Transforming Aerodynamic Datasets into Parametric Equations for use in Multidisciplinary Design Optimization

By
Jeffery M. Scott

Advisor: Dr. John R. Olds

AE 8500, Fall 1998
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia

TABLE OF CONTENTS

TABLE OF FIGURES

1. ABSTRACT

This paper presents a method of transforming aerodynamic datasets generated in Aerodynamic Preliminary Analysis System (APAS) into parametric equations which may subsequently be used in a multidisciplinary design optimization (MDO) environment for analyzing aerospace vehicles.

APAS is an analysis code which allows the user to create a simple geometric model of a vehicle and then calculate the aerodynamic force coefficients of lift, drag, and pitching moment over a wide range of flight conditions. As such, APAS is a very useful tool for conceptual vehicle designs since it allows the force coefficients for a given design to be calculated relatively quickly and easily.

However, APAS suffers from an outdated user interface and, because it is tedious to generate a new dataset during each design iteration, it is quite difficult to integrate into an MDO framework. Hence the desire for a method of transforming the APAS output into a more usable form.

The approach taken and described in this paper involves the use of regression analysis techniques to accomplish the data transformation with three goals in mind. The first goal was to develop a parametric model for calculating the aerodynamic coefficients for a single unique geometry. The second goal was to extend this model to capture the effects of changes in vehicle geometry. The third goal was to write a Fortran program that would be capable of automatically carry out the regression analysis on a given APAS data set and produce the desired parametric equations. This paper presents the results and gives the model developed for analyzing a sample vehicle with a fixed geometry as well as the results of a sample vehicle with a variable geometry. The Fortran computer code is also given.

2. NOMENCLATURE

APAS	Aerodynamic Preliminary Analysis System
AR	wing aspect ratio
C_d	coefficient of drag
C_{do}	coefficient of drag at zero lift
C_l	coefficient of lift
C_{lo}	coefficient of lift at zero angle of attack
C_m	coefficient of pitching moment
DSM	Design Structure Matrix
HABP	Hypersonic Arbitrary Body Program
K	parameter relating C_d to C_l
M	Mach number
MDO	multidisciplinary design optimization
POST	Program to Optimize Simulated Trajectories
S	lift slope parameter
UDP	Unified Distributed Panel
α	angle of attack (degrees)
Λ	wing leading edge sweep angle (degrees)

3. INTRODUCTION

3.1. Background

Aerodynamic Preliminary Analysis System (APAS) is an industry-standard tool for calculating aerodynamic force coefficients at the conceptual design stage of aerospace vehicles. This tool, developed by NASA and Rockwell International for use in the design of the Space Shuttle, is useful for aerodynamic analysis, but due to it's highly interactive nature, it does not lend itself well to an iterative or optimized overall vehicle design process.

Aerodynamic analysis of a vehicle using APAS first requires the user to define a geometric model of the vehicle. This is done by specifying various parameters for individual components such as wings, fuselages, vertical tails, etc. For example, wings are defined by specifying planform area, aspect ratio, thickness-to-chord ratio, taper ratio, sweep angle and dihedral; fuselages by specifying length, cross-sectional area at various points along the longitudinal axis of the fuselage, and width-to-height ratio of each defined cross-sections. Required geometric data is entered manually via the keyboard with limited graphical interface. The process is prone to error and is difficult to duplicate or repeat accurately. In addition, changes to the vehicle configuration require a new model to be created in APAS because component geometric parameters cannot be altered at the keyboard once created.

Analysis in APAS is accomplished by defining specific flight conditions and vehicle attitude at which to calculate the aerodynamic force coefficients (typically lift, drag, and pitching moment). The required inputs are Mach number, angle of attack, flap deflection angle, altitude, skin friction coefficient and sideslip angle [1]. For advanced launch vehicles, the user typically defines a range of 10 - 15 Mach number and altitude pairs from liftoff to orbit that model the vehicle's expected flight path. Aerodynamic coefficients are then calculated at 8 - 12 angles of attack (or sideslip angles) for each flight condition. The result is a very large set of tabular aerodynamic data (over 100 data

points for each coefficient table). The entire aerodynamic dataset must be regenerated each time the overall vehicle geometry changes other than photographically.

Two separate analysis codes are used to perform the actual flow calculations. The first is Unified Distributed Panel (UDP) which is a vortex paneling code capable of analyzing subsonic and low supersonic flow conditions. The second code is Hypersonic Arbitrary Body Program (HABP) which, as the name suggests, is used for hypersonic flow conditions and is based on local surface inclination methods in which pressure coefficients are calculated.

APAS is inherently a difficult program to use due to its outdated user interface and cumbersome interactive format for creating and modifying a vehicle model. For each geometry change, regenerating the aerodynamic database might take an experienced user 4 - 6 hours. These characteristics make integrating APAS into a design optimization process wherein the vehicle geometry is allowed to change very difficult. Because of this, the valuable analysis capability of APAS is often under-utilized at the conceptual design stage of an aerospace vehicle.

3.2. Multidisciplinary Design Optimization Effort

In conceptual design, the typical order of execution of the various disciplines may be illustrated in a Design Structure Matrix (DSM), as shown in Figure 1, below. Here the DSM serves to organize the flow of information from one discipline to another. The "inputs" from propulsion to trajectory optimization, for example, are shown as lines above and to the right of the boxes, and the "outputs," or feedback, are shown as lines below and to the left of the boxes.

Overall vehicle layout and configuration is determined prior to or as a part of the aerodynamic design and analysis stage, whereupon propulsion, trajectory, aeroheating and weights and sizing are carried out in a tightly coupled, iterative process. Aerodynamics is typically left out of the loop (i.e. no feedback into the aerodynamics

discipline exists) once initial analysis has been performed due to the time constraints involved in recreating geometric models in APAS. Once the aerodynamic analysis has been done, the vehicle is only permitted to scale up or down photographically which ensures that the non-dimensional aerodynamic force coefficients remain unchanged throughout the iterative design process. Although this facilitates a rapid design process, it does not contribute to an overall optimized vehicle design since the aerodynamic coefficients are set at the beginning of the process and not allowed to vary as design knowledge increases.

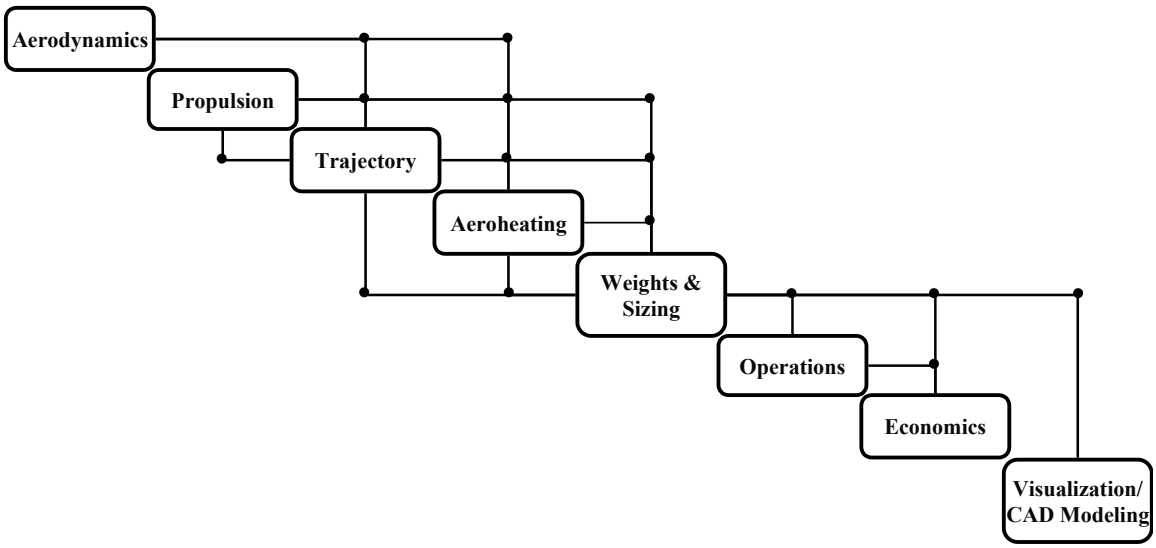

Figure 1 – Design Structure Matrix

In order to alleviate some of the problems associated with APAS, to make better use of its capabilities, and to facilitate its integration into an MDO environment, this research project was undertaken with several goals in mind. The first goal was to develop a method of transforming APAS datasets into a parametric model for calculation of the aerodynamic coefficients for a single unique geometry, and the second goal was to extend this method to capture the effects of changes in vehicle geometry. The third and final goal was to write a Fortran code that would automatically carry out the regression analysis for a given APAS dataset and produce the desired parametric equations.

Using the parametric equations to approximate a vehicle's aerodynamics and the change in aerodynamics with change in geometry, a gradient-based optimizer would be able to determine the optimum geometric parameters given a desired objective function. For example, the methods outlined below were used to capture the effect of changing the wing aspect ratio. An optimizer could then vary the aspect ratio and determine the force coefficients according to the parametric equations rather than requiring actual APAS analysis of several manually-created models, each with a unique aspect ratio. Extended to several geometric parameters, this would reduce the time and effort required to optimize a conceptual-level vehicle design.

One example of an optimization tool is NASA's Program to Optimize Simulated Trajectories (POST), a tool commonly used to perform the trajectory analysis and optimization for aerospace vehicles [7]. POST requires as inputs the APAS tables of lift, drag and moment at specific Mach numbers which it uses in calculating drag losses and lift forces along the vehicle's trajectory. In calculating the lift and drag forces throughout the trajectory, POST interpolates between explicit data points given in the aerodynamic tables as needed.

Given the parametric model developed here, the aerodynamic tables may be replaced with parametric equations. This will allow POST to internally calculate the force coefficients at run-time as opposed to repeatedly "looking up" values in a table, reducing CPU time. In addition, POST may use its internal optimization routines to vary the geometric parameters for which the parametric model was developed and calculate the optimum values of those parameters for a given objective function (such as minimum vehicle dry weight).

