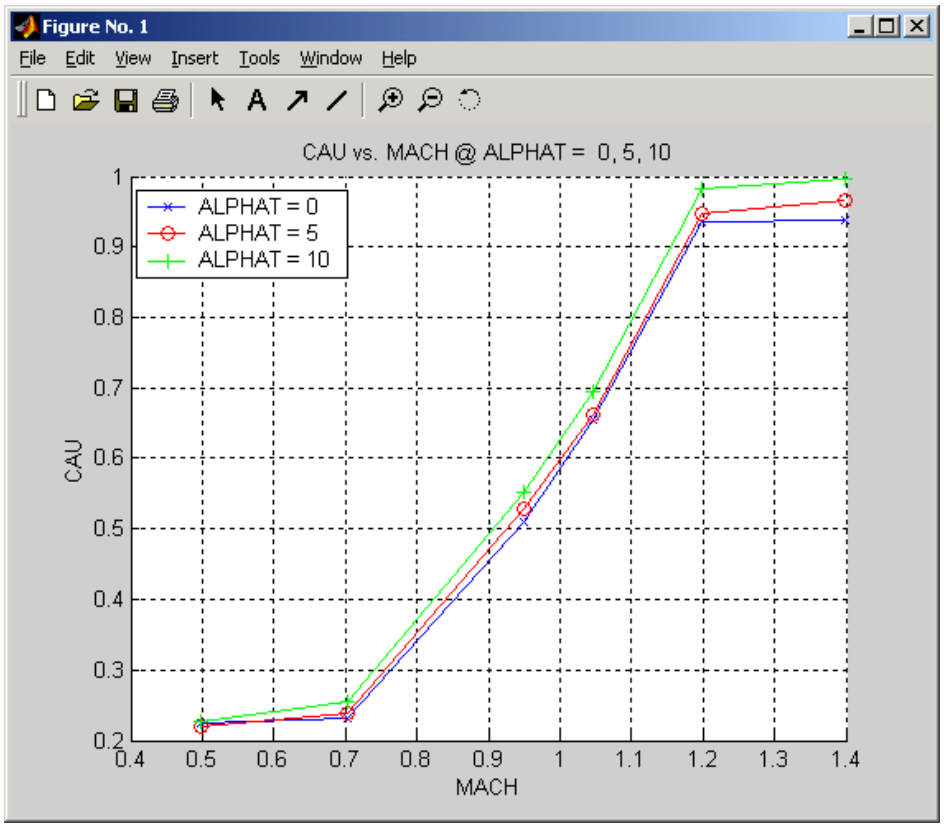


Figure 12. Typical Cross Plot, Forebody Axial Force Versus Mach

3. Carpet Plot

The Carpet Plot option allows the user to create plots that indicate the control authority and static stability of the airframe. Here, the user typically plots pitching moment versus normal force with lines of constant fin deflection angle and lines of constant AoA. Figure 13 shows the Carpet Plot variable input dialogue box, and Figure 14 shows the resulting plot. In this example, five AoA runs were selected for a given configuration and Mach number that had various fin deflection angles. The PIV is AoA and the Secondary Independent Variable (SIV) is fin deflection angle since it is the varying parameter across the set of runs.

The user must specify what values of the PIV are to be plotted; however the values of the SID are taken from the specified database parameter (in this case the Fin2 Deflection Angle). The values must be entered in the same way as the Cross Plot, separating each value by a space

The user can change the way the stability plot is annotated. The values for the SIV and the PIV can be shown on the plot at the ends of the lines and/or in the plot legend. The user can change these options by checking or un-checking the relevant checkboxes at the bottom of the Carpet Plot window. Both options are shown in the example plot in Figure 14.

Another common Carpet Plot is pitching moment versus drag or axial force, as shown in Figure 15.

By making use of the Shift Moment Reference Point (MRP) option located under Data in the menu bar, the user can quickly generate a set of stability maps for a variety of Center-of-Gravity (CG) locations.

The options for the plots produced are discussed in the Plot Options section.

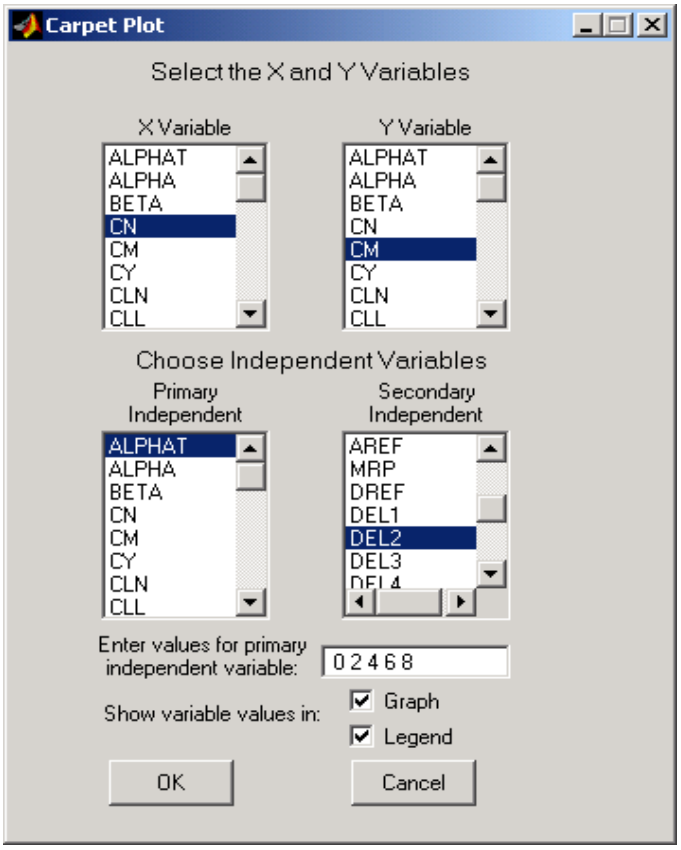


Figure 13. Carpet Plot Input Dialogue Box

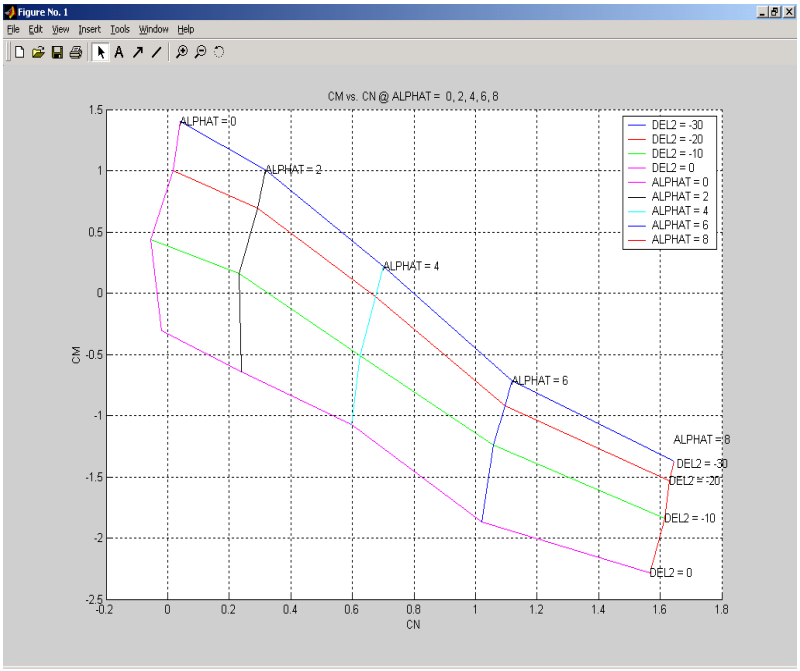


Figure 14. Typical Carpet Plot – CM-v-CN

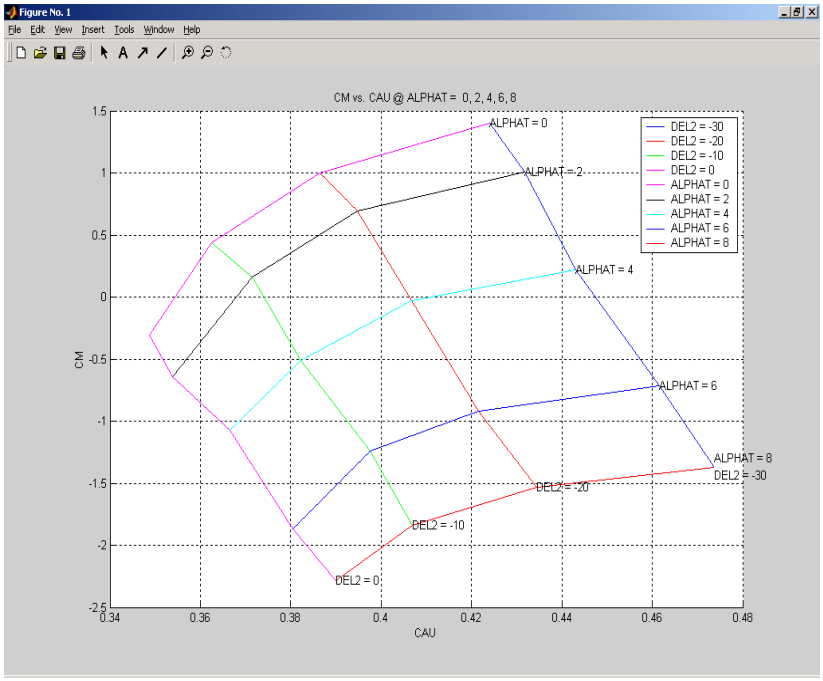


Figure 15. Typical Carpet Plot – CM-v-CD

4. 3-D Plot

The 3-D plot allows the user to view a coefficient as a function of another coefficient and a parameter. The window for this plot type is shown in Figure 16. The X variable is the PIV and the Y variable is the parameter or the SIV and the Z variable is the coefficient to be plotted. Choosing more than one Z variable will produce multiple plots, one for each Z variable chosen. An example of a plot is Normal Force versus AoA and Fin Deflection as shown in Figure 17. Another example is Pitching Moment versus AoA and Mach Number as shown in Figure 18.

The plot options are useful in seeing the 3-D graph in many different styles. The style shown in the first example below is Surf plot with Interpolated shading. The style shown in the second example below is Mesh plot (shading does not apply to Mesh plots). Other plot types that can be chosen are Surfc (Surf with contour lines on the XY plane), Surfl (Surf with shading as if a light source was shining on the surface), Meshc (Mesh with contour lines on the XY plane), Meshz (Mesh with elevation lines), Contour (2-D contour lines), and Contour3 (3-D contour lines at their appropriate Z elevations). Another option on the plot window is the colorbar. For most of the plot types, the color of the surface indicates the Z value. The colorbar displays the relationship between the Z value and the color. An example is shown on the plot in Figure 17.

All of the features offered in the MATLAB figure window as described under the Plot Options section are offered for the 3-D plot. One option that is extremely useful for the 3-D plot is the rotation button, which is the dotted line in a circle on the bar below the menus. Pushing this button allows the user to rotate the surface on the graph about all of the axes by clicking and dragging the mouse across the figure.

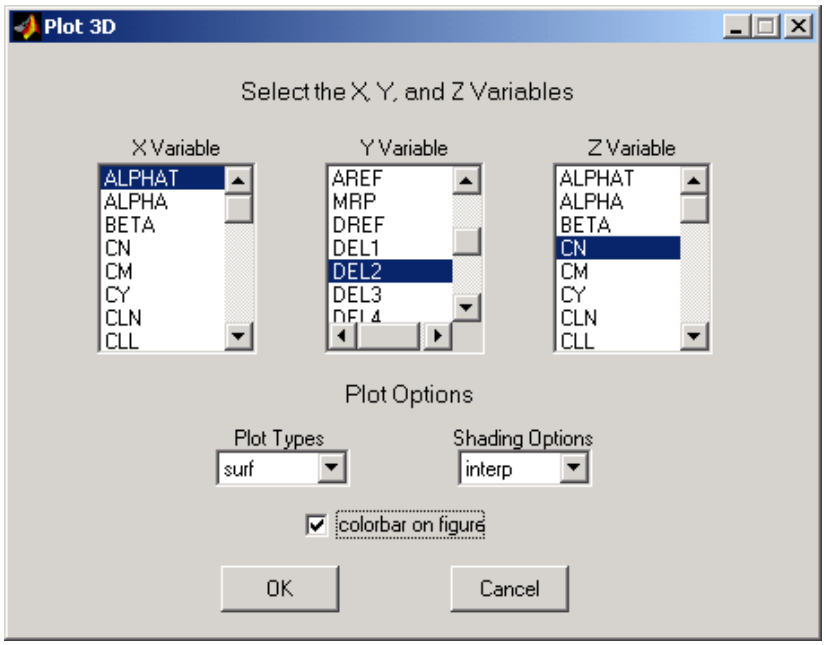


Figure 16. 3-D Plot Input Dialogue Box

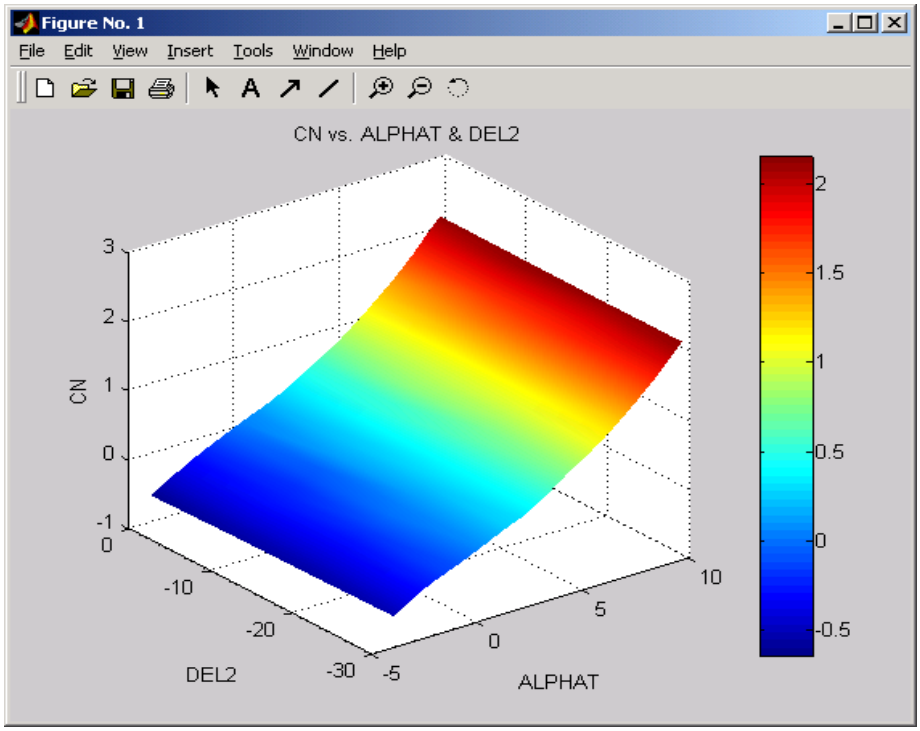


Figure 17. Typical 3-D Plot – CN-v-AOA-v-Fin-Delta

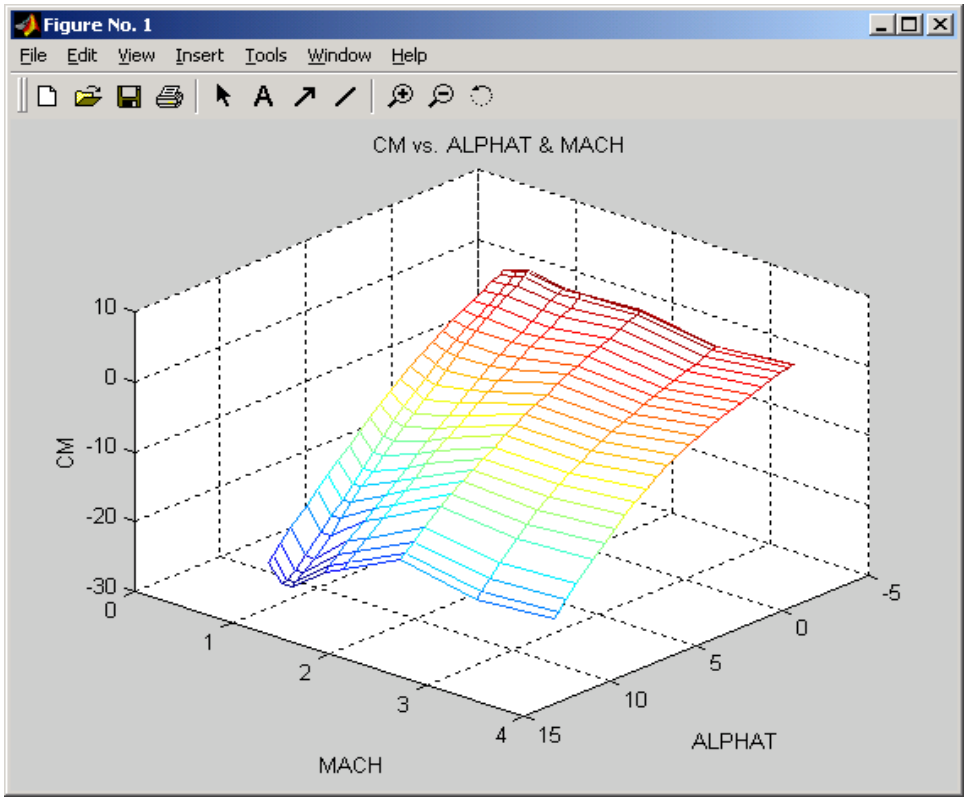


Figure 18. Typical 3-D Plot – CM-v-AOA-v-Mach

C. Export Data

The Export Data function allows the user to send data from the MATLAB/AAS program to Microsoft Excel. This function is useful in seeing the actual data values before and after manipulations have been performed on the data, since the MATLAB/AAS program has no ability to see the data in the program. Also, it is useful to have the data in Excel to be able to use it for other purposes.

The window for the Export to Excel function is shown in Figure 19. The figure offers many different options to the user. The first box allows the user to choose which runs in the program to send to Excel. The next two boxes allow the user to select which coefficients and parameters to see in each of the selected runs. The Other Data to Include box has a list of other values that can be entered in the Excel sheets. The coefficient and parameter indices are the index numbers for the coefficients and parameters as determined by the database format. Other options are to show the curve fit data, the biases, the differenced runs, the current axis of the data, and the current moment reference points. If data for those choices exists, they will be put in the Excel sheets.

The bottom of the Export window has options that allow the user to restrict the data that is exported. Choosing the Show Only option activates the list next to it. The user can choose to see just the data that was most recently plotted, the curve fit data, and/or the differenced runs. If any of these options are chosen, the program will not send the data for the individual runs. The last checkbox allows the user to choose whether or not to export even incremented data. If the user chooses to do so, the data that is exported so that the PIV is set to even increments and all of the rest of the coefficients are linearly interpolated at those points. This is done for all of the data, including the plotted data, the differenced data, as well as the data in the individual runs.

Note: An extra feature of the Export function is shown if there is a least squares curve fit on the data. If the curve fit data is exported, the coefficients of the least squares curve fit will appear above the columns of data. Also, below the columns of data, the Excel formulas for the polynomials of the curve fits will be under their respective columns of data. The formulas will be associated with an independent variable value, which will be below the column of the PIV and can be altered.

Also, if some of the runs have different numbers of coefficients and/or parameters, the missing data is filled with MATLAB Not-a-Numbers (NaN's). This data will show up as -1.#IND in the appropriate spaces in the Excel worksheets.

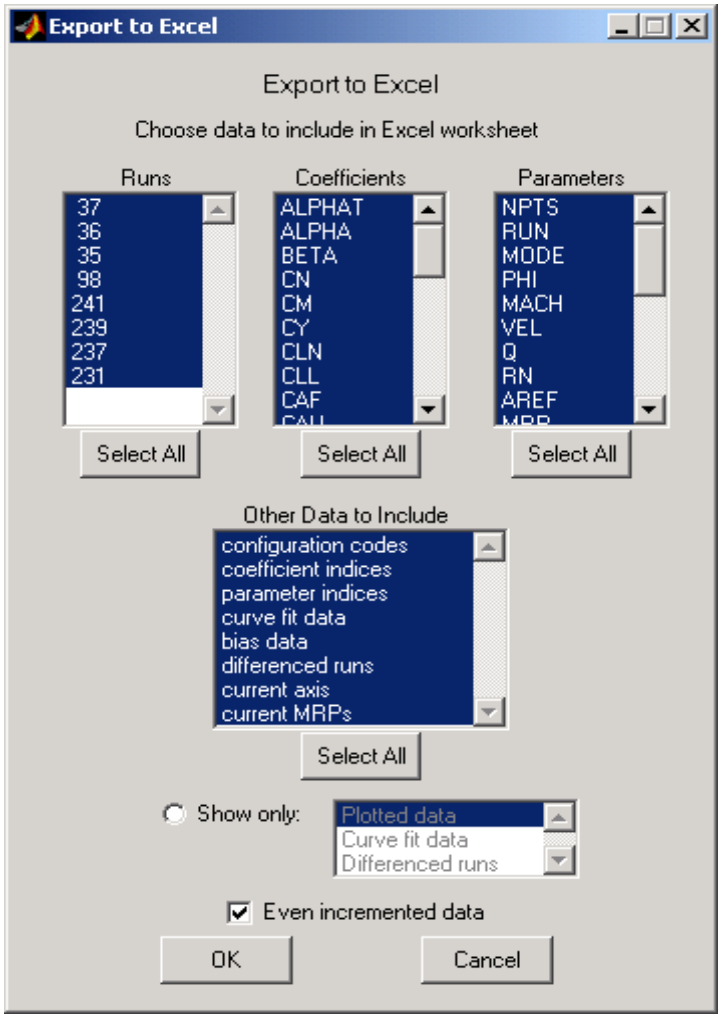


Figure 19. Export Control Dialogue Screen

D. Data Manipulation

MATLAB/AAS handles data manipulations in a different manner than Aerolab. For most of the manipulations the user chooses to do, those changes are performed immediately on the data. The exceptions to that are the Sort Data option and the curve fits, which are performed after the user chooses a plot or the export function. So, for example, if the user chooses to shift the moment reference point and then auto bias the data, those manipulations will be done immediately after they are chosen.

1. Sort Data

The Sort Data option is the first option under the Data menu. Choosing this option only toggles the Sort Option, and causes the Sort Data menu option to be checked or unchecked. The purpose of this option is to allow the user to choose whether to sort the data for plotting or for exporting to Excel. If this option is on, for each of the plotting options, the data will be sorted according to the primary independent variable. For the export function, the data in the runs will be sorted according to the sweep variable.

2. Change Coordinate Axis System

The Change Coordinate System operates on the primary 6-coefficient data (storage locations 8-13). Upon choosing the Change Axis option under the Data menu, the program will search for the parameter containing the current axis. If that parameter is not found or the current axis is unknown, the user will be warned that performing the incorrect axis transformation can lead to wrong data. If the user presses OK, the window in Figure 20 appears. If the current axis is found to be tunnel fixed axis, the window in Figure 21 appears. If the current axis is found to be body fixed axis, the window in Figure 22 appears. The equations for the coordinate conversion are shown in Figure 23.