Use of the parametric model would thus provide a method of rapidly performing multiple design iterations in which vehicle geometry is changed, essentially bringing aerodynamics back "in the loop," i.e. including APAS analysis within a multi-disciplinary design optimization environment. Coupled with the Fortran code, multiple

design iterations could then be accomplished with greater speed and greater flexibility, and ostensibly producing a more highly optimized final vehicle design.

4. DEVELOPMENT OF MODEL

4.1. Test Vehicle

For this research effort, a representative aerospace vehicle designed by a team of graduate students at Georgia Tech was used as a test vehicle. This vehicle, named *Polaris*, is a rocket-powered reusable commercial launch vehicle designed for the space tourism market. It has a winged body configuration allowing horizontal take-off and landing, a fuselage fineness ratio of 8, swept wing (leading edge sweep angle = 55°) with an aspect ratio of 1.86 and theoretical planform area of 688 ft^2, and single vertical tail. A three-dimensional CAD model of the vehicle was produced using the I-DEAS solid modeling package. This model is shown in Figure 2.

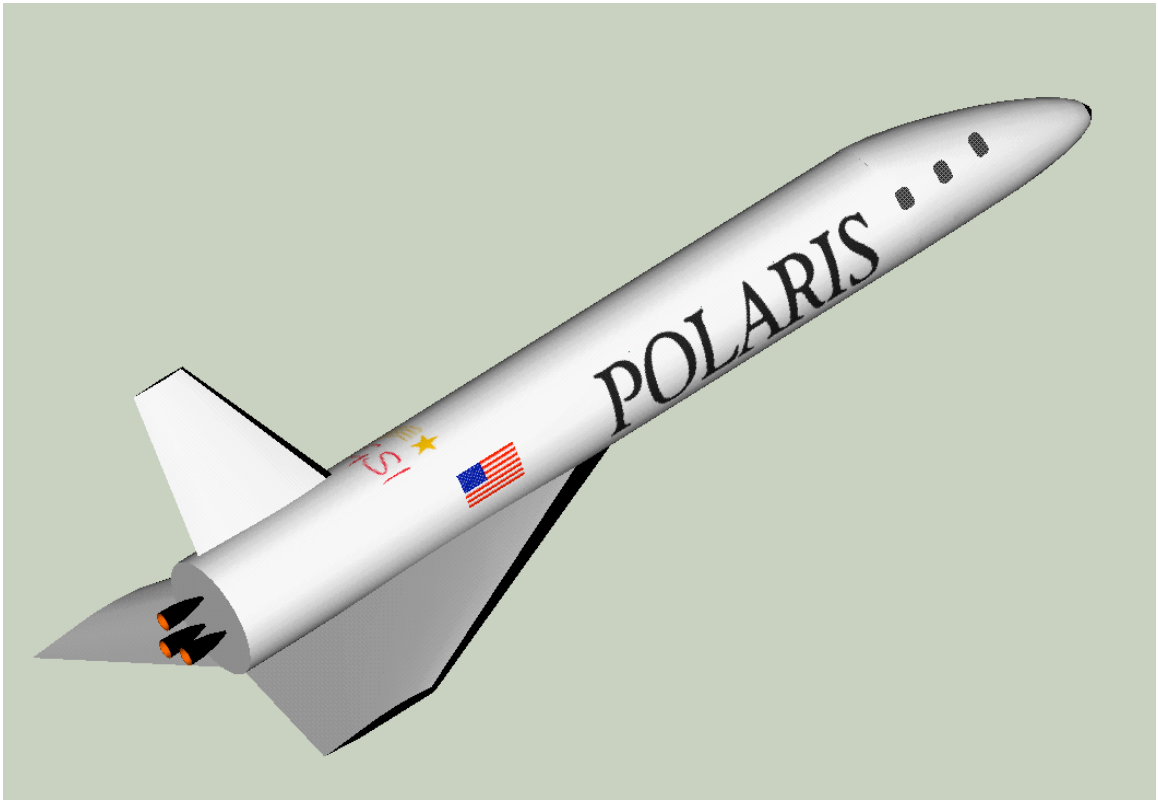

Figure 2 - Polaris launch vehicle

4.2. APAS Analysis

A simplified geometric model of *Polaris* was created in APAS. The fuselage was created using two components, the nose and the main fuselage. The wing was created as a single component while the vertical tail was not modeled since it contributes little to the drag, nothing to lift, and since lateral stability and control was not a concern for this project. The APAS model is shown in Figure 3.

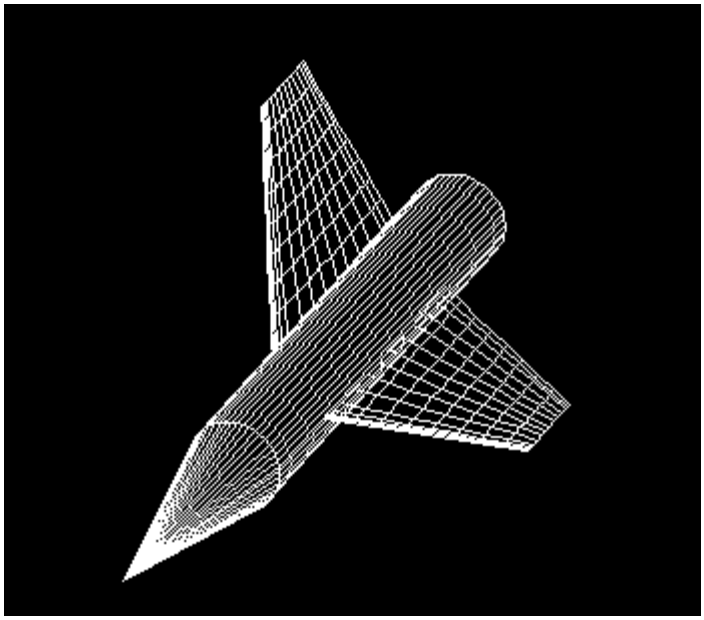

Figure 3 – APAS Model

Aerodynamic analysis for the test vehicle was performed over a sub-orbital trajectory. The flight path was assumed to follow a trajectory from sea level to an altitude of about 250,000 feet, from Mach 0.3 at liftoff to Mach 18.0 at altitude. APAS was used to calculate the force coefficients for fourteen particular Mach numbers in the specified range, at nine angles of attack for each Mach number, ranging from -15 degrees to +15 degrees. Thus a total of 126 data points for each force coefficient were obtained for use in the basic model. From this dataset, parametric equations were developed to calculate lift and drag as a function of Mach number.

Additional models of *Polaris* were generated in APAS in which the wing aspect ratio was varied. In this case, analysis was performed for models with wing aspect ratios of 1.5, 2.0 and 2.5 (in addition to the 1.86 AR wing). In total, 504 data points were generated for each force coefficient in order to carry out the regression analysis.

The appropriate form of the equations used to calculate lift and drag coefficients may be seen by plotting C_l vs. angle of attack and C_l vs. C_d (drag polar). These plots are

shown for the test vehicle at three different Mach numbers: 0.3, 1.5 and 8.0.

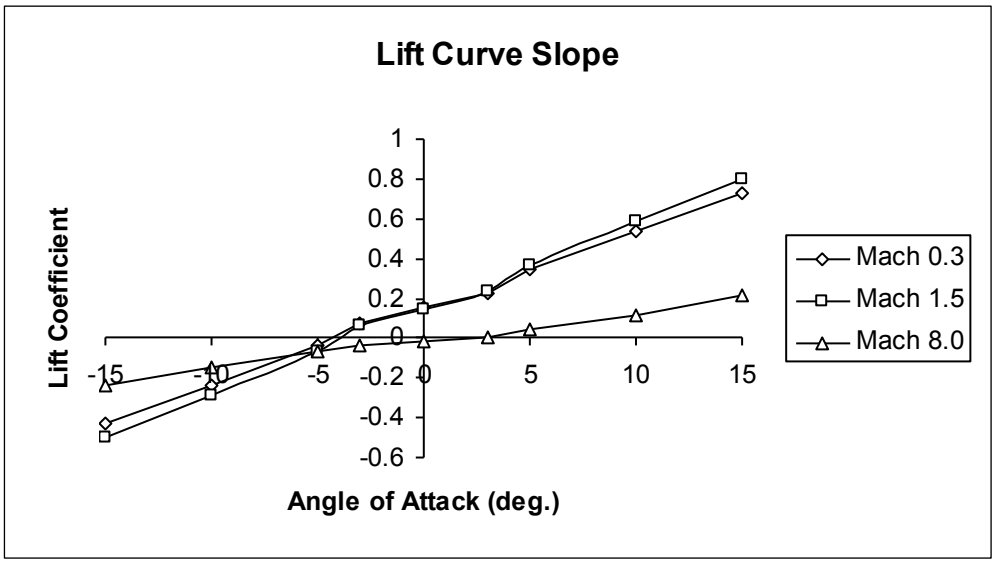

Figure 4 - Lift Curve Slope

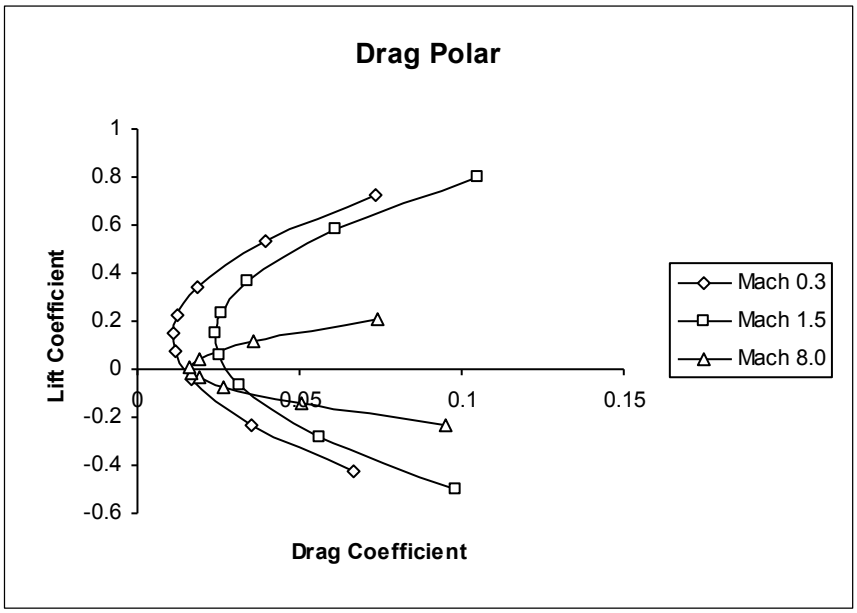

Figure 5 - Drag Polar

As indicated in the plot of lift curve slope (Figure 4), the relationship between lift and angle of attack is nearly linear for a given Mach number, hence the following equation applies:

$$C_l = C_{lo} + S * \alpha \qquad (1)$$

where C_{lo} is the lift coefficient at an angle of attack of zero degrees, S is the lift slope curve parameter and α is the angle of attack (either degrees or radians may be used in the regression; here angles in radians were used). This linear relation is what one would expect from basic aerodynamic theory [3].