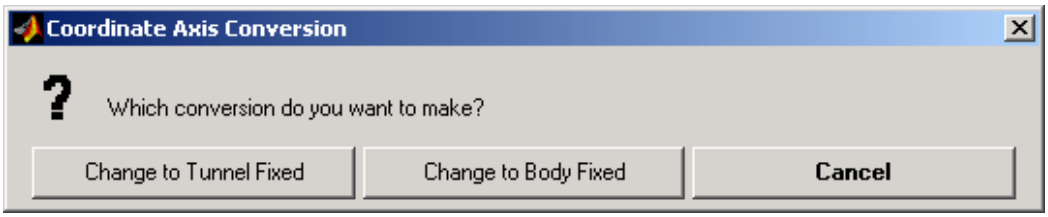


Figure 20. Coordinate Axis Conversion for Unknown Current Axis

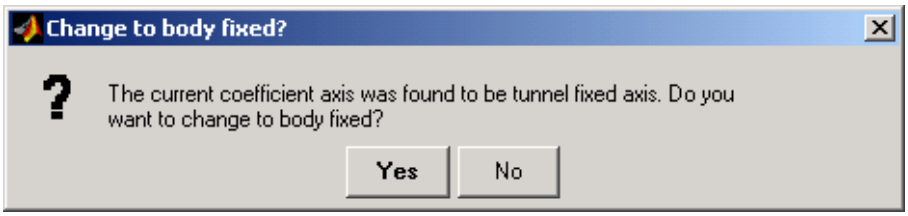


Figure 21. Coordinate Axis Conversion for Tunnel Fixed Current Axis

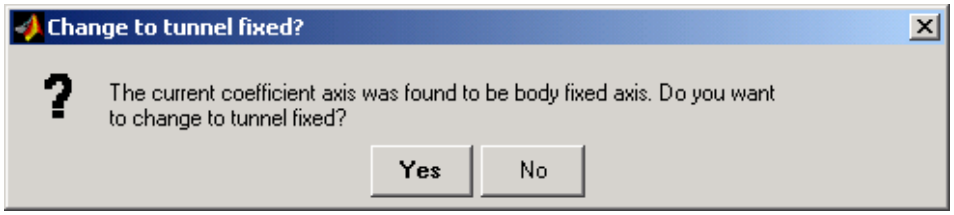


Figure 22. Coordinate Axis Conversion for Body Fixed Current Axis

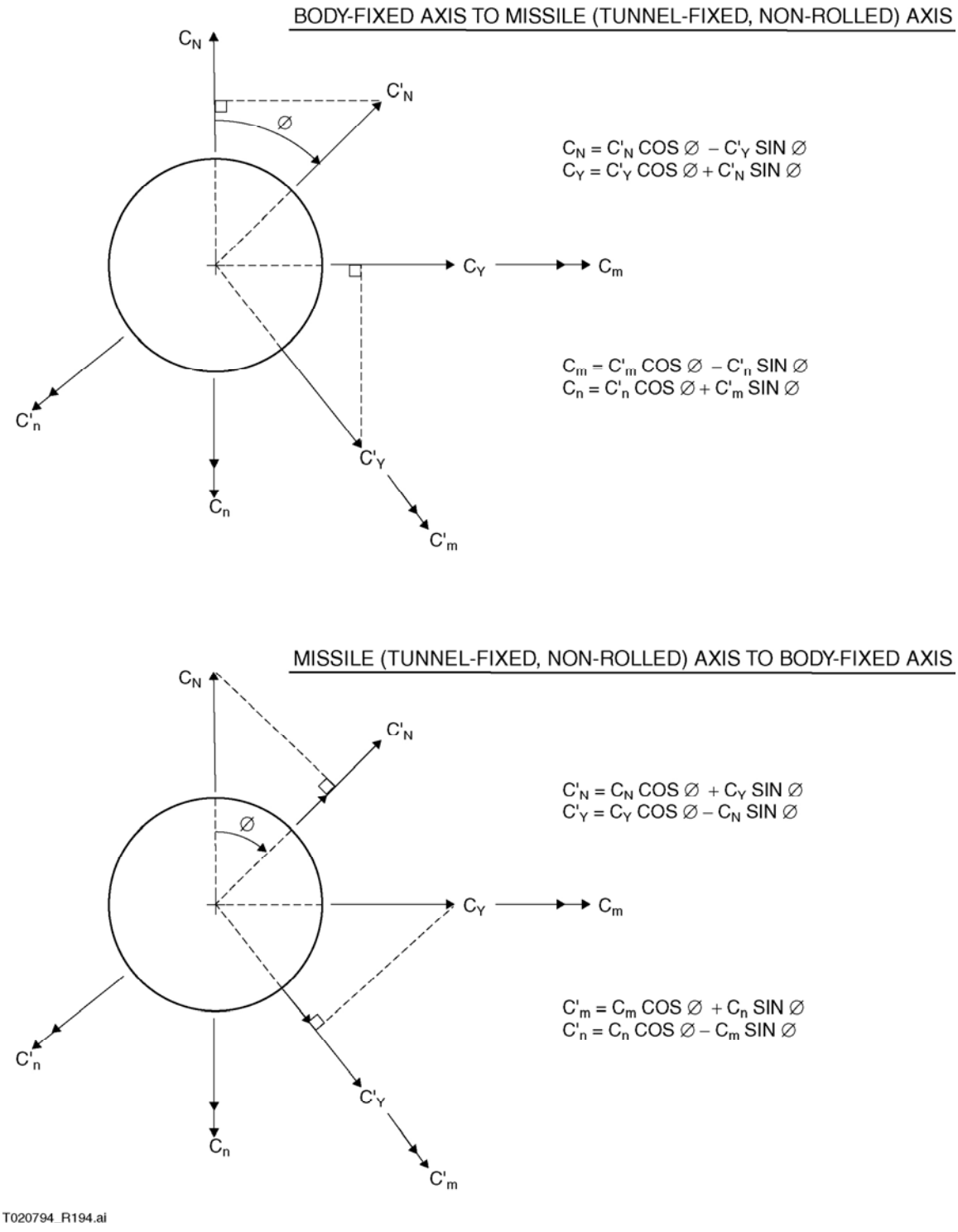


Figure 23. Coordinate Transformation Equations

3. Shift Moment Reference Point

The shift MRP option provided under the Data menu bar allows the user to change the moment reference point about which the original data set is reduced. This option in conjunction with carpet plot allows the user to create sets of stability maps that indicate how the stability and control for a given airframe change through the flight.

Shift MRP operates with all plot types. However, it only operates in the data primary 6-component moment data. Table 1 lists the equations used to perform the moment shift.

To Shift the MRP, the user must first choose the runs to change (Fig. 24). Once the runs are chosen, the Stored MRP box will show the MRP for the original data and the Current MRP box will contain the current MRP for the data. The two will be different if the user has already shifted the MRP. If the user has chosen more than one run and the runs have different stored or current MRPs, the relevant box will contain a dash instead of a number. The MRP that the user wants to change the selected runs to is entered in the box under the Enter New MRP prompt. Clicking OK will update the MRP. Pressing the Reset MRP Data button will change the current MRP for the selected runs back to the stored MRP.

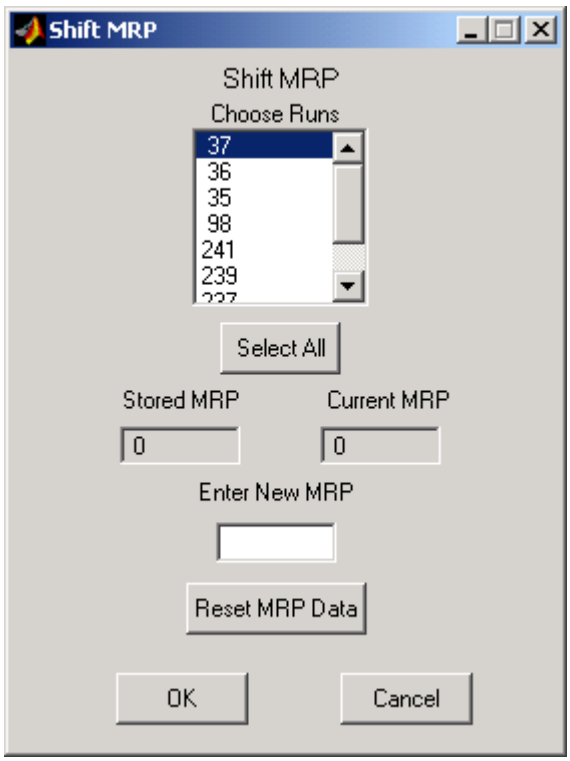


Figure 24. Shift MRP dialogue Screen

Table 1. Shift Moment Reference Point Transfer Equations

CM(new) = CM(old) + CN * (MRP(new) – MRP(old))
CLN(new) = CLN(old) + CY * (MRP(new) – MRP(old))
Note: MRP are given in calibers (typically body diameter)

4. Bias Data

Figure 25 presents the Bias control dialogue. This option allows the user to specify additive and/or multiplicative biases to all or individual runs and all or individual coefficients. In the example provided, a single run has been inputted four times and an incremental bias of 5 was applied to each run on the ALPHAT variable (which is the PIV). The resulting plot is presented in Figure 26.

Clicking on each of the coefficients and runs will show the biases that are currently on them in the additive and multiplicative boxes. If the biases are different for the selected runs and coefficients, the boxes will have a dash instead of a number. They can be changed by typing a new value in either of the boxes and pressing Enter or clicking outside of the box, or just by pushing the OK button. Pressing the reset button will reset the bias for the selected runs and coefficients.

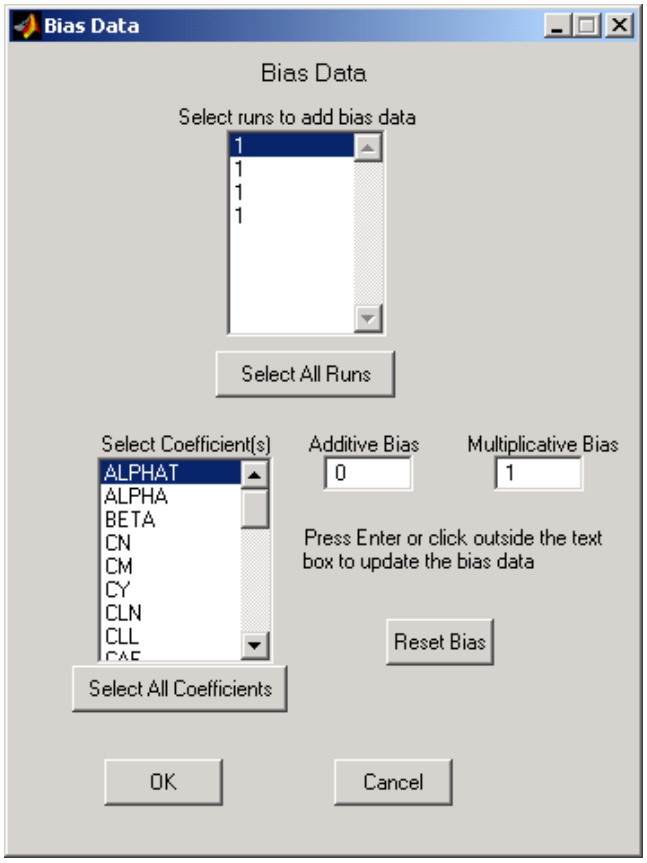


Figure 25. Bias Control Dialogue

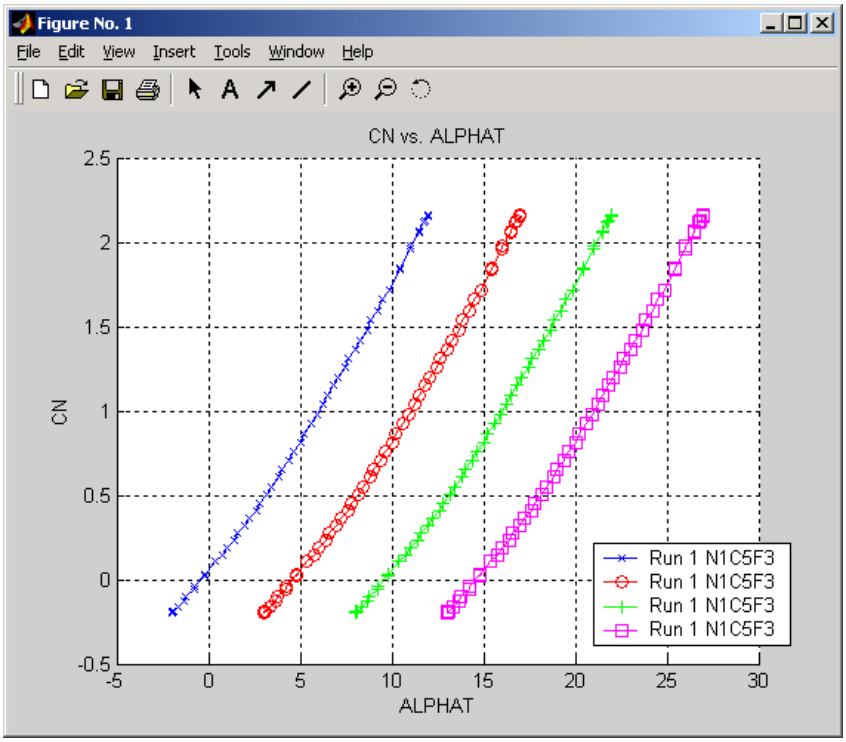


Figure 26. Simple-Bias Typical Plot

5. Auto Bias Data

Auto Bias is an option that allows the user to select one run containing zero-shift bias information and apply that bias information to all runs in a given set. For example, if the user has control increments for a symmetric missile configuration, then CN, CM, CY, CLN and CLL should pass through (0,0) for the delta = 0 case. In many instances, tunnel flow angularities and/or model asymmetries may introduce non-zero biases that must be removed. Auto Bias was designed to do just that. Figure 27 presents the Auto Bias dialogue window.

The user must first choose the base run that contains the zero-shift bias. The independent variable then must be chosen. The coefficients that the user wants to auto bias must be selected and added to the auto biased coefficients by pressing the -> button. If auto bias has already been applied, the current auto biased coefficient box will contain those coefficients that have been auto biased. The user can press the Delete or Delete All button to remove the auto bias on the coefficients.

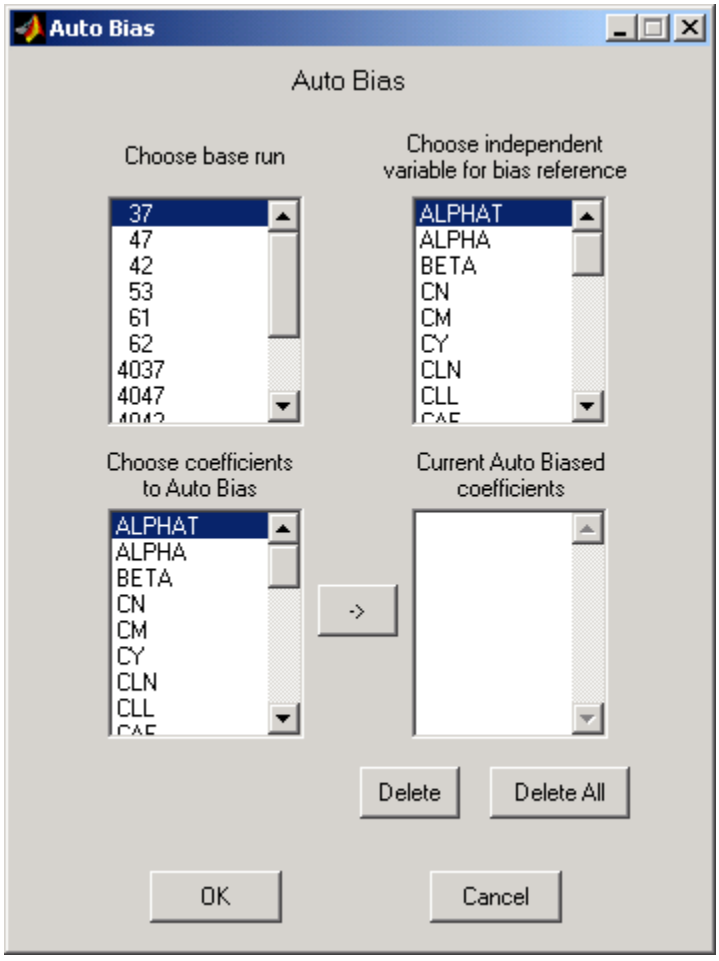


Figure 27. Auto-Bias Control Dialogue

6. Forced Delta Symmetry

Forced delta symmetry is used to impose symmetry at zero AoA on runs that have equal and opposite fin deflections angles. Forced delta symmetry, in conjunction with Auto-bias removal allow the user to quickly impose symmetry about zero. Once this is done, the user may use the Export option to view the corrected data in Excel. This data may then be cut and pasted into an external file for later incorporation into an aerodynamic model for 6-DOF simulations.

Figure 28 presents the dialogue control box for the forced delta symmetry option. Figure 29 presents a typical plot before applying forced delta symmetry and auto-bias, and Figure 30 presents the same plot after applying these options. The logic used to impose delta symmetry is presented in Table 2.

In the forced delta symmetry window, the user must first choose the base run(s). If there are two base runs present the user must choose the Two Base Runs option and choose the base runs from the pull down menus containing the runs. The user must also choose the independent variable and the coefficients to force delta symmetry on. Then, the user must add the run symmetry pairs, the runs with equal and opposite fin deflections. This is done by choosing the first run of the pair in the first window, choosing the second run of the pair in the second window, and then pushing the Add button. The run pair will then appear in the currently loaded symmetry pair box. The Delete and Delete All buttons can be used to remove one or more pairs. If delta symmetry has already been applied, the user can press the Remove Forced Symmetry button to remove all of the delta symmetry that has been applied to the data.