For a given Mach number, the relationship between lift and drag is non-linear, and the shape of the curves in the graph above suggests the use of an equation of the form:

$$C_d = C_{do} + K * C_l^2 \qquad (2)$$

or, alternately:

$$C_d = C_{do} + K_1 * C_l + K_2 * C_l^2 \qquad (3)$$

where C_{do} is the drag coefficient corresponding to zero lift condition and K, K_1 and K_2 are parameters.

In these two equations, the constants C_{lo}, C_{do}, K, and S expected in a conventional single Mach number analysis are replaced with quadratic or cubic polynomial equations that vary these coefficients with Mach number, M. For example, the equation to calculate K as a function of Mach number may take the form,

$$K = K(M) = \beta_0 + \beta_1 * M + \beta_2 * M^2 \qquad (4)$$

where β_0, β_1, and β_2 are constants.

Similar polynomial equations can be determined for C_{lo}, C_{do} and S. In this way, the simple relationships of equations (1) and (2) can be extended to model the entire flight regime.

The second goal of this research was to extend the multivariable regression analysis technique described above to include *geometric* variables in the parametric

equations. For example, an equation was derived to calculate C_l and C_d over the entire flight regime as a function of the wing aspect ratio. APAS was used to generate tables of lift and drag coefficients for three different aspect ratios. Then, using the basic relationships in equations (1) and (2), a new version of equation (4) was determined having the form,

$$K(M) = \beta_0 + \beta_1 * M + \beta_2 * AR + \beta_3 * M * AR + \beta_4 * AR^2 + \beta_5 * M^2 \qquad (5)$$

where AR represents wing aspect ratio. Quadratic polynomial equations such as this are commonly used in regression analysis to capture the first and second order effect of each variable as well as the interaction effect between the two variables [1].

4.3. Regression Analysis

To develop the desired parametric equations, methods of regression analysis were used to model the datasets generated in APAS. Specifically, the Least Squares method was employed to determine the relations between Mach number, lift and drag. The APAS data was imported into a Microsoft Excel spreadsheet and the regression analysis was carried out using Excel's built-in regression tool.

Equations of the form of (1) and (2) (and equation (3) when including both first and second order terms produced more accurate results) were derived for the set of C_l and C_d coefficients corresponding to each Mach number at which the test vehicle was analyzed in APAS. In other words, a regression was done at each Mach number with α as the independent variable and C_l as the response variable, and a second regression was done at each Mach number with C_l as the independent variable and C_d as the response variable. From this, values of C_{lo}, C_{do}, S and K were obtained at each Mach number.

The values of C_{lo}, C_{do}, S and K were then used to perform a new regression for each of these parameters against Mach number, i.e. using Mach number as the

independent variable and C_{lo}, C_{do}, S and K as response variables. This regression permitted the determination of suitable equations describing the relationship between these four parameters and Mach number.

4.4. Parametric Equation Development

As discussed above, two separate codes, UDP and HABP, are used in conjunction with APAS to calculate the force coefficients for the subsonic and hypersonic regimes. Unfortunately, neither one is well suited for the transonic regime. UDP, a code based on a vortex panel method and slender body theory, is valid for linearized subsonic and low supersonic flow [5]. HABP is valid for hypersonic flow, essentially for speeds above Mach 4, since it is based on various impact methods such as Newton's sine squared law which are increasingly accurate as Mach number increases [5]. Thus, the region between Mach 2 and Mach 4 is difficult to model accurately with APAS. Typically, UDP is used up to about Mach 1.5, and HABP is used at Mach 2.0 and above to essentially "split the difference."

Also, in order to model the mathematical discontinuity that occurs at Mach 1 where (mathematically, at least) the drag goes to infinity, two separate parametric models were required since a single set of equations would be unable to efficiently model the discontinuity. A complete set of parametric equations was therefore derived for subsonic flow and a second complete set was derived for supersonic flow. The subsonic equations are primarily quadratic equations where the best fit of the data was obtained using only Mach number and it's square, but the supersonic equations are primarily cubic wherein adding the M^3 term provided the best fit of the data.

In addition, modeling the drag coefficient was most accurate for the subsonic case when equation (3) was used, however supersonically the drag coefficient was best modeled using equation (2), i.e. having only a single K parameter as opposed to two.

The parametric equations developed for the test vehicle are given in the following results section.

5. RESULTS

5.1. Non-varying Geometry

The equations developed for *Polaris* in the subsonic flow regime using the regression analysis techniques are as follows:

Subsonic Flow

$$C_{lo} = 0.153 - 0.006*M + 0.01*M^2 \quad (6)$$

$$C_l = C_{lo} + S * \alpha \quad (7)$$

$$S = 2.293 - 0.666*M + 1.071*M^2 \quad (8)$$

$$C_{do} = 0.014 - 0.0002*M - 0.0016*M^2 \quad (9)$$

$$K_1 = -0.047 - 0.003*M + 0.005*M^2 \quad (10)$$

$$K_2 = 0.177 + 0.005*M - 0.011*M^2 \quad (11)$$

$$C_d = C_{do} + K_1*C_l + K_2*C_l^2 \quad (12)$$

The accuracy of each of these equations with respect to actual APAS data is shown graphically below in Figures 6 through 10. In each plot, the actual data points generated in APAS are shown in bold lines and the fitted curves are shown in dashed lines.

The coefficient of determination, R^2, is also shown for each curve fit. The coefficient of determination is an indicator of the measure of variability in the response variable that is accounted for by the predictor variable(s), and thus provides a measure of the validity of the regression model used. A coefficient of determination equal to one indicates a perfect fit of the equation to the data (all data points fall on the regression line) and a value equal to zero indicates no relation whatsoever between the response variable and predictor variable(s).