If the user wants to auto bias the data as well as force delta symmetry, the auto bias must be performed after the forced symmetry is applied. Otherwise the base run(s) may not be centered at zero.

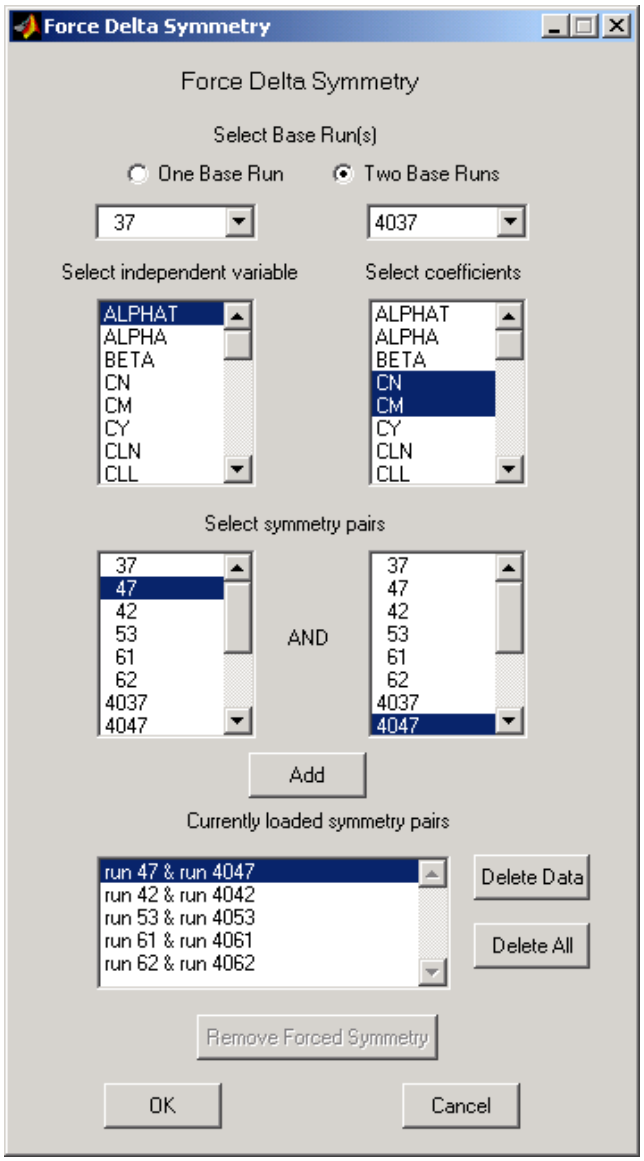


Figure 28. Forced Delta Symmetry Control Dialogue

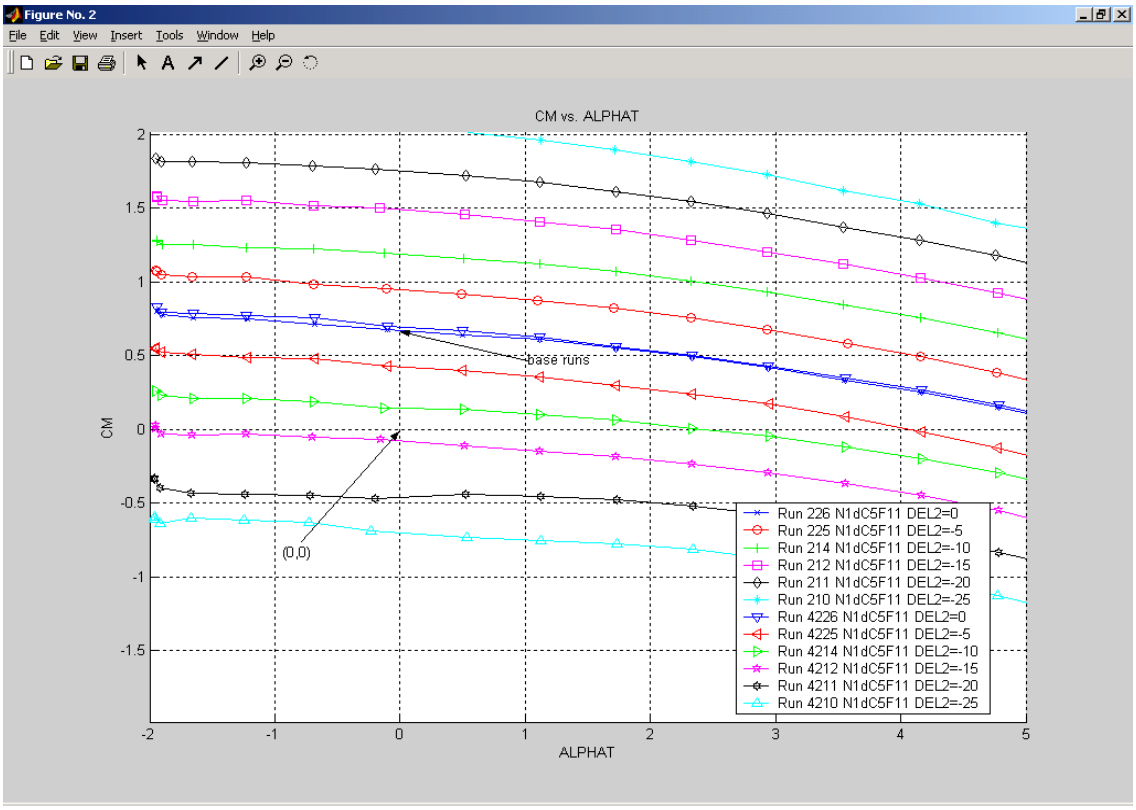


Figure 29. Plot Before Imposing Delta Symmetry and Auto-Bias

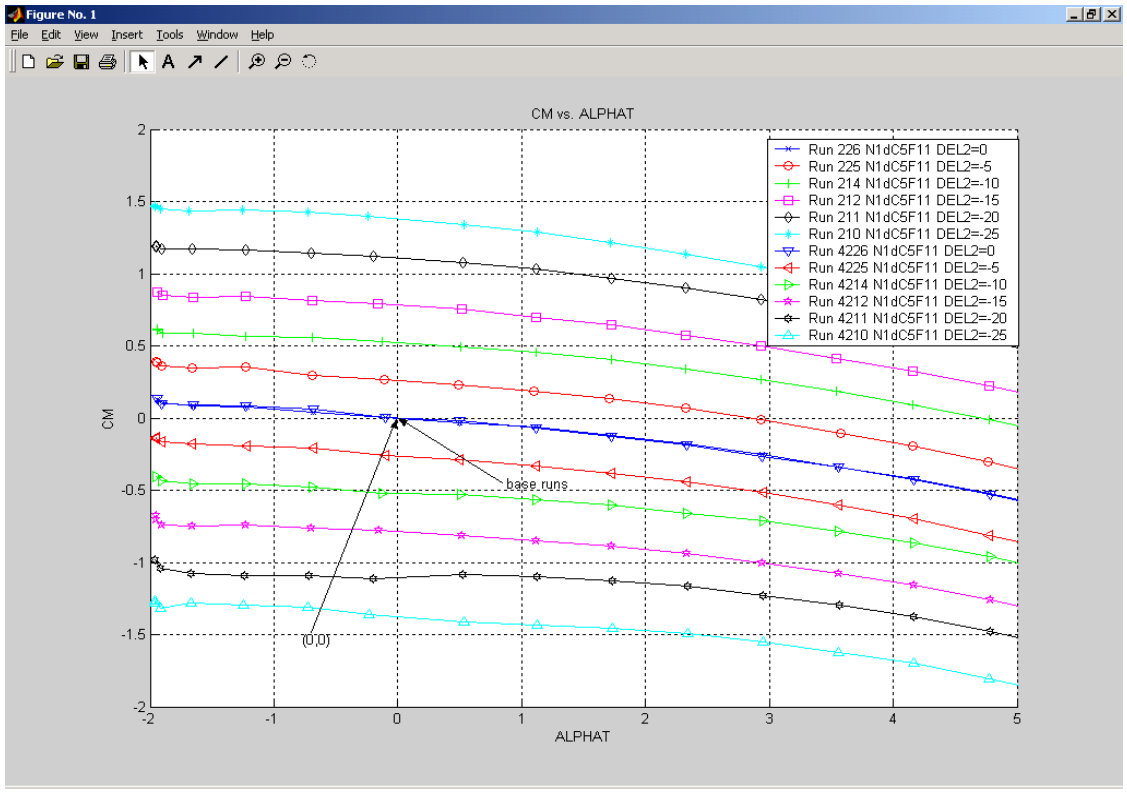


Figure 30. Plot After Imposing Delta Symmetry and Auto-Bias

Table 2. Forced Delta Symmetry Methodology

CX = coefficient that the forced symmetry is being applied to
Base Runs (δ=0): ZS = zero shift; distance of base runs from zero at Alpha=0 One Base Run: $ZS = CX_{br}(0)$ Two Base Runs: $ZS = (CX_{br1}(0) + CX_{br2}(0)) / 2$
Non-zero deflections: δ = non-zero fin deflection AD = average distance from the base run(s) B = bias added to the coefficient $AD = [\| CX_{+\delta}(0) \| + \| CX_{-\delta}(0) \|] / 2 - ZS$ If $CX(0) > ZS$ $B = AD - \| CX(0) - ZS \|$ If $CX(0) < ZS$ $B = \| CX(0) - ZS \| - AD$

7. Differencing Runs

Figure 31 presents the dialogue control for differencing, and Figure 32 presents a plot of a typical differenced curve. The 'Plot only differenced curves' allows the user to choose to display the differenced data by itself in any 2-D graph that is done.

To enter new differenced data, the user must choose the two runs from the runs list boxes and choose the operation from the pull down box in the middle and then push the Add button. Once this is done, the runs and the operation done to them will be added to the Currently loaded differenced data box. The operations that can be done to the runs include adding, subtracting, multiplying, and dividing. The Delete Data and Delete All buttons can be used to remove some or all of the differenced runs.

The differenced data is computed by creating even incremented data for the PIV and interpolating the rest of the data at those data points.

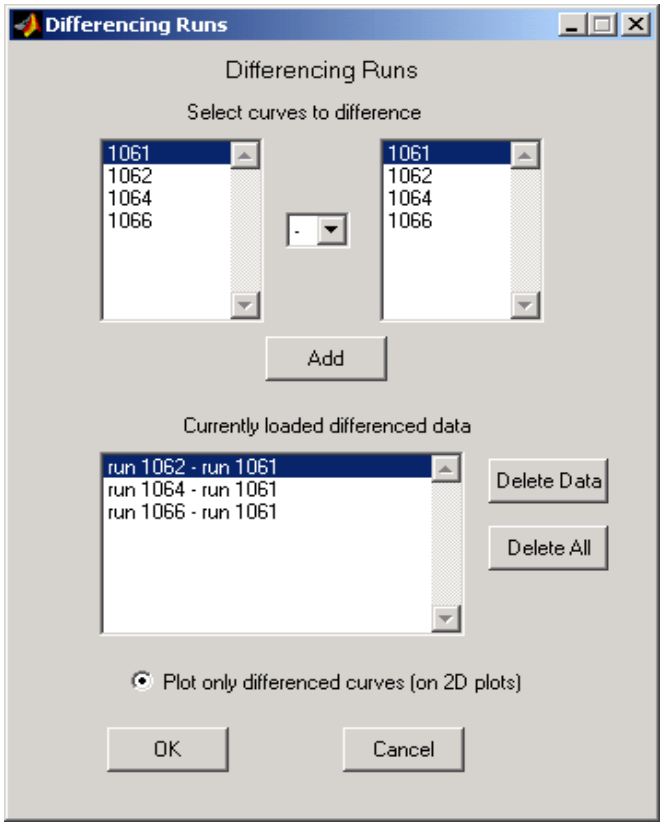


Figure 31. Differencing Control Dialogue

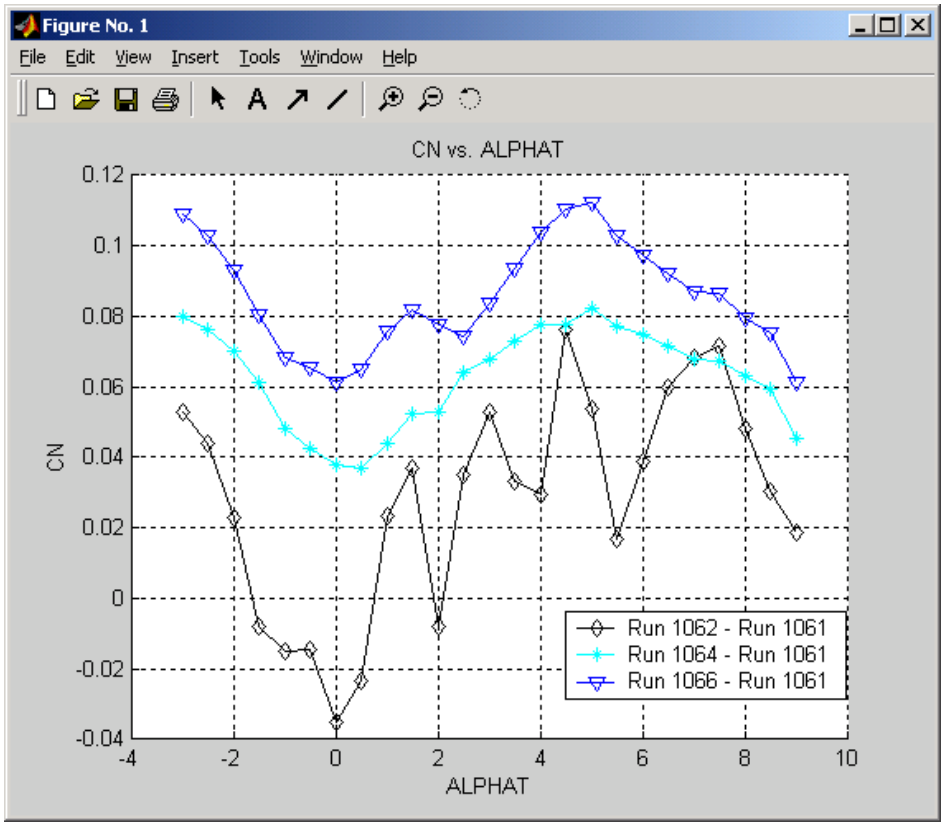


Figure 32. Differencing Plot

8. Curve Fitting

Listed below are the curve fitting options that are available to the user. The windows that are opened once the user chooses a curve fit under the Curve Fits folder in the Data menu are shown in Figure 33. The curve fit chosen by the user will only apply to the 2-D and Cross Plots. If the least squares curve fit option is used, once the graph (or graphs if there are multiple Y variables) appears, a window will open that allows the user to see the coefficients for the polynomial. An example of this window is displayed in Figure 34. The user can choose the curve and the polynomial term and the coefficient will be displayed in the coefficient box.

When the truncate at data end points option is chosen, the program will round up for the minimum and round down for the maximum. Because of this, some of the first and last data points may be omitted by the curve fit. In the example graphs shown below, this option was turned off and the maximum and minimum were entered manually.

Curve Fits:

- Least Squares
- Linear Interpolation
- Cubic Spline

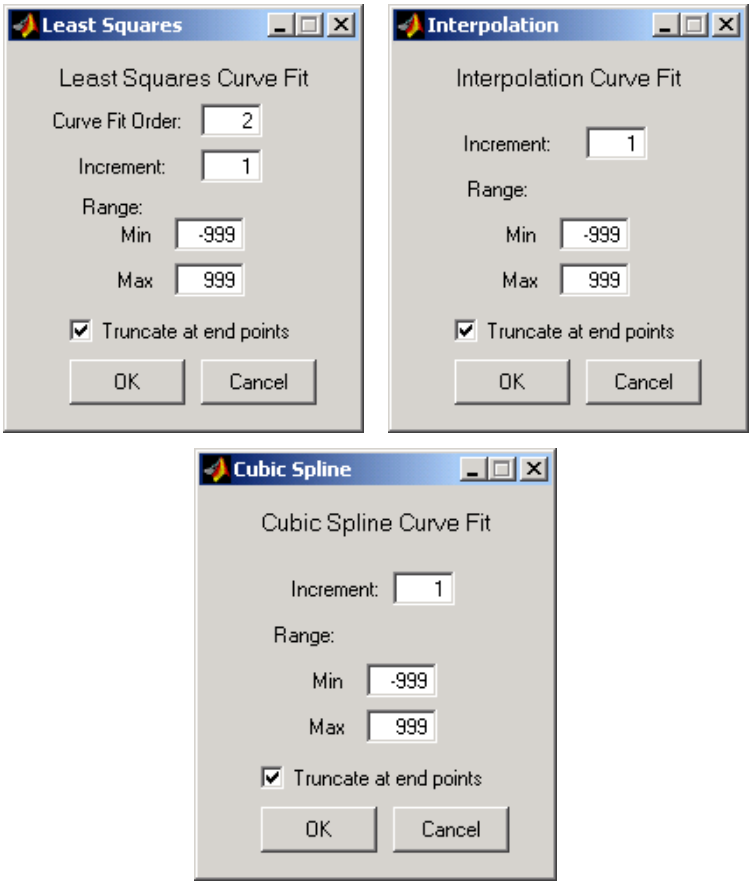


Figure 33. Curve Fitting Control Dialogue Screens

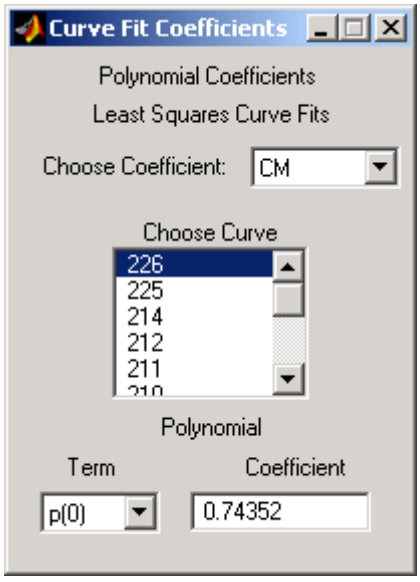


Figure 34. Polynomial Coefficients Window

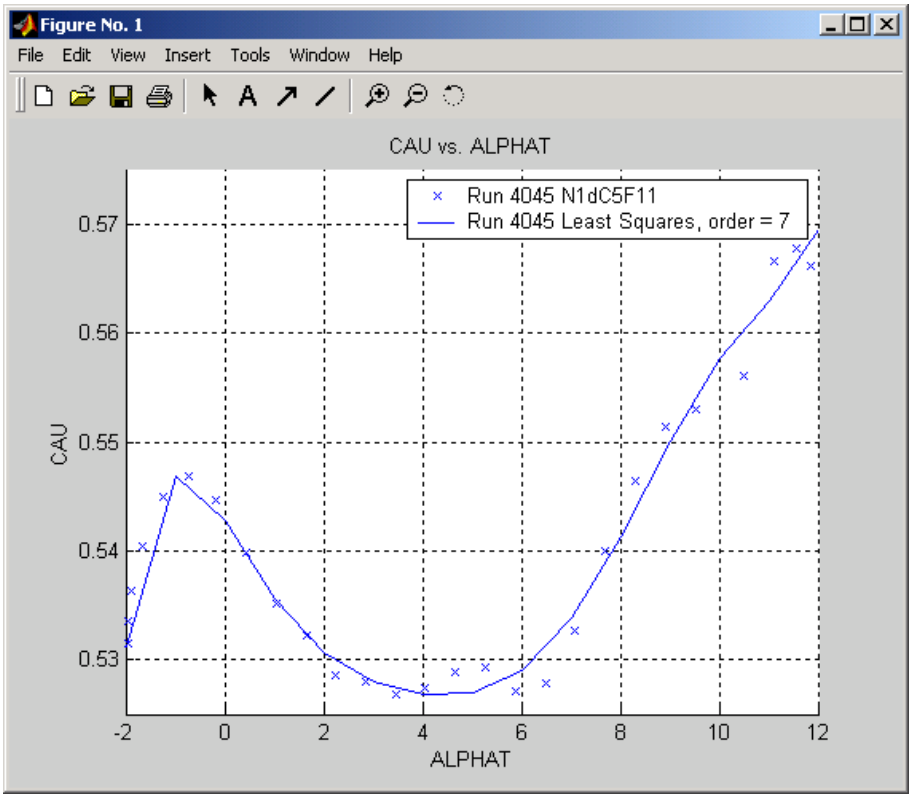


Figure 35. Sample Plot with Least Squares Curve Fit

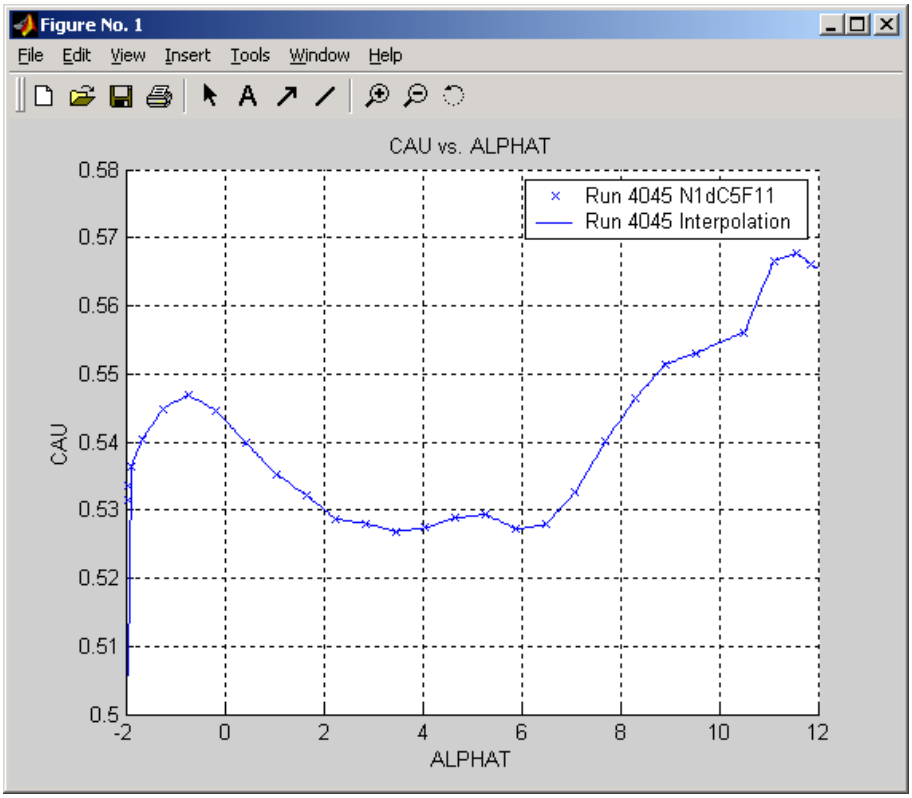


Figure 36. Sample Plot Using Interpolation

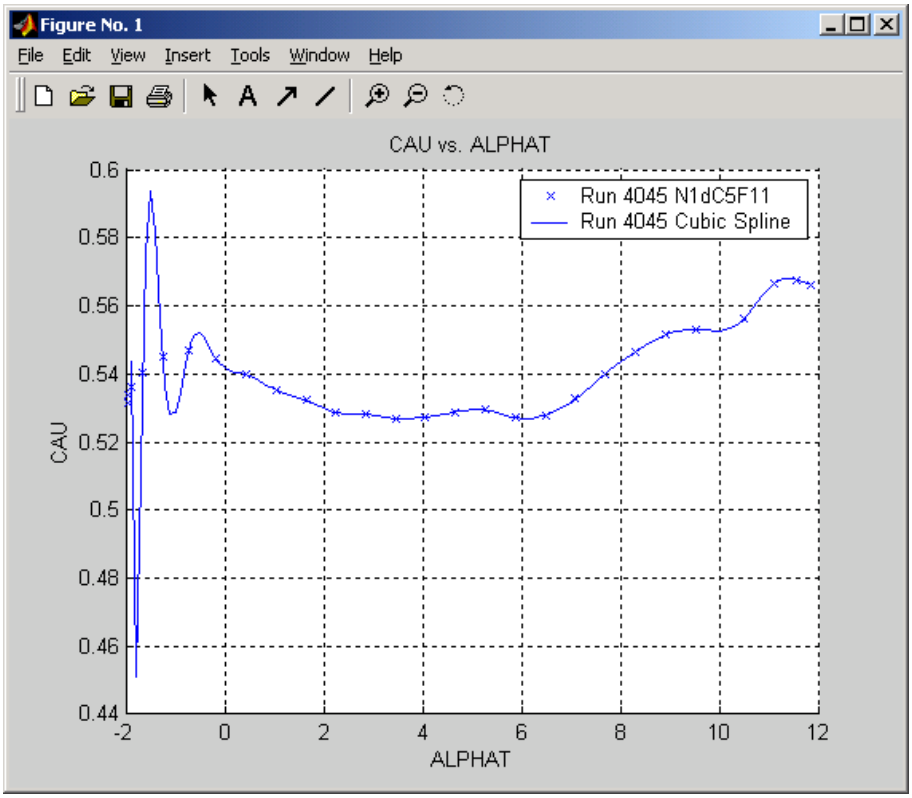


Figure 37. Sample Plot Using Cubic Spline

9. Original Data

The Original Data option under the Data menu allows the user to remove any data manipulations that have previously been done to the data. Upon pressing this option, the user is asked whether they really want to change the data back to the data from the database. If the user chooses OK, all of the manipulations on the data are removed.

E. Plot Options

Figure 38 presents a typical MATLAB/AAS X-Y plot. The figure window is a basic MATLAB plot window that is used whenever data is plotted in MATLAB. The figure has a number of useful features of which MATLAB has incorporated into the graph window. Some of these features are discussed here.

When the figure is first opened, the legend will appear as a white box in the upper right hand corner of the figure and it will contain each of the curves' color and symbol followed by the descriptor of the curve. If the plot is a X-Y plot, the constant coefficients, constant parameters, and configuration codes will also be in the legend, if those choices were made. This legend is mobile and can be moved simply by clicking on it and moving it with the mouse. To remove the legend, under the Insert menu click the Legend option. To put the legend back on the plot, click that option again.

Below the menu bar are some useful tools. The large A icon, when pressed, will allow the user to add annotations to the graph. The arrow pointing to the upper right allows the user to add arrows on the graph. The diagonal line allows the user to add lines to the graph. The magnifying glass with a plus sign allows the user to zoom in on any place on the graph while the magnifying glass with a minus sign zooms out. The dotted line in a circle is used mainly for the 3-D plots in that it allows the user to move the axes around to get different perspectives of the data.

The menus also have some useful functions. Under the File menu, the graph can be saved as a MATLAB .fig file by choosing the Save As option. This file can be opened later from the MATLAB command window. The graph can also be saved as other picture files as a .bmp, .jpg, .tif, or .emf file by choosing the Export option. The figure can also be copied and pasted by using the Copy Figure option under the Edit menu, but saving the graph as a picture file produces a much better image. The graph can be printed by choosing the print option under the File menu. The File menu also offers a Print Preview option.