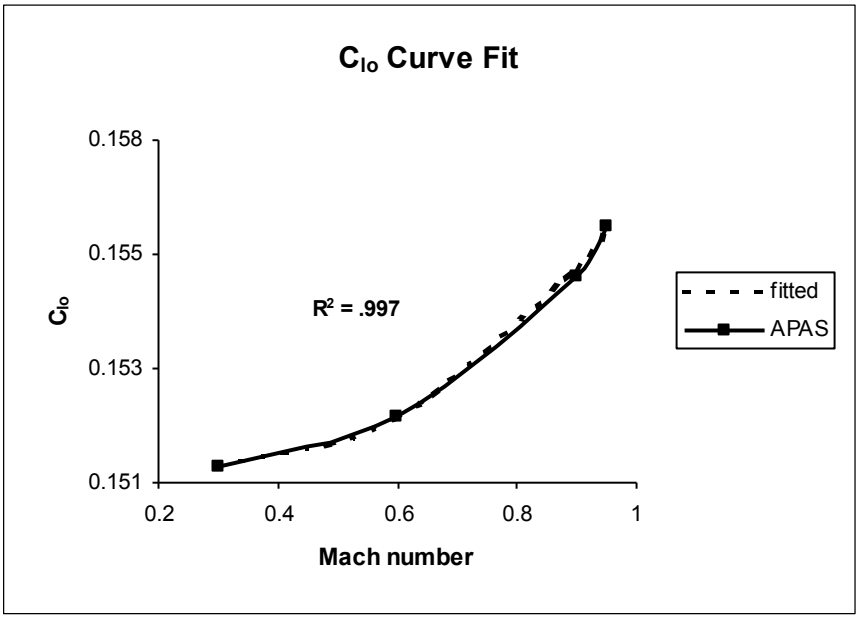

Figure 6 – C_{lo} Curve Fit

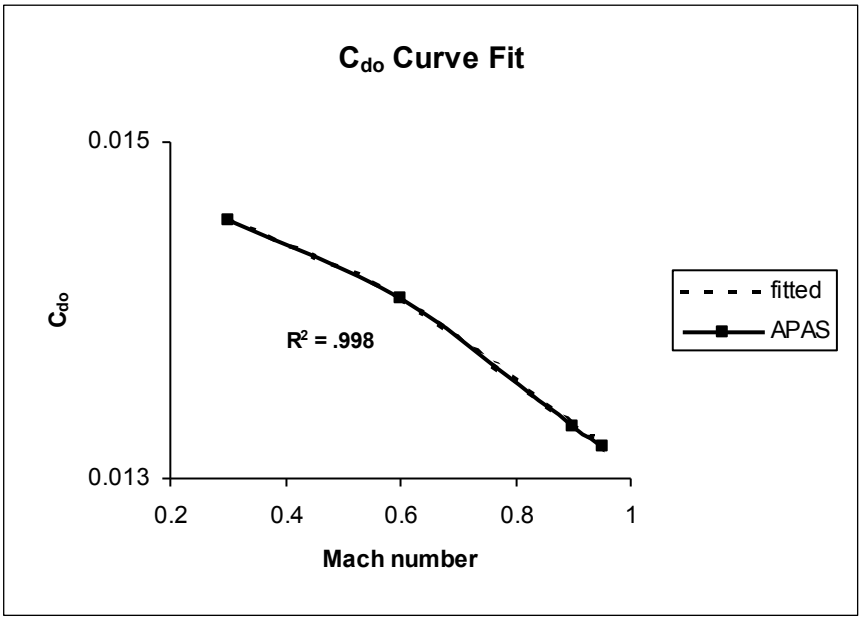

Figure 7 - C_{do} Curve Fit

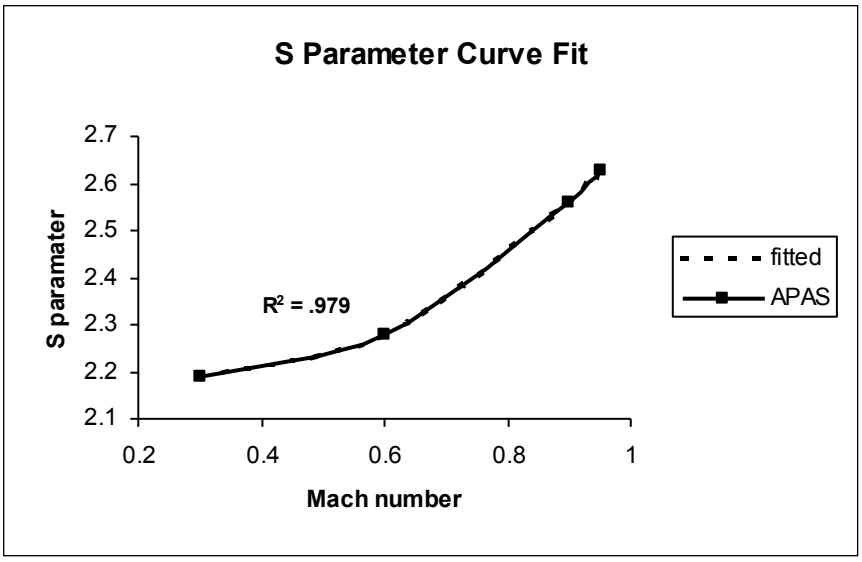

Figure 8 – S Parameter Curve Fit

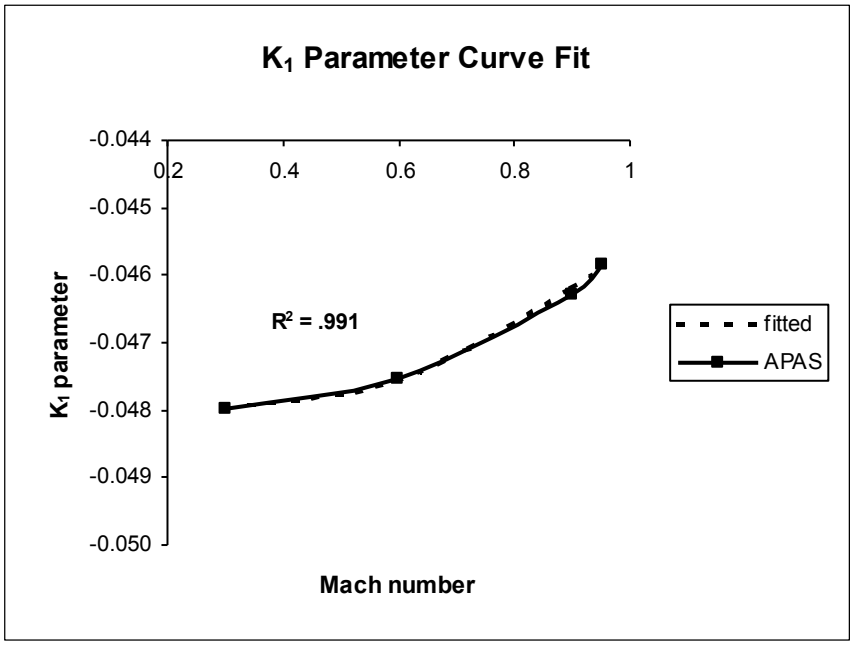

Figure 9 - K_1 Parameter Curve Fit

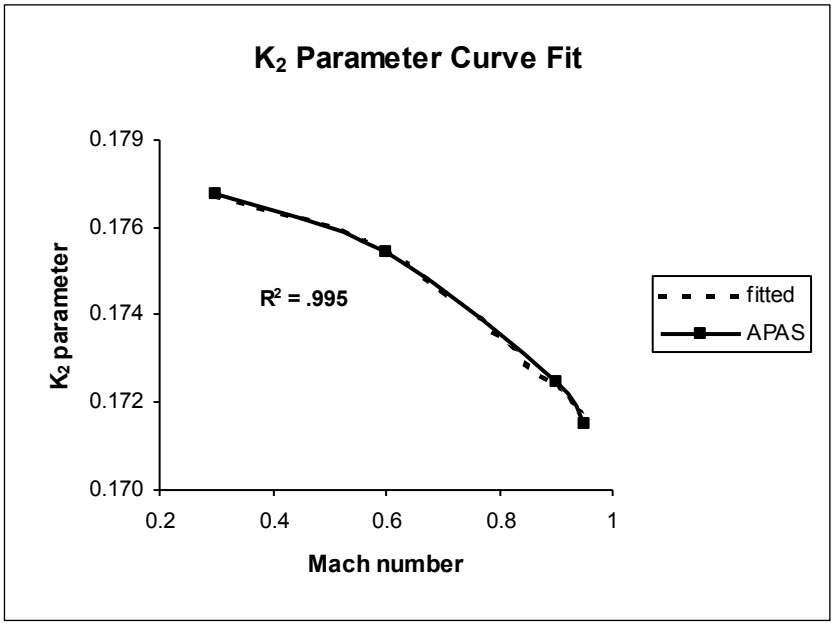

Figure 10 – K_2 Parameter Curve Fit

As the above plots show, excellent curve fits were obtained for all of the parameters as a function of Mach number, with coefficients of determination of ~0.98 or greater. This indicates that the regression model used accurately represents the APAS tabular data and that the equations may then be used to accurately calculate the aerodynamic coefficients for the test vehicle.

The equations developed for the test vehicle in the supersonic flow regime are the following:

Supersonic Flow

$$C_{lo} = 0.214 - 0.082*M + 0.008* M^2 - 0.0002*M^3 \quad (13)$$

$$C_l = C_{lo} + S * \alpha \quad (14)$$

$$S = 3.729 - 0.862*M + 0.0761* M^2 - 0.0021*M^3 \quad (15)$$

$$C_{do} = 0.0385 - 0.003*M + 0.0002*M^2 + 0.000007* M^3 \quad (16)$$

$$K = -0.291 + 0.396*M - 0.029*M^2 + 0.0007*M^3 \quad (17)$$

$$C_d = C_{do} + K * C_l^2 \quad (18)$$

In this case, cubic equations produced the best curve fits in contrast to the quadratic equations used for the subsonic regime. The curve fit for each of these equations is also shown graphically below in Figures 11 through 14.

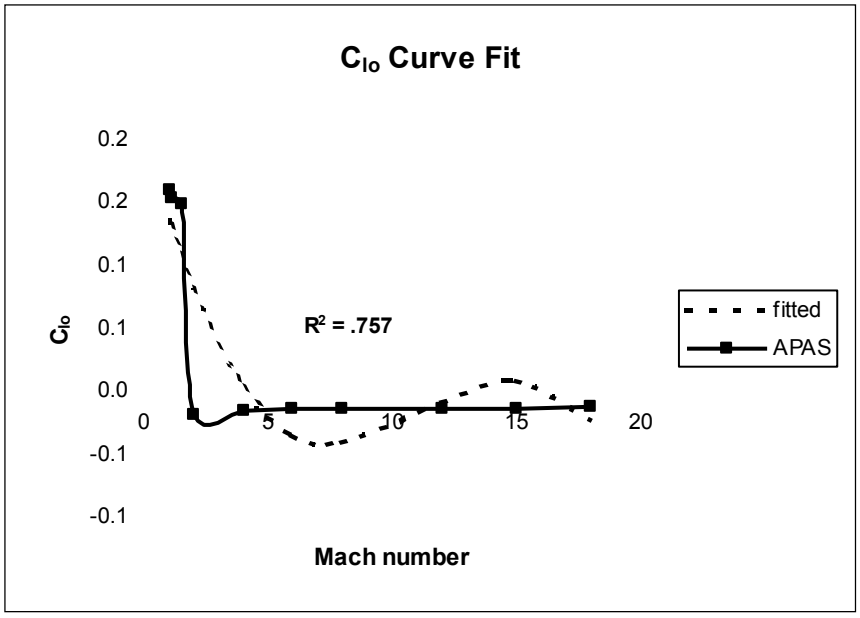

Figure 11 - C_{lo} Curve Fit

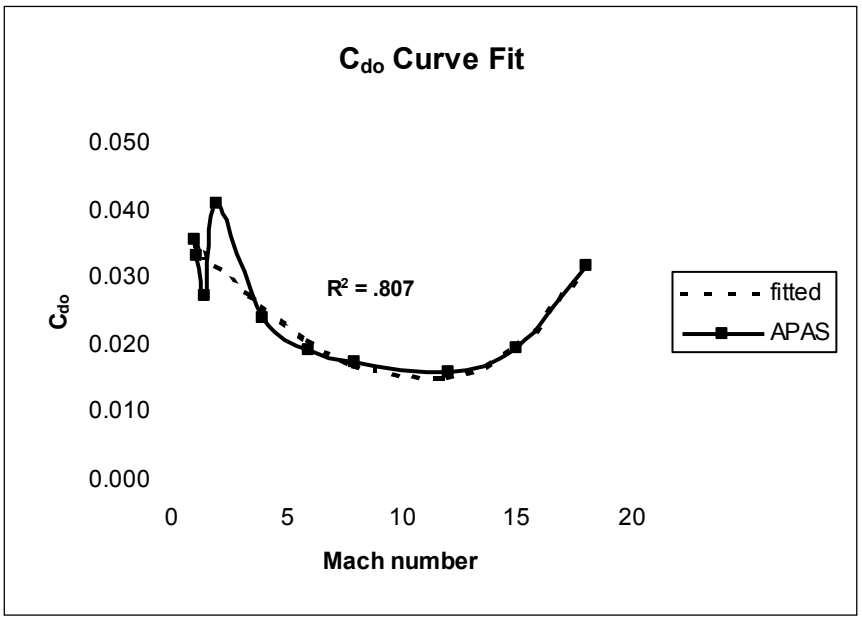

Figure 12 – Cdo Curve Fit

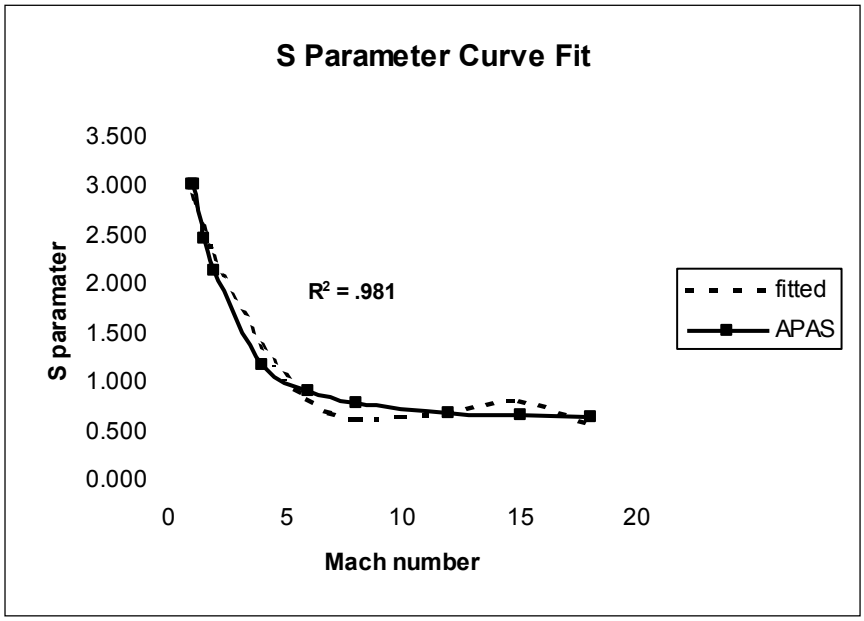

Figure 13 – S Parameter Curve Fit

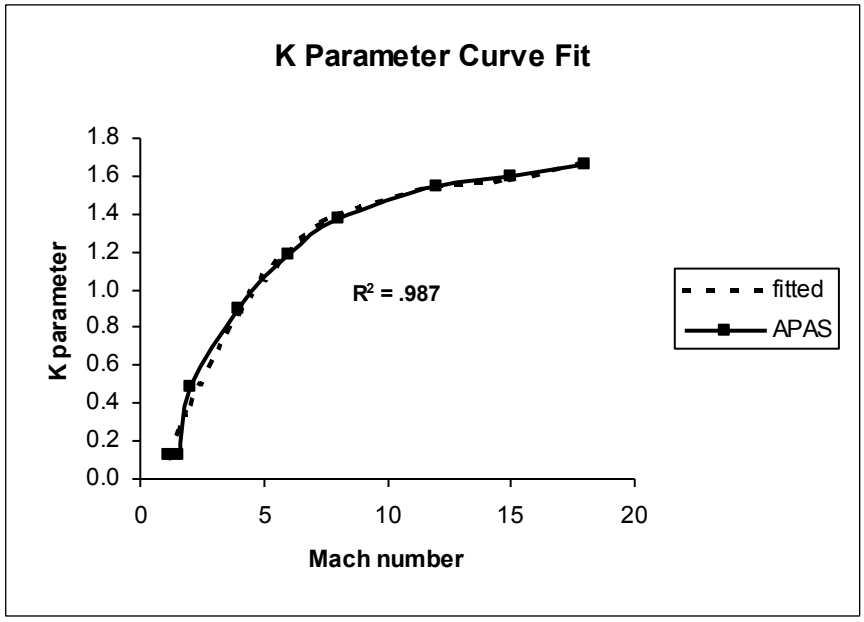

Figure 14 – K Parameter Curve Fit

As the above plots show, excellent curve fits were obtained for the S and K parameters in the supersonic flow model. However, it was somewhat difficult to fit a curve to the lift and drag coefficient data due to the different manner in which these values are calculated in UDP and HABP. The break in the data at Mach 2 where the

analysis tool used changes from UDP to HABP is obvious.

For example, the jump in C_{do} at Mach 2 that is seen in Figure 12 is not an actual reflection of conditions encountered in flight, but simply a result of the different analysis method employed by the tool. A large increase in drag occurs near Mach 1, but not at Mach 2. In this case in particular, the smooth fitted curve may actually give a more accurate reflection of flight conditions, and hence the relatively low R^2 value (0.735) is perhaps somewhat misleading.

The error between the fitted curves and explicit APAS data was determined also in order to evaluate the accuracy of the parametric equations. The error in lift and drag coefficients at both subsonic and supersonic flow conditions is graphically illustrated in Figures 15 through 18.