The axes scales, labels, increments, and other properties can be changed by selecting the Axes Properties option under the Edit menu. Clicking on this option will bring up a window as shown in Figure 39. The scales, increments, and labels for the axes will be set to Auto, but clicking on the Auto checkboxes next to any of those properties will allow the user to change those values. In addition, the scale can be changed to logarithmic and the grids can be turned off. Also, under the Labels tab for the window shown in Figure 39 the labels for the X and Y axes and the title can be changed.

The line colors and symbols can also be changed from the figure window. Under the Edit menu, the Current Object Properties option allows the user to change the properties of any of the objects in the figure. Upon opening this window, clicking the pull down menu at the top will display a list of the objects in the figure. The best way to pick the different curves is to click on the legendlines, which contain the strings that are in the legend for each of the curves. An example of this window is shown in Figure 40. The line properties allow the user to change the style, thickness, and color of the line. The marker properties allow the user to change the style, size, and color of the line marker. The label for the legend can also be changed by typing in a new string in the Legend Label box at the bottom.

Other useful tools provided by the MATLAB graph window are a basic curve fitting tool and a statistics tool. These are found under the Tools menu. Although a curve fitting tool is provided in the MATLAB/AAS program, the Basic Fitting option can be useful. The window for this tool is shown in Figure 41. Clicking on any of the checkboxes in the list of curve fits will apply that curve fit to the curve that is selected in the pull down menu at the upper left corner of the window. Choosing a polynomial curve fit will cause the coefficients of the polynomial to be displayed in the center window. Checking the Show Equations option will display the polynomial equation on the graph. Choosing the Plot Residuals option will show the residuals. On the right side of the figure is a window that will display the values of the curve fit function for the values of the independent variable (X) as entered by the user in the text box.

The Data Statistics window is also useful in analyzing the plotted data. Choosing that option under the Tools menu will open a window with certain statistics displayed for the variables plotted. The minimum, maximum, mean, median, standard deviation, and range is displayed for the X and Y variables. An example of this window is shown in Figure 42.

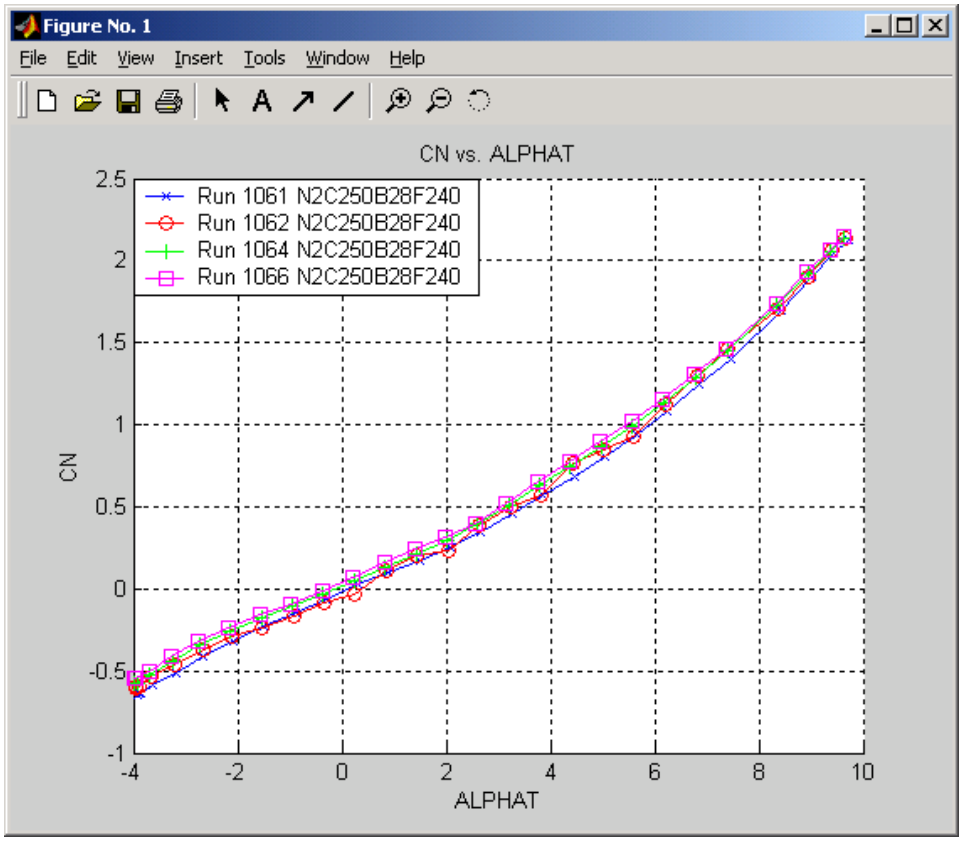


Figure 38. Typical MATLAB Plot

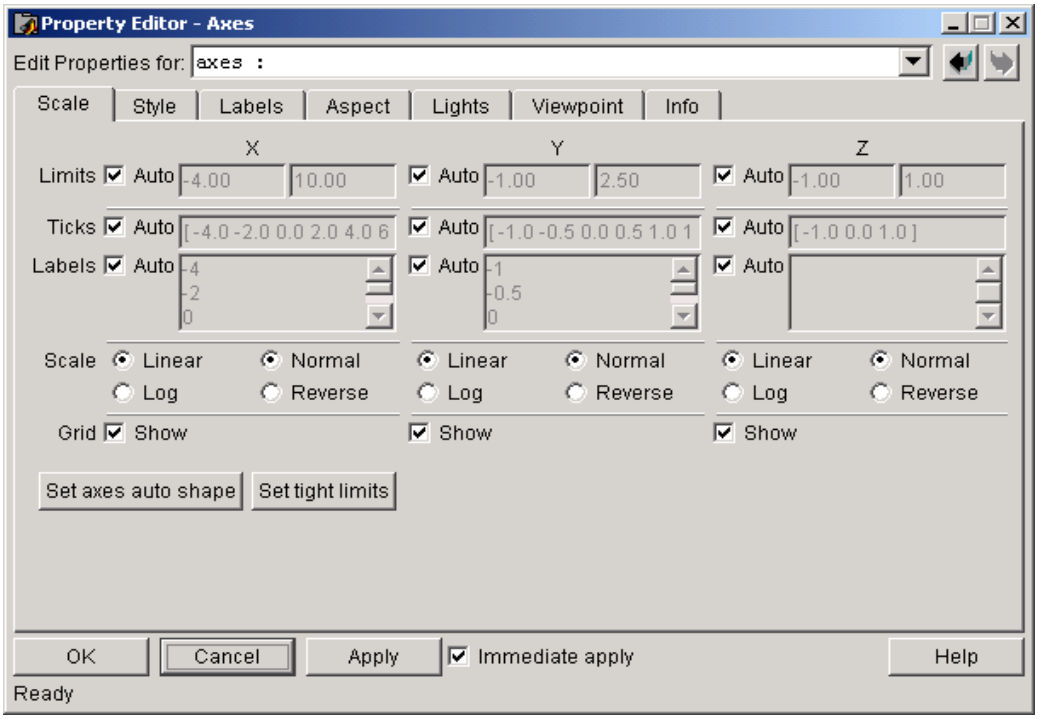


Figure 39. Axes Property Editor

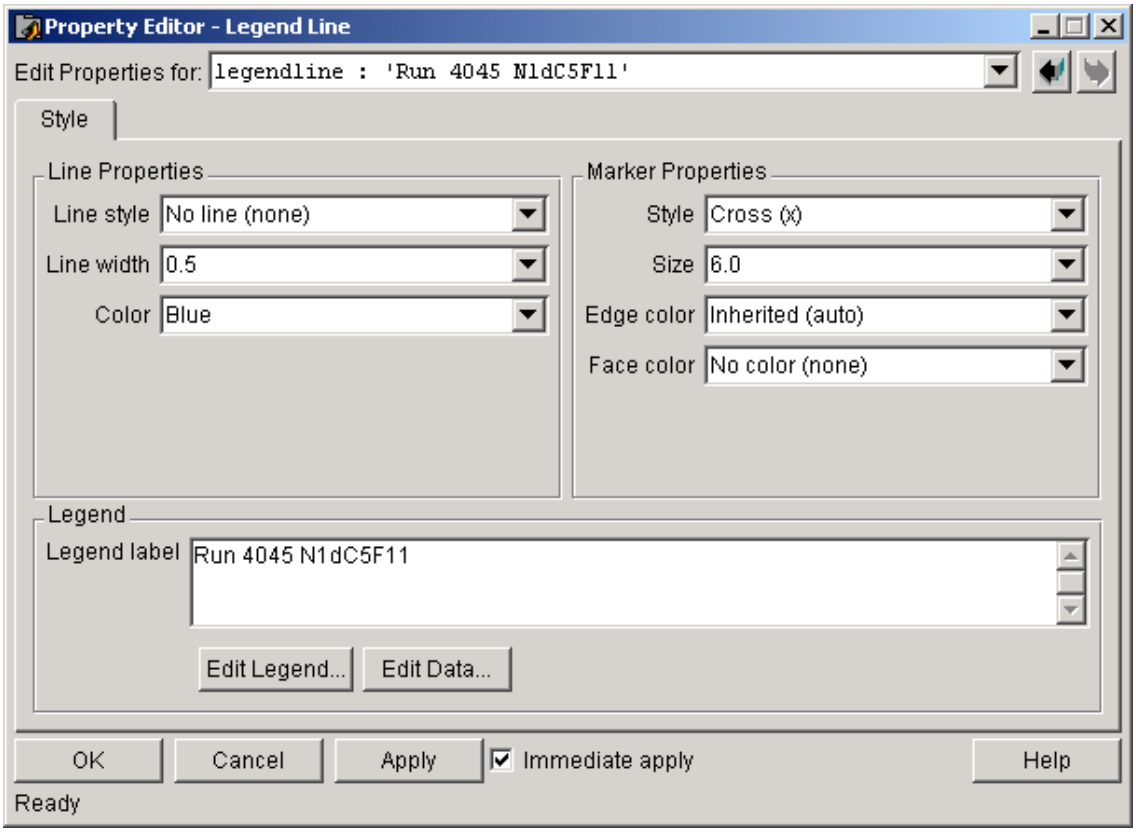


Figure 40. Lines Property Editor

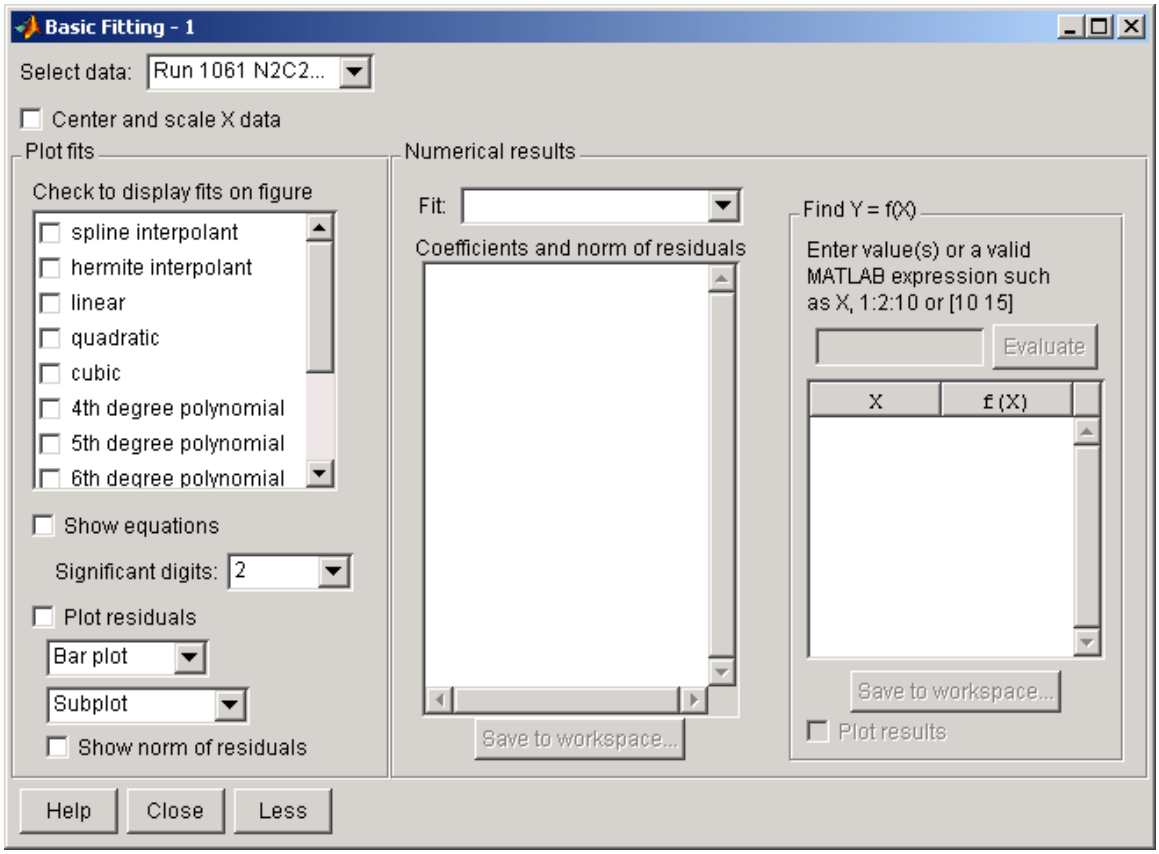


Figure 41. Basic Curve Fitting Window

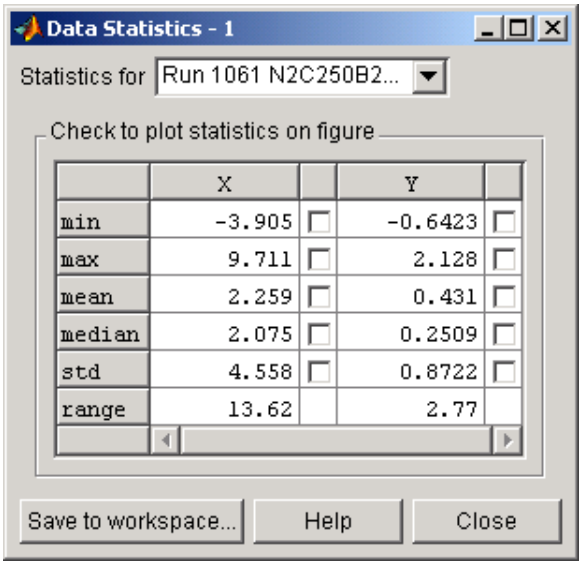


Figure 42. Data Statistics

F. Conclusions and Remarks

There are many areas that the MATLAB/AAS scripts can be improved, but because of time constraints, these revisions were not done. The run input method is restrictive in what data can be inputted into the program. Runs can be inputted from only one data file at a time. The user, therefore, is not able to select runs from different database files (.DAT files) to enter in MATLAB.