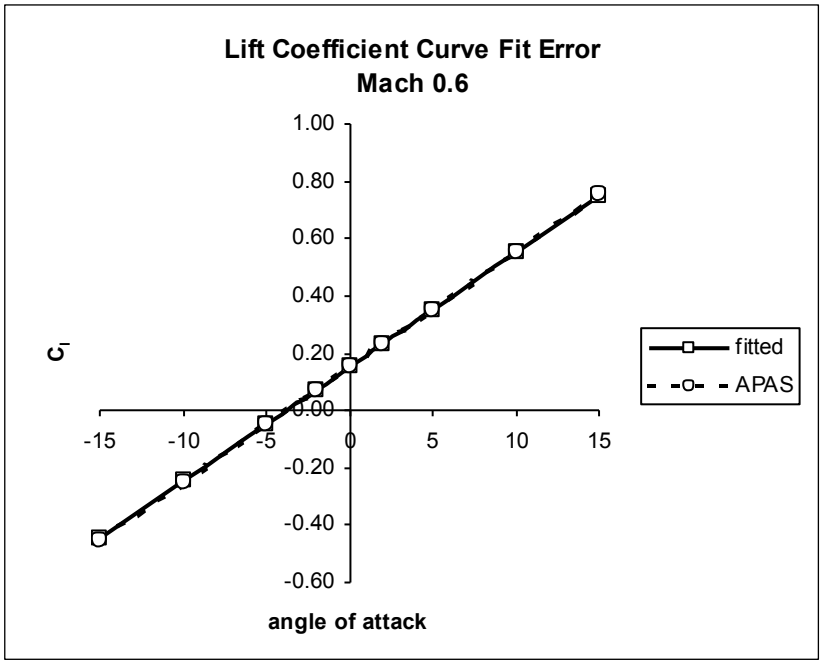

Figure 15 – Subsonic Lift Coefficient Error

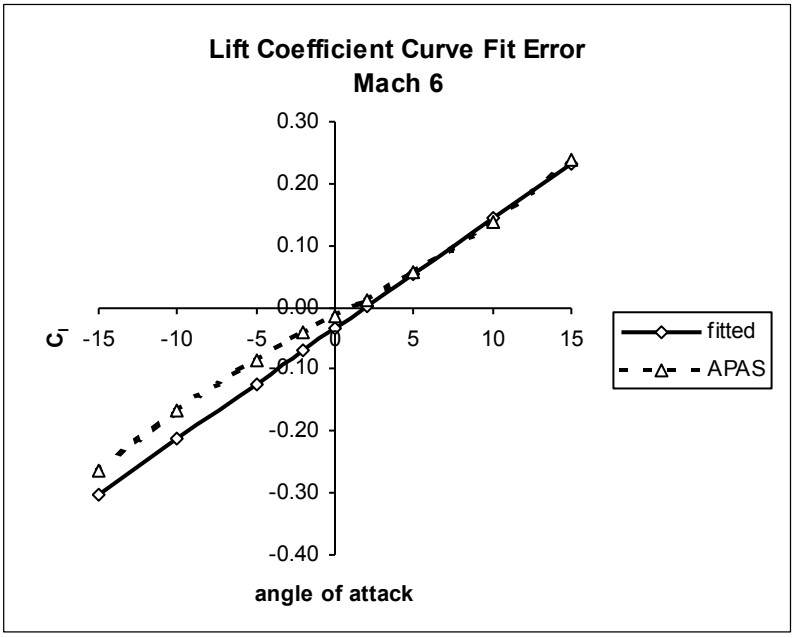

Figure 16 – Supersonic Lift Coefficient Error

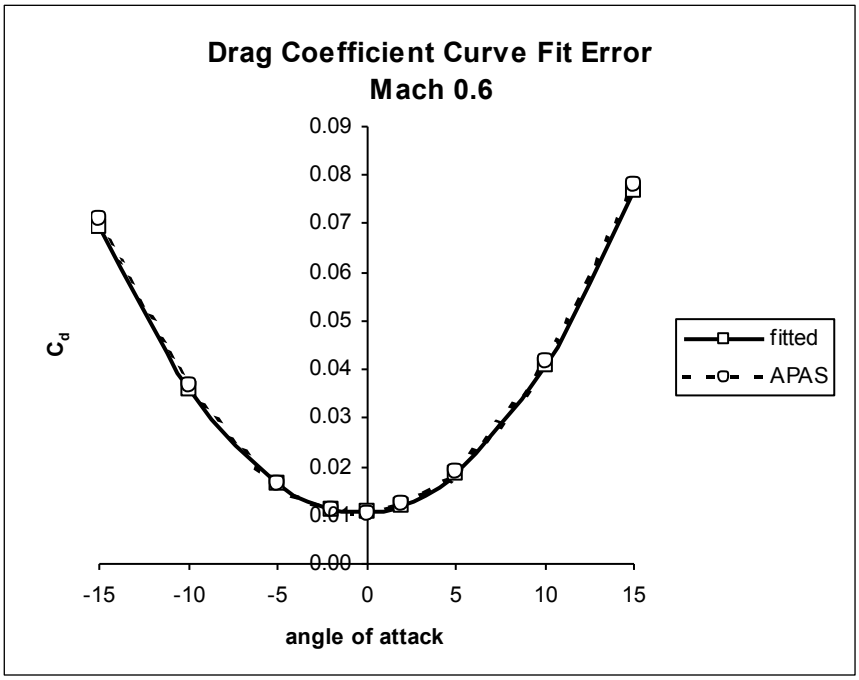

Figure 17 – Subsonic Drag Coefficient Error

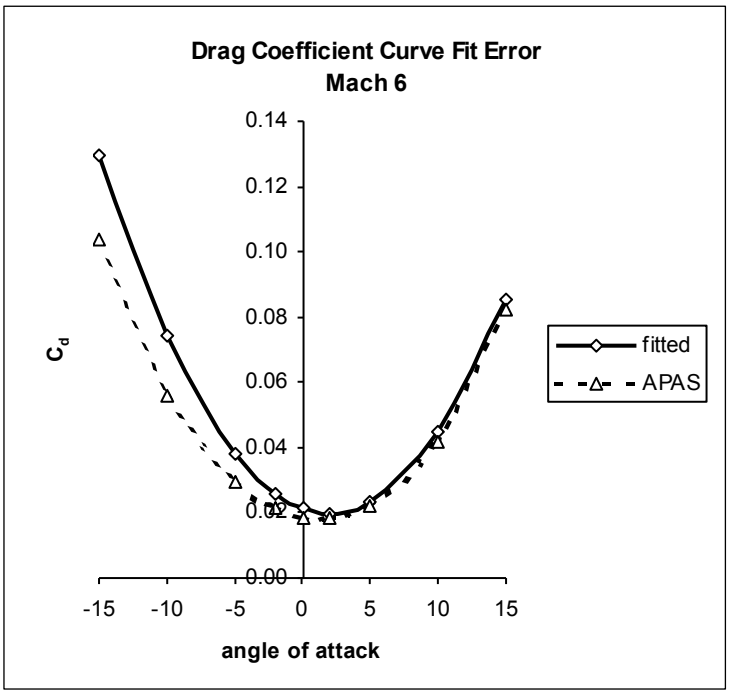

Figure 18 – Supersonic Drag Coefficient Error

As may be seen in the above plots, the curve fits for subsonic flow velocities are extremely accurate. The error between the fitted data and APAS data is on the order of 1% at a given data point. The results for supersonic flow conditions are also quite accurate for lift coefficient. The supersonic drag coefficient curve fit is less accurate, with most of the error occurring primarily at negative angles of attack.

5.2. Wing Aspect Ratio

The techniques discussed above were applied to changing the aspect ratio of the test vehicle's wing. Three additional models were created in APAS, all identical except for aspect ratio. Aspect ratios of 1.5, 2.0 and 2.5 were used, and parametric equations were then developed to calculate C_{lo}, C_{do}, C_l, C_d, S, K_1 and K_2 as a function of wing aspect ratio and Mach number. These equations are all of the form of equation (5) wherein the first and second order term of each parameter (Mach number and aspect ratio), as well as the interaction term of the two variables, are used to model the data. The subsonic and supersonic equations are given below.

Subsonic Flow

$$C_{lo} = 0.11 - 0.013*M + 0.028*AR - 0.0003*M*AR + 0.0063*AR^2 + 0.0234*M^2 \quad (19)$$
$$C_{do} = 0.104 - 0.005*M + 0.002*AR - 0.0001*M*AR - 0.0004*AR^2 + 0.0027*M^2 \quad (20)$$
$$S = -0.664 - 1.606*M + 2.413*AR + 0.429*M*AR - 0.46*AR^2 + 1.046*M^2 \quad (21)$$
$$K_1 = -0.049 + 0.0023*M + 0.029*AR - 0.002*M*AR - 0.005*AR^2 + 0.004*M^2 \quad (22)$$
$$K_2 = 0.62 - 0.018*M - 0.34*AR + 0.014*M*AR + 0.058*AR^2 - 0.018*M^2 \quad (23)$$

Supersonic Flow

$$C_{lo} = 0.099 - 0.022*M - 0.013*AR - 0.0002*M*AR + 0.004*AR^2 + 0.001*M^2 \quad (24)$$
$$C_{do} = 0.047 - 0.071*M + 0.016*AR + 0.0002*M*AR - 0.0006*AR^2 + 0.0004*M^2 \quad (25)$$
$$S = 0.747 - 0.339*M + 1.752*AR - 0.051*M*AR - 0.263*AR^2 + 0.018*M^2 \quad (26)$$
$$K_1 = -0.026 - 0.0005*M + 0.012*AR - 0.002*M*AR - 0.002*AR^2 + 0.00001*M^2 \quad (27)$$
$$K_2 = 0.268 + 0.262*M - 0.291*AR - 0.002*M*AR + 0.057*AR^2 - 0.01*M^2 \quad (28)$$

The accuracy of the above in the subsonic flow regime is again very good. However, the error of the supersonic equations is considerable for the lift coefficients. This is shown in Figures 19 and 20, where the force coefficients were calculated using an aspect ratio of 2.0.

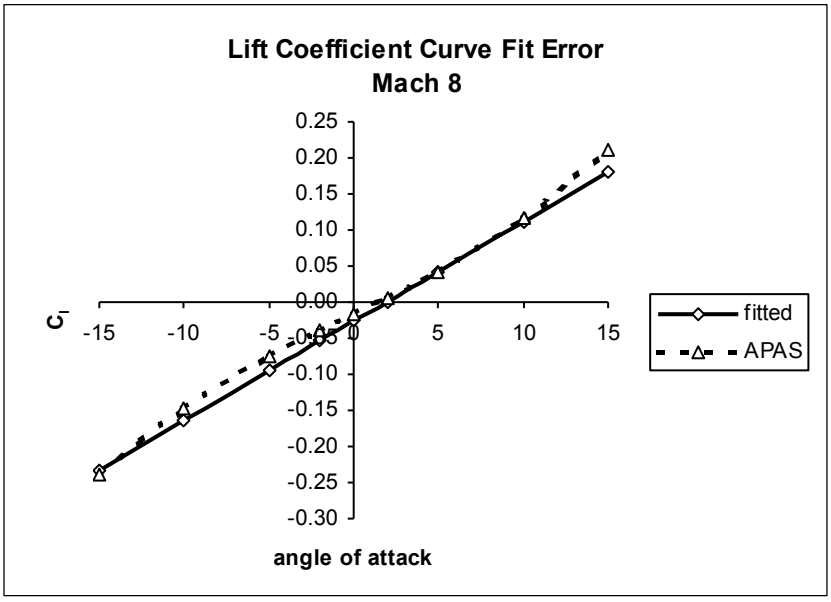

Figure 19 – Supersonic Lift Coefficient Error

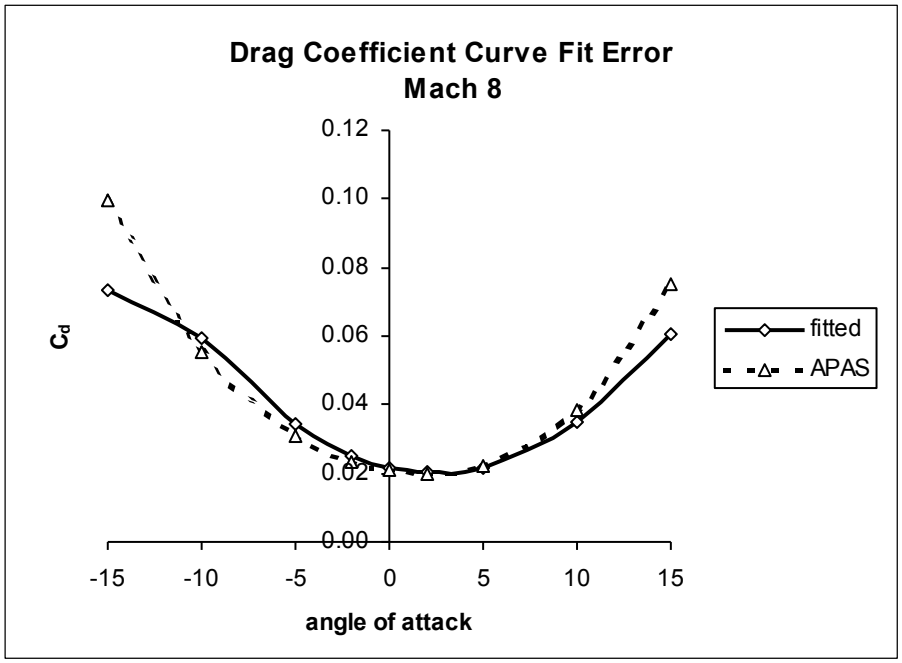

Figure 20 – Supersonic Drag Coefficient Error

The error between fitted data points and APAS data points for lift coefficients was as high as 78% with an average error of 27%. In contrast, the accuracy of the equations are better for supersonic drag coefficients, with the error between a given fitted data point and the corresponding APAS data point being about 10% on average, although at the highest and lowest angles of attach the error is as high as 26%. This is an interesting result given that the drag coefficients are calculated from the lift coefficients. Figure 21 shows the percent error at each data point and the average error for the Mach 8 case.

Alpha		Cl	Cl	% error	Cd	Cd	% error
deg.	radians	fitted	APAS		fitted	APAS	
-15	-0.261799	-0.2330	-0.2392	2.58%	0.073198	0.0994	26.36%
-10	-0.174533	-0.1641	-0.1468	11.82%	0.059471	0.0553	7.54%
-5	-0.087266	-0.0953	-0.0746	27.69%	0.034137	0.0311	9.76%
-2	-0.034907	-0.0539	-0.0386	39.69%	0.025076	0.0235	6.71%
0	0	-0.0264	-0.0166	58.81%	0.021594	0.0209	3.32%
2	0.034907	0.0012	0.0056	78.67%	0.020158	0.02	0.79%
5	0.087266	0.0425	0.0424	0.31%	0.021842	0.0224	2.49%
10	0.174533	0.1114	0.117	4.77%	0.034881	0.0385	9.40%
15	0.261799	0.1803	0.2122	15.03%	0.060712	0.0752	19.27%
			Avg.	26.59%		Avg.	9.52%

Figure 21 – Data Error

6. REGRESSION ANALYSIS TOOL

6.1. Fortran Code

The results presented thus far were obtained by manually carrying out the regression analysis using Microsoft Excel, which has a built-in regression function. For the test vehicle described above, this involved doing the regression analysis a total of seventy-two times: one for each Mach number to determine C_{lo} and S; twice for each Mach number to determine whether it was better to calculate C_d using the square of Mach number only or both the square and the cube of Mach number; and three times for each parameter C_{lo}, C_{do}, S, K_1 and K_2 to determine if they were best modeled as a function of Mach number only, or whether to include Mach number squared and Mach number cubed for both the subsonic and supersonic cases. Additional effort was put into investigating other forms of the regression model for various parameters, such as exponential or logarithmic models, neither of which proved to significantly improve the accuracy of the parametric equations.