Also, the coefficient data is stored in MATLAB as a single 2-D array so that if some of the runs inputted have a lesser number of coefficients than other runs, the missing data is set equal to MATLAB NaN's. This does not affect the user except in what is exported to Excel, which is discussed in the Section II.C. In future versions of the scripts, it would be better to store the data in cell arrays.

In the scripts, the locations of the parameters and coefficients are assumed to be the same in every run. For example, if the user chooses ALPHA as the X variable in a plot, the location of ALPHA in each of the runs is assumed to be the same location as in the first run. If this is not true, the plot that is produced will be incorrect. In future versions, it would be better to use the coefficients and parameters indices to find the location of the chosen variables for each of the runs.

The differencing function of the program will perform the chosen operation to all of the coefficients in the two runs chosen. A more efficient means of performing the differencing would be to difference just the coefficients that are chosen in the different plotting functions.

III. PYTHON SCRIPTING STUDY

A. Introduction

The purpose of this program is to quickly take an output file from the AP program and reorganize the data into a format that is more understandable and compatible with AeroLab and other aerodynamic analyzing software.

This program was written in the Python scripting language and converted into a standalone Windows executable, APparse.exe. To see the original coding, the user needs to download the Python 2.2 program from www.python.org. Open the Python IDLE GUI and under the File menu, Open the file APparse.py.

B. Operation

The user must first double click on the APparse.exe file. A DOS window will pop up with the prompt: 'Enter name of the file to be parsed including the path name:' The user will then type in the full pathname and filename of the AP output file.

For example:

```
Enter name of the file to be parsed including the path
name (enter 0 to exit):  c:\APfiles\example.txt
```

Once the user types enter two lines will appear. They will say, for example:

```
c:\APfiles\example.out created
c:\APfiles\example.ou2 created
```

The user will then be given the option of parsing another file. A new prompt will come up that says: 'Enter name of the file to be parsed (enter 0 to exit)' followed by some instructions on how to enter the file name. If the user wants to exit, they can just enter 0 at the prompt. Otherwise, if the next input file has the same drive and directory as the last input file, only the filename needs to be entered. If the directory or drive is different, the whole path name needs to be entered.

For example:

```
Enter name of the file to be parsed (enter 0 to exit)
(if the drive and directory is the same as the last file,
just enter the filename):  example2.txt
```

The program will then look in the same directory as the last file (for this example, c:\APfiles) and will produce two output files for the current input file.

If the user choose to exit, the program will display 'Press any key to continue…'. Once a key is pressed program will end and the screen will disappear.

C. Output

Two files are created by the program for each input file. The first is the .out file and the second is the .ou2 file. They are both ASCII text files that can be opened in any Windows text editor, as Notepad.

1. .out File

The .out file is very similar to the original AP file. The only difference is that the angle of attack is the primary independent variable while the Mach number is the secondary independent variable. This means that under each of the variable headings, as 'Total Static Aerodynamics', 'Alpha' is the first variable in the list and it is varied for each line at a constant Mach number. The Mach number is then changed for the next section of the same variable headings.

The beginning of this file will contain the same header information as the AP file in exactly the same format. After that information, the Mach number for the next section is shown, and the data is divided into the same variable sets as the AP file with a varying angle of attack. The change in Mach number between the sections is shown by a dashed line across the page.

2. .ou2 File

The .ou2 file has the same data as the .out file but it is in a more raw form. This file can be used as the input to other programs, as AeroLab. The only headers for this file are in the first two lines. The first line contains the different variable set names, which are set above the first variable in each of those sets. The second line contains the actual names of the variables lined up over their values.

The data is once again organized such that angle of attack is the primary independent variable. But in this file, the Mach number is the second variable in the values. After the Mach number, each of the variable values is presented in succession. The first set of variables is set as Total Static Aerodynamics.

REFERENCES

1. Auman, Aerodynamic Analyzer System (Ver. 4.5). U. S. Army Missile Command, Redstone Arsenal, AL, Unpublished, April-May 1993.

2. Auman, User's Guide to the Aerodynamic Database. U. S. Army Missile Command, Redstone Arsenal, AL, TR-RD-SS-92-7, April 1992, Unclassified.

3. Landingham, A General Purpose database Program (GENDB) for the Perkin-Elmer Computer, Letter Report RD-SS-87-04, November 1986, Unclassified.

4. Landingham, Aerodynamic Data Base User's Guide. U. S. Army Missile Command, Redstone Arsenal, AL, Technical Report T-79-62, June 1979.

5. Auman; Tauchen, AeroLab User's Guide, U. S. Army Aviation and Missile Command, Redstone Arsenal, AL, Technical Report RD-SS-03-09, April 2003.

APPENDIX A
MENU BAR

Appendix A – Menu Bar

MATLAB/AAS Menu Bar

- File
- Data
- Runs

MATLAB/AAS Menu Bar

File

Input Runs

Save Run Set

Load Run Set

Data

Sort Data

Change Axis

Shift MRP

Bias Data

Auto Bias

Force Symmetry

Differencing

Curve Fit

None

Least Squares

Interpolation

Cubic Spline

Original Data

Runs

Choose runs

APPENDIX B
DEFINITIONS

Appendix B - Definitions

Definitions

PIV Primary Independent Variable: The coefficient/variable that was swept during a run (typically angle-of-attack, roll angle, side-slip angle, or Mach number)

SIV Secondary Independent Variable: Used when sets of runs are plotted together, and it refers to the single run parameter that varies across the run set (typically, fin deflection angle, roll angle, Mach number)

APPENDIX C
MATLAB/AAS INSTALLATION

Appendix C – MATLAB/AAS Installation

There are certain procedures that must be done first to the MATLAB/AAS scripts. First, the user must have MATLAB 6.0 or later for all of the functions in the scripts to work. There are 40 separate script files and one java .class file in the MATLAB/AAS program and all of those files must be copied into the same directory.

Also, since the MATLAB/AAS program uses a Java class to read from the binary databases, the MATLAB program must be directed to where that class file is. This is done by typing 'edit classpath.txt' at the MATLAB prompt, which opens that file in the MATLAB editor. At the bottom of the file, type the directory for the MATLAB/AAS files. This only needs to be done once.

Once MATLAB is first opened, the user must change the Current Directory to the directory that the files were copied into. This is done by pressing the '…' button at the top of the MATLAB command window. Then, the user must type in 'aas' at the prompt and the MATLAB/AAS main window will appear.

APPENDIX D
PYTHON SCRIPTING FILES

Appendix D – Python Scripting Files

```
# APparse.py

# Code written by: Jonathan Newby
#                  summer intern
#                  Dynetics, Inc.
#                  Missile Systems Department

# The purpose of this Python script is to take output files from the AP program and format them.

# Two files are produced by this script. The first is similar to the format of the AP output file
# except that the data is organized by angle of attack being the primary independent variable and
# the mach number being the secondary independent variable. The data is outputted in blocks of variables
# in the same way as the AP file. Headers separate the data blocks describing the type of variables.

# The second file produced by this script is a data file that is not separated into different blocks.
# The data is only separated by different mach numbers. So, the first block of data includes the angle of
# attack, the mach number, and all of the dependent variables. The first dependent variables are the
# total static aerodynamic coefficients, which is assumed to be the last set of variables in the AP file.


# This first function, sscanf, is used to take a list of numbers read from the input file and convert
# the list to floating point numbers.
def sscanf(a):
    b=[]
    for x in range(0,len(a)):
        b.append(float(a[x]))
    return b

# This function takes in a line from the file that contains a list of variables. The second input
# argument is the number of variable names that line contains. It returns a list that contains the
# variables separated out.
def varnames(vstr,numvar):
    vname = []
    for i in range(1,numvar+1):
        # The leading and trailing blanks are deleted by this command so the first character is a letter
        vstr=vstr.strip()
        wordcount=0
        for x in range(1,len(vstr)):
            if vstr[x]==' ':
                if vstr[x+1]==' ':
                    # The word is assumed to not have two blanks in succession inside it
                    break
                else:
                    # If two successive blanks aren't found, the number of letters in the word is incremented
                    wordcount=wordcount+1
            else:
                # If the next word in the line is not a blank, the number of letters in the word is incremented
                wordcount=wordcount+1
        # The variable name is picked out of the line inputted by using the number of letters in the name
        temp = vstr[0:wordcount+1]
        # The variable name is added to the list of variable names
        vname = vname + [temp]
        # The name is deleted from the line so the next name can be found
        vstr=vstr[wordcount+2:len(vstr)]
    return vname

# This is the function used to retrieve the pathname entered for the first input file.
# The purpose is that the user doesn't have to enter the drive and directory of
# the input file every time since the files are probably all in the same directory.
def pathname(filename):
    for x in range(0,len(filename)-1):
        # if the last character in the filename is a backslash, the path name is the current filename
        if filename[len(filename)-1]=='\\':
            path = filename
        # otherwise the last character is deleted so the character before that can be checked.
        else:
            filename = filename[:-1]
    return path

import string

# The input file is obtained from the user
filein = raw_input('Enter name of the file to be parsed including the path name (enter 0 to exit): ')
path = ''
# This loop is for the user to continue execution of the program on multiple files;
# entering 0 will break the loop and end the program
while 1:

# The input file is opened; if there is an error, the user is given the choice to try
# inputting the file name again. The 'if' statements are there to check if the path
# should be added to the inputted file name
    while 1:
        try:
            if filein=='0':
                break
            fin = open(filein,'r')
            break
        except IOError:
            print 'There was an error opening the file; the file may not exist'
            filein = raw_input('Enter the filename again (enter 0 to exit): ')
            if not path=='':
                if not filein=='0':
                    if not filein[1:3]==':\\':
                        filein = path + filein
```

```
# the output file is created by adding a '.out' extension to the input file
    if filein=='0':
        break
    fileout = filein[:-3]+'out'
    fout = open(fileout,'w')
# The first two lines of the file are copied directly to the output file
    header = fin.readline()
    fout.write(header)
    header = fin.readline()
    fout.write(header)
    fout.write('\n')
    while 1:
    # This loop keeps reading the lines until the word 'ANGLE' is found
        line = fin.readline()
        line = line.split()
        if len(line)>0:
            if line[0]=='ANGLE':
                break
# The 5th word in the line is assumed to be the value for the first angle of attack breakpoint
    alpha=[float(line[4])]
# The next few lines create the line that contains the reference diameter and writes it into the output file
    header = line[6]
    for x in range(7,len(line)):
        header = header + ' ' + line[x]
    for x in range(1,53):
        header = ' ' + header
    fout.write(header)
    fout.write('\n')
# The next loop reads the header information and copies it into the ouput file unless the word 'SUMMARY' is found in the
line
    while 1:
        line = fin.readline()
        sline = line.split()
        if len(sline)>0:
            if sline[0]=='SUMMARY':
                break
        fout.write(line)
# This just skips the lines until the first variable header
    for x in range(1,9):
        fin.readline()
# All the variables used for reading the data are initialized
    flines = 1
    mlines = 1
    fheader = []
    numvari = []
    vari = []
    data = []
    alphadata = []
    mach = []

# This next loop reads the data for the first angle of attack.
# The number of variable sections (flines), the names for the variable sections (fheader),
# the number of variables in each section (numvari), the names of the variables in the sections (vari),
# the number of mach breakpoints (mlines), and the actual mach values (mach) are recorded in this section.
# This information is used to reduce the time needed for execution of reading the data for the
# other angles of attack

    while 1:
        machdata = []
        line = fin.readline()
        line = line.strip()
        sline = line.split()
    # The lines in the file are read until the line is not blank.
    # Thus the split line would have a value (and have a length greater than 0)
        if len(sline)>0:
        # The TOTAL STATIC AERODYNAMICS section is assumed to be the last section;
        # Thus, the mach numbers and the number of mach lines is only recorded in that section

            if not sline[0]=='TOTAL':
            # If the section is not the last section (TOTAL) the number of variable sections is incremented
                flines = flines + 1
            # The fheader variable stores the line containing the names of the variable sections
                fheader = fheader + [line]
                fin.readline()
            # varline reads in the line containing the variable names in this section
                varline = fin.readline()
                fin.readline()
                fin.readline()
            # This line is assumed to be the first line of data in the section
                line = fin.readline()
                sline = line.split()
            # The number of variables is recorded as the number of values in this
            # line minus one (since the first value is the mach number)
                numvari = numvari + [len(sline)-1]
            # The varname function is used to take out the names of the variables in the line of variable names
                tempvari = varnames(varline,len(sline))
        # The first variable name is deleted since it is assumed to be 'Mach No.'
                temp = tempvari.pop(0)
        # The list of the variable names is added to the set of variable names, vari
                vari = vari + [tempvari]
        # the values of the variables are stored in mline by using the sscanf function on the split line
                mline = sscanf(sline)
                temp = mline.pop(0)
            # The data is added to the mach data set to be stored later
                machdata = machdata + [mline]
                line = fin.readline()
                sline = line.split()
            # If there is more than one mach breakpoint, the other lines are read in until a blank space or
            # the keyword 'NOTE:' is found
```

```
                if not line=='\n':
                    if not sline[0]=='NOTE:':
                        while 1:
                            mline = sscanf(sline)
                            temp = mline.pop(0)
                            machdata = machdata + [mline]
                            line = fin.readline()
                            if line=='\n':
                                break
                            else:
                                sline=line.split()
                                if sline[0]=='NOTE:':
                                    break
                fin.readline()
            # The mach data is added to the data for that angle of attack
                alphadata = alphadata + [machdata]
            else:
            # This is the same as the above, except for the 'TOTAL' section of variables.
            # Since this should be executed just once, the mach numbers are recorded here.
                fheader = fheader + [line]
                fin.readline()
                varline = fin.readline()

                fin.readline()
                fin.readline()
                line = fin.readline()
                sline = line.split()
                numvari = numvari + [len(sline)-1]
                tempvari = varnames(varline,len(sline))
                temp = tempvari.pop(0)
                vari = vari + [tempvari]

                mline = sscanf(sline)
                temp = mline.pop(0)
                mach = mach + [temp]
                machdata = machdata + [mline]
                line = fin.readline()
                sline = line.split()
                if not line=='\n':
                    if not sline[0]=='NOTE:':
                        while 1:
                            mlines = mlines + 1
                            mline = sscanf(sline)
                            temp = mline.pop(0)
                            mach = mach + [temp]
                            machdata = machdata + [mline]
                            line = fin.readline()
                            if line=='\n':
                                break
                            else:
                                sline=line.split()
                                if sline[0]=='NOTE:':
                                    break
                alphadata = alphadata + [machdata]
                break
# Once the data for the first angle of attack is finished, it is added to the total data
    data = data + [alphadata]
# This next section is similar to the first except that many of the characteristics of
# the data are not found and recorded but used to optimize the code operation.
    while 1:
        line = fin.readline()
        sline = line.split()
        if len(sline)>0:
            if sline[0]=='ANGLE':
                alpha.append(float(sline[4]))
                break
    endfile=0

    while endfile==0:
        alphadata = []
        machdata = []
        while 1:
            line = fin.readline()
            sline = line.split()
            if len(sline)>0:
                if sline[0]=='SUMMARY':
                    break

        for x in range(1,flines+1):
            while 1:
                line =fin.readline()
                sline = line.split()
                if len(sline)>0:
                    if sline[0]=='MACH':
                        break
            line = fin.readline()
            line = fin.readline()
            for y in range(1,mlines+1):
                line = fin.readline()
                sline = line.split()
                mline = sscanf(sline)
                mline.pop(0)
                machdata = machdata + [mline]
            alphadata = alphadata + [machdata]
            machdata = []
        data = data + [alphadata]
        while 1:
            line = fin.readline()
            sline = line.split()
            if len(sline)>0:
```

```
                if sline[0]=='ANGLE':
                    alpha.append(float(sline[4]))
                    break
            elif len(line)==0:
                endfile=1
                break
# Once the end of file is found the input file is closed.
    fin.close()
# This is the main section of writing the data into the output files.
# The mach no. is put at the top of each of the sections of mach number.
# This is followed by the header of the variable section.
# 'ALPHA' is put at the beginning of the variable names and each of the
# variable names are written. All of the names are right justified.
# The for the right justification is given as the name length plus two spaces
# unless the length of the name is less than 10, then the width is 10 plus two
# spaces. The data is written in much the same way, with the width determined
# in the same manner. A dashed line is used to separate the mach breakpoints.

    for imach in range(0,mlines):
        fout.write('MACH NO. = '+ str(mach[imach])+'\n')
        fout.write('\n')
        for ihead in range(0,flines):
            fout.write(fheader[ihead]+'\n\n')
            fout.write('ALPHA  ')
            for ivar in range(0,numvari[ihead]):
                if len(vari[ihead][ivar])<10:
                    width = 12
                else:
                    width = len(vari[ihead][ivar])+2
                fout.write(string.ljust(vari[ihead][ivar],width))
            fout.write('\n\n')
            for ialpha in range(0,len(alpha)):
                fout.write(string.ljust(str(alpha[ialpha]),7))
                for ivari in range(0,numvari[ihead]):
                    if len(vari[ihead][ivari])<10:
                        width = 12
                    else:
                        width = len(vari[ihead][ivari])+2
                # This is the call to the actual data value. The data list is organized such that
                # the first list index is the angle of attack breakpoint, the second is the
                # variable set, the third is the mach breakpoint, and the fourth is the actual
                # variable in the variable set. This number is converted to a string to be outputted.
                    num = str(data[ialpha][ihead][imach][ivari])
                    fout.write(string.ljust(num,width))
                fout.write('\n')
            fout.write('\n\n')

        fout.write('------------------------------------------------------------------')
        fout.write('------------------------------------------------------------------')
        fout.write('\n\n')
    print fileout+' created'
    fout.close()

# The first output file is closed and the second output file is opened. It is
# created with a .ou2 extension.

# The procedure is similar to the other output file but with some differences.
# The Mach No. is now included as a variable after alpha. All of the variables
# are now given in a single block, not separated into variable sections.

# Also, since the Reynold's number variable is included in multiple sections,
# the program checks for this variable until the first time it is found in a section.
# It is then included in the variable names and the values only once. Each successive
# time it is found, it is passed over.

# The first line in the file contains the header lines for the variable sets.
# These lines are put in the right order by determining the total width of the
# variable names in each set. A check must be done for the Reynold's number variable
# as in the other sections to line up the names in the correct place.

# The variable names are then written in the file with alpha and mach no. first.
# The Reynold's number name is only written once and then the rest of the variable names.

# The data is then written in the file. Successive loops are used to determine the
# correct value and write it in the file.

    fileout2 = fileout[:-1] + '2'
    fout = open(fileout2,'w')
    fout.write('                ')
    width = 0
    for ivar in range(0,numvari[flines-1]):
        if len(vari[flines-1][ivar])<10:
            twidth = 12
        else:
            twidth = len(vari[flines-1][ivar])+2
        width = width + twidth
    fout.write(string.ljust(fheader[flines-1],width))
    reyn = 0
    for ihead in range(0,flines-1):
        width = 0
        if vari[ihead][0]=='REYS. NO./MACH NO./FT.':
            if reyn==0:
                reyn=1
                fout.write('                        ')
            startv=1
        else:
            startv=0
        for ivar in range(startv,numvari[ihead]):
            if len(vari[ihead][ivar])<10:
                twidth = 12
```

```
            else:
                twidth = len(vari[ihead][ivar])+2
            width = width + twidth
        fout.write(string.ljust(fheader[ihead],width))
    fout.write('\n')

    fout.write('ALPHA  ')
    fout.write('MACH NO.  ')
    for ivar in range(0,numvari[flines-1]):
        if len(vari[flines-1][ivar])<10:
            width = 12
        else:
            width = len(vari[flines-1][ivar])+2
        fout.write(string.ljust(vari[flines-1][ivar],width))
    reyn = 0
    for ihead in range(0,flines-1):
        if vari[ihead][0]=='REYS. NO./MACH NO./FT.':
            if reyn==0:
                reyn=1
                fout.write('REYS. NO./MACH NO./FT.  ')
            startv = 1
        else:
            startv = 0
        for ivar in range(startv,numvari[ihead]):
            if len(vari[ihead][ivar])<10:
                width = 12
            else:
                width = len(vari[ihead][ivar])+2
            fout.write(string.ljust(vari[ihead][ivar],width))
    fout.write('\n\n')
    for imach in range(0,mlines):
        for ialpha in range(0,len(alpha)):
            reyn = 0
            fout.write(string.ljust(str(alpha[ialpha]),7))
            fout.write(string.ljust(str(mach[imach]),10))
            for ivar in range(0,numvari[flines-1]):
                if len(vari[flines-1][ivar])<10:
                    width = 12
                else:
                    width = len(vari[flines-1][ivar])+2
                num = str(data[ialpha][flines-1][imach][ivar])
                fout.write(string.ljust(num,width))
            for ihead in range(0,flines-1):
                if vari[ihead][0]=='REYS. NO./MACH NO./FT.':
                    if reyn==0:
                        reyn=1
                        fout.write(string.ljust(str(data[ialpha][ihead][imach][0]),24))
                    startv = 1
                else:
                    startv = 0
                for ivar in range(startv,numvari[ihead]):
                    if len(vari[ihead][ivar])<10:
                        width = 12
                    else:
                        width = len(vari[ihead][ivar])+2
                    num = str(data[ialpha][ihead][imach][ivar])
                    fout.write(string.ljust(num,width))
            fout.write('\n')
        fout.write('\n')
    print fileout2+' created'
    fout.close()
    # The pathname for the last input file is found to be used if the user only inputs the filename
    path = pathname(filein)
    print '\nEnter name of the file to be parsed (enter 0 to exit)'
    filein = raw_input('(if the drive and directory are the same as the last file, just enter the filename): ')
    # If the user did not choose to exit, the second character is checked for a :\ which would indicate that
    # a drive was given and so a new pathname was given. If not, the input file is the pathname plus the
    # given filename
    if not filein=='0':
        if not filein[1:3]==':\\':
            filein = path + filein
# This command just gives the user a chance to see what has happened before
# the program window is deleted.
raw_input('Press any key to continue...')
```

INITIAL DISTRIBUTION LIST

		Copies
Weapon Systems Technology Information Analysis Center ATTN: Ms. Vakare Valaitis 1901 N. Beauregard Street, Suite 400 Alexandria, VA 22311-1720		1
Defense Technical Information Center 8725 John J. Kingman Rd., Suite 0944 Ft. Belvoir, VA 22060-6218		2
AMSAM-RD,		1
AMSAM-RD-AS-I-RSIC		2
AMSAM-RD-AS-I-TP		1
AMSAM-L-G-I,	Mr. Dayn Beam	1
AMSAM-RD-SS,	Dr. Richard Amos	1
AMSAM-RD-SS-AT	Mr. Richard Kretzschmar	1
	Mr. Lamar Auman	1
	Mr. Richard Eric Kreeger	1
US Army Materiel System Analysis Activity ATTN: AMXSY-MP Aberdeen Proving Ground, MD 21005		1
Commander Naval Surface Warfare Center Dahlgren Division ATTN: Mr. Thomas Hymer (Code: G23) 17320 Dahlgren Road Dahlgren, VA 22448-5100		1
Missile and Space Intelligence Center ATTN: Mr. Jon Hall Redstone Arsenal, AL 35898-5500		1
The Boeing Company P.O. Box 516 ATTN: Mr. Brian Est (MC S500-3100)		1
	Mr. Joe Corce	1
St. Louis, MO 63166-0516		

INITIAL DISTRIBUTION LIST (CONT)

	Copies
Dynetics, Inc. P.O. Box 5500 ATTN: Mr. Michael Landers Mr. James Packard Mr. Brandon Hiller Mr. Mark Miller Huntsville, AL 35814	 1 1 1 1
KBM 4946 Research Drive ATTN: Mr. Ray Deep Huntsville, AL 35805	 1
Lockheed Martin Missiles and Fire Control - Orlando ATTN: Mr. John C. Lagow 5600 Sand Lake Road MP-605 Orlando Florida 32819-8907	 1
Lockheed Martin Missiles & Fire Control - Dallas P.O. Box 650003 ATTN: Mr. Tim Fennell (HSWT) Mr. Ed McQuillen Dallas, TX 75265-0003	 1 1
Lockheed Martin Missiles & Fire Control - Advanced Projects P.O. Box 1339 ATTN: Dr. Lance H. Benedict Manassas, VA 20108	 1
Veridian Engineering Division ATTN: Mr. Roman Paryz P.O. Box 400 Buffalo, NY 14225	 1
Allied Aerospace North American Trisonic Wind Tunnel ATTN: Mr. Richard Hughes 400 Duley Road El Segundo, CA 90245	 1

INITIAL DISTRIBUTION LIST (CONCL)

	Copies
Allied Aerospace Low Speed Wind Tunnel ATTN: Mr. David Sanford 3050 Pacific Highway San Diego, CA 92101	1
Raytheon 1151 East Hermans Road ATTN: Mr. David Corder Tuscon, AZ 85706	1
NASA, Langley Research Center 16 Victory Street ATTN: Mr. Tom Horvath (Mail Stop 408A) Mr. Mike Difulvio Hampton, VA 23681	1 1
Auburn University Aerospace Engineering Department 211 Aerospace Building ATTN: Dr. John Burkhalter Dr. Anwar Ahmed Auburn, AL 36849-5338	1 1
The Department of Aerospace Engineering and Mechanics The University of Alabama 205 Hardaway Hall, Box 870280 ATTN: Dr. Thomas A. Zeiler Dr. Charles L. Karr Tuscaloosa, AL 35487-0280	1 1
The University of Alabama in Huntsville (UAH) Propulsion Research Center Tech Hall, S234 ATTN: Dr. Brian Landrum Huntsville, AL 35899	1
Mr. Aaron Tauchen 116 Mykeys Way Huntsville, AL 35811	1