Clearly, this is a very time consuming process, one that cannot be carried out at each design iteration. Hence, a Fortran program was written to enable the regression analysis to be carried out automatically. This program, entitled *regression*, is very simple to use and requires minimal effort by the user.

Regression requires as an input the APAS-generated tables of aerodynamic force coefficients of lift and drag in the form of a POST input deck, and generates as output a text file containing the values of the five parameters (C_{lo}, C_{do}, S, K_1 and K_2) at each Mach number and a listing of parametric equations for both subsonic and supersonic flow. The program also calculates a value of the coefficient of determination, R^2, for each parametric equation as a gauge of the accuracy of the same.

The Fortran program uses only polynomial equations, either cubic or quadratic, to fit the aerodynamic data generated in APAS. It does not allow the user to select other

types of regression models (e.g. exponential). This is a result of the method used to solve for the parameter coefficients, namely matrix operations which are discussed in further detail below.

Regression uses several matrix operations to solve for the parameter coefficients, including matrix multiplication, transposition, and inversion. These operations are used to solve the matrix equation:

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \qquad (29)$$

where **b** is a vector containing the coefficients in a particular parametric equation, **X** is the matrix of predictor variables (Mach number, Mach number squared, angle of attack, etc.), $\mathbf{X}^T$ is the transpose of matrix **X**, and **Y** is the matrix of independent or response variables (C_d, C_l, etc.). This is the standard Least Squares linear regression solution using matrix operations [2].

The program solves this equation in six steps. First, data is read from the input file to assemble the matrix **X**. Second, the matrix is transposed. Third, the matrix multiplication operation of $\mathbf{X}^T\mathbf{X}$ is carried out. Fourth, the square product matrix $\mathbf{X}^T\mathbf{X}$ is inverted. Fifth, the inverted matrix is multiplied by $\mathbf{X}^T$ and sixth, this product matrix is multiplied by **Y**, giving the desired vector of coefficients. These steps are carried out as many times as may be required to complete the entire regression analysis.

Regression was written in Fortran 90, a superset of Fortran 77, since Fortran 90 has intrinsic matrix manipulation functions such as multiplication and transpose, features which made writing the code simpler and which also reduce CPU time at execution. Fortran 90 also allows dynamic allocation of arrays which means array sizes are dependent on user inputs and need not be explicitly defined by the programmer. This significantly reduces memory usage by obviating the need to allocate extra-large arrays to allow program flexibility.

The program has been compiled to run as a DOS program on PC computers, the executable file is *regression.exe*.

6.2. Program Execution

The program requires several steps in its implementation. The user must first carry out the vehicle analysis in APAS. Next, the user must run the Fortran program *apasdat* to generate a POST input deck text file. This input deck must be slightly edited to delete the POST table data found at the beginning of the lift coefficient table as well as the drag coefficient table (see Figure 22). If desired, the table of pitching moment coefficients may be deleted since it is not used by *regression*. The input file must be located in the same directory as the executable file. Once the POST input file created in *apasdat* is modified, the code may then be executed by typing "regression" at the DOS prompt.

```
l$tab table=5hclt  ,2,5halpha,5hmach , 12, 15,8*1,
         0.30,
        -15.0,   -0.3743,
        -10.0,   -0.2418,
         -5.0,   -0.1092,
         -2.0,   -0.0297,

$end
l$tab table=5hcdt  ,2,5halpha,5hmach ,  9, 14,8*1,
         0.30,
        -15.0,    0.0531,
        -10.0,    0.0278,
         -5.0,    0.0139,
```

Figure 22 – POST input deck (partial)

When it is executed, *regression* prompts the user to input the number of angles of attack at which the APAS analysis was performed, the number of subsonic Mach numbers used in the analysis, and the number of supersonic Mach numbers. Since the program was written in Fortran 90, the size of each array is dynamically defined (a feature not available in Fortran 77) based on the user inputs and thus there are no limits to how many angles of attack may be used or how many Mach number points may be used.

The input file containing the lift and drag coefficients must be named "aerodeck.txt" and must be located in the same directory as the executable file.

A listing of the code is included in Appendix A. The code includes several linear algebra subroutines and functions obtained from a Fortran-related web site maintained by the University of Tennessee at Knoxville [8].

6.3. Program Output

The output of *regression* consists of a listing of the parameters C_{lo}, C_{do}, S, K_1 and K_2 calculated for each Mach number, a set of equations for the subsonic flow regime, and a set of equations for the supersonic flow regime. The output is written to a file called "equations.txt" located in the same directory as the executable and input files.

A sample output from the Fortran program for the test vehicle is shown in Figure 23. This output is based on the same APAS data as was used in the original Excel analysis.

Note that the Fortran program uses equation (3) rather than equation (2), the latter being the equation used in the manual regression analysis. The accuracy in the drag coefficient calculation improves slightly when both the K_1 and K_2 parameters are included.

Mach #	Clo	Cdo	S	K1	K2
0.30	0.15150	0.0142	2.20589	-0.04797	0.17719
0.60	0.15243	0.0137	2.30464	-0.04755	0.17581
0.90	0.15501	0.0129	2.55838	-0.04629	0.17268
0.95	0.15590	0.0128	2.64517	-0.04580	0.17166
1.05	0.16080	0.0357	3.05157	-0.04685	0.16663
1.10	0.15421	0.0334	3.06074	-0.04555	0.16813
1.50	0.14844	0.0273	2.49097	-0.04856	0.18204
2.00	-0.01860	0.0409	2.16678	-0.00446	0.48765
4.00	-0.01474	0.0240	1.18988	-0.00939	0.89671
6.00	-0.01390	0.0193	0.92194	-0.01278	1.17532
8.00	-0.01354	0.0175	0.80560	-0.01550	1.35621
12.00	-0.01330	0.0160	0.71116	-0.01868	1.52079
15.00	-0.01317	0.0195	0.67954	-0.01756	1.57726
18.00	-0.01297	0.0320	0.65445	-0.01335	1.64789

Subsonic Equations	R^2
Clo = 0.152501 + -0.006420 * M + 0.010402 * M^2	0.997
S = 2.290521 + -0.574222 * M + 0.985477 * M^2	0.997
Cdo = 0.014437 + -0.000257 * M + -0.001556 * M^2	0.999
K1 = -0.047381 + -0.003582 * M + 0.005441 * M^2	0.996
K2 = 0.176496 + 0.005610 * M + -0.011134 * M^2	0.999

Supersonic Equations	
Clo = 0.213707 + -0.081599*M + 0.008163*M^2 + -0.000242*M^3	0.756
S = 3.760170 + -0.869027*M + 0.076720*M^2 + -0.002124*M^3	0.981
Cdo = 0.038470 + -0.003404*M + 0.000027*M^2 + 0.000008*M^3	0.806
K1 = -0.059226 + 0.018492*M + -0.002042*M^2 + 0.000065*M^3	0.619
K2 = -0.213437 + 0.367748*M + -0.026420*M^2 + 0.000653*M^3	0.995

Figure 23 – *Regression* Output

7. SUMMARY

This research has demonstrated that aerodynamic datasets generated in APAS for a given aerospace vehicle may be successfully transformed into a set of parametric equations through methods of linear regression. While the resulting accuracy of the parametric equations is very good for dataset transformations involving force coefficients as a function of Mach number only, the accuracy of the equations generated by fitting force coefficients as a function of Mach number and a specific geometric parameter (wing aspect ratio in this case) is not as good. It appears from this research that regression analysis is perhaps not applicable to the latter case, at least not using a regression model of the form of equation (5). Further research would be required to determine a more suitable model, perhaps using additional predictor variables (e.g. higher order terms), in order to obtain parametric equations whose accuracy is within an acceptable error range.

Because of the lack of success in modeling a vehicle's aerodynamics when geometric parameters are included, the second goal of this research effort has not been met. If a suitable regression model cannot be found using the methods discussed above, it may be necessary to test an entirely different approach to the task of integrating APAS more fully into an MDO environment. Possible avenues for research include further developing the *ideas2apas* translator[1] to include vehicle components such as wings and engine nacelles, or automating the analysis process by writing a computer code to automatically run UDP and HABP. Efforts have been undertaken at NASA's Langley, Virginia research facility in this latter area.

The Fortran code *regression*, written to automatically carry out the regression analysis and derive parametric equations from a given input file, provides a quick and simple means of transforming APAS datasets into parametric equations, but does not address the need to analyze multiple configurations of a given vehicle.

[1] This code was developed by Peter Bellini as part of his AE 8500 project (Fall 1997) at Georgia Tech.

The program could be modified and extended to include the parametric modeling of a vehicle with changing geometric parameters if a suitable regression model were found. This would entail modifying the program to allow it to read several input files, one from each APAS model analysis, and writing new DO loops to include the additional geometric parameters in the regression analysis.

8. REFERENCES

1. Schnackel, J; Dovenmuehle, R., "Statistical Experimental Design and its Role in Aerospace Vehicle Design Efforts," AIAA 90-1809, AIAA Aerospace Engineering Conference & Show, February 1990.

2. Neter, John; Kutner, Michael; Nachtsheim, Christopher, and Wasserman, William; Applied Linear Statistical Models, 4th Edition, Irwin, Chicago, IL, 1996.

3. Anderson, John D. Fundamentals of Aerodynamics, 2nd Edition. McGraw Hill, New York, NY, 1984.

4. Vanderplaats, Garret N., Numerical Optimization Techniques for Engineering Design, McGraw Hill, New York, NY, 1984.

5. Bonner, E., Clever, W., and Dunn, K., *Aerodynamic Preliminary Analysis System II, Part I - Theory*, NASA Contractor Report #182076, April 1991.

6. Sova, G. and Divan, P., *Aerodynamic Preliminary Analysis System II, Part II - User's Manual*, NASA Contractor Report #182077, April 1991.

7. Brauer, G.L. et. al., *Program to Optimize Simulated Trajectories – Volume II*, NAS1-18147, September 1989.

8. World Wide Web, Netlib Repository, URL http://www.netlib.org.

9. APPENDIX A – Fortran Program Code

```
      program regression
c this program will perform the least squares method of regression analysis
c on a data set to derive a set of parametric equations for calculating the
c aerodynamic force coefficents of lift and drag as a function of mach number
      implicit none
      integer angle, machsub, machsup, mach
      real, dimension (:), allocatable :: alpha,machnum,s,cdo,clo,cd
      real, dimension (:), allocatable :: k1,k2,y,yb,ya
      real, dimension (:,:), allocatable :: cl,x2,x3,x3b,x3c,x4,xt2
      real, dimension (:,:), allocatable :: xt3,xt3b,xt3c,xt4,xn2,xn3a
      real, dimension (:,:), allocatable :: xn3b,xn3c,xn4
      real rconv, rsq
      real xx2(2,2), xx3(3,3), xx4(4,4)
      real xi2(2,2),xi3(3,3),xi4(4,4)
      real b2(2), b3(3), b4(4)
      real work(4), det(2)
      integer info, job, i, j, d, z
      integer ipvt(4)
      job=11
      rconv=0.01745329252
      open (unit=20, file="aerodeck.txt", status="old")
      open (unit=21, file="equations.txt",status="unknown")
      print*, "Enter the number of angles of attack used in apas"
      read*, angle
      print*, "Enter the number of subsonic mach numbers used in apas"
      read*, machsub
      print*, "Enter the number of supersonic mach numbers used in apas"
      read*, machsup
      mach=machsub + machsup
c define array sizes
      allocate (alpha(angle), machnum(mach),cl(angle,mach),s(mach))
      allocate (clo(mach),cd(mach),k1(mach), k2(mach), y(machsup))
      allocate (yb(machsub),x2(angle,2), x3(angle,3), x3b(machsub,3))
      allocate (x3c(machsup,3),x4(machsup,4),ya(angle),xt2(2,angle))
      allocate (xt3(3,angle),xt3b(3,machsub),xt3c(3,machsup))
      allocate (xt4(4,machsup),xn2(2,angle), xn3a(3,angle),cdo(mach))
      allocate (xn3b(3,machsub),xn3c(3,machsup), xn4(4,machsup))
c  calculate clo and s for all mach numbers, assign to arrays clo and s
c  equation is cl = clo + s * alpha
      print*,"reading in info"
      do j = 1, mach
         read(unit=20, fmt=*) machnum(j)
         do i=1,angle
           read(unit=20,fmt=*) alpha(i), cl(i,j)
c convert degrees to radians
              alpha(i)=alpha(i)*rconv
              x2(i,1) = 1.0
              x2(i,2) = alpha(i)
              ya(i) = cl(i,j)
         end do
         xt2=transpose(x2)
         xx2 = matmul(xt2,x2)
c invert the matrix
         call invert_two_by_two(xx2, xi2)
         xn2=matmul(xi2,xt2)
         b2=matmul(xn2,ya)
         clo(j)=b2(1)
         s(j)=b2(2)
      end do
```

```
c calculate cdo, k1, k2 for all mach numbers, assign to arrays cdo, k1, k2
c equation is cd = cdo + k1 * cl + k2 * cl^2
       do j = 1, mach
          read(unit=20, fmt=*) machnum(j)
          do i=1,angle
                read(unit=20,fmt=*) alpha(i), cd(i)
                x3(i,1) = 1.0
                x3(i,2) = cl(i,j)
                x3(i,3) = cl(i,j)**2
                ya(i) = cd(i)
          end do
          xt3=transpose(x3)
          xx3=matmul(xt3,x3)
c  matrix inversion step
          call invert_three_by_three(xx3,xi3)
          xn3a=matmul(xi3,xt3)
          b3=matmul(xn3a,ya)
          cdo(j)=b3(1)
          k1(j)=b3(2)
          k2(j)=b3(3)
       end do
       print*,"initial regression for each mach number is complete..."
       print*,"beginning overall regression analysis..."
c  write to file all each coefficient at each mach number
       write(unit=21,fmt=70)
       do j=1,mach
          write(unit=21,fmt=71),machnum(j),clo(j),cdo(j),s(j),k1(j),k2(j)
       end do
c
c  initial subsonic and supersonic regressions are complete, i.e. all
c  parameters (clo, s, cdo, k1, k2) have been obtained.  next step
c  is to do the regression of each parameter against mach number.
c
c  note that all subsonic parameters are calculated as a function of
c  mach # and mach # squared, while all supersonic parameters use
c  an additional mach # cubed coefficient (except cdo, which is the
c  same as subsonic)
c
c  matrix equation is (xt*x)^-1 * xt * y
c do clo first:  equation is clo = bo + b1*m + b2 * m^2
       print*,"subsonic clo..."
       do j=1,machsub
          x3b(j,1)=1.
          x3b(j,2)=machnum(j)
          x3b(j,3)=machnum(j)**2
          yb(j)=clo(j)
       end do
       xt3b=transpose(x3b)
       xx3 = matmul(xt3b,x3b)
       call invert_three_by_three(xx3,xi3)
       xn3b=matmul(xi3,xt3b)
       b3 = matmul(xn3b,yb)
       write (unit=21,fmt=91)
       write (unit=21,fmt=90)
       call rsquare(yb,machsub,x3b,3,b3,rsq)
       write (unit=21,fmt=51) b3(1),b3(2),b3(3),rsq
c  subsonic regression of s on m, m^2
       print*,"subsonic s..."
       do j=1,machsub
          yb(j)=s(j)
       end do
       b3=matmul(xn3b,yb)
       call rsquare(yb,machsub,x3b,3,b3,rsq)
```

```
      write (unit=21,fmt=52) b3(1),b3(2),b3(3),rsq
c  subsonic regression of cdo on m, m^2
      print*,"subsonic cdo..."
      do j=1,machsub
         yb(j)=cdo(j)
      end do
      b3=matmul(xn3b,yb)
      call rsquare(yb,machsub,x3b,3,b3,rsq)
      write (unit=21,fmt=53) b3(1),b3(2),b3(3),rsq
c  subsonic regression of k1 on m, m^2
      print*,"subsonic k1..."
      do j=1,machsub
         yb(j)=k1(j)
      end do
      b3=matmul(xn3b,yb)
      call rsquare(yb,machsub,x3b,3,b3,rsq)
      write (unit=21,fmt=54) b3(1),b3(2),b3(3),rsq
c  subsonic regression of k2 on m, m^2
      print*,"subsonic k2..."
      do j=1,machsub
         yb(j)=k2(j)
      end do
      b3=matmul(xn3b,yb)
      call rsquare(yb,machsub,x3b,3,b3,rsq)
      write (unit=21,fmt=55) b3(1),b3(2),b3(3),rsq
      write (unit=21,fmt=91)
      write (unit=21,fmt=*) "supersonic equations"
c  supersonic regression of clo on m, m^2, m^3
      print*,"supersonic clo..."
      do j=1,machsup
         x4(j,1) = 1.0
         x4(j,2) = machnum(j+machsub)
         x4(j,3) = machnum(j+machsub)**2
         x4(j,4) = machnum(j+machsub)**3
         y(j)=clo(j+machsub)
      end do
      xt4=transpose(x4)
      xx4=matmul(xt4,x4)
c  gaussian elimination step
      call sgefa(xx4, 4, 4, ipvt, info)
c  matrix inversion step
      call sgedi(xx4, 4, 4, ipvt, det, work, job)
c solve for coefficients
      xn4=matmul(xx4,xt4)
      b4=matmul(xn4,y)
      call rsquare(y,machsup,x4,4,b4,rsq)
      write (unit=21,fmt=56) b4(1),b4(2),b4(3),b4(4),rsq
c  supersonic regression of s on m, m^2, m^3
      print*,"supersonic s..."
      do j=1,machsup
         y(j)=s(j+machsub)
      end do
      b4=matmul(xn4,y)
      call rsquare(y,machsup,x4,4,b4,rsq)
      write (unit=21,fmt=57) b4(1),b4(2),b4(3),b4(4),rsq
c  supersonic regression of cdo on m, m^2
      print*,"supersonic cdo..."
      do j=1,machsup
         y(j)=cdo(j+machsub)
      end do
      b4=matmul(xn4,y)
      call rsquare(y,machsup,x4,4,b4,rsq)
      write (unit=21,fmt=58) b4(1),b4(2),b4(3),b4(4),rsq
```

```
c  supersonic regression of k1 on m, m^2, m^3
       print*,"supersonic k1..."
       do j=1,machsup
          y(j)=k1(j+machsub)
       end do
       b4=matmul(xn4,y)
       call rsquare(y,machsup,x4,4,b4,rsq)
       write (unit=21,fmt=59) b4(1),b4(2),b4(3),b4(4),rsq
c  supersonic regression of k2 on m, m^2, m^3
       print*,"supersonic k2..."
       do j=1,machsup
          y(j)=k2(j+machsub)
       end do
       b4=matmul(xn4,y)
       call rsquare(y,machsup,x4,4,b4,rsq)
       write (unit=21,fmt=60) b4(1),b4(2),b4(3),b4(4),rsq
       print*," "
       print*,"regression analysis complete"
c  end of linear regression operations; the parametric curve fits are complete
and
c  saved to file "equations.txt"
51    format("clo = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2",21x,f5.3)
52     format("  s = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2",21x,f5.3)
53     format("cdo = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2",21x,f5.3)
54     format(" k1 = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2",21x,f5.3)
55     format(" k2 = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2",21x,f5.3)
56     format("clo = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2 + ",
     $      f9.6," * m^3",3x,f5.3)
57     format("  s = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2 + ",
     $      f9.6," * m^3",3x,f5.3)
58     format("cdo = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2 + ",
     $       f9.6," * m^3",3x,f5.3)
59     format(" k1 = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2 + ",
     $       f9.6," * m^3",3x,f5.3)
60     format(" k2 = ",f9.6," + ",f9.6," * m + ",f9.6," * m^2 + ",
     $       f9.6," * m^3",3x,f5.3)
70     format(2x,"mach #",6x,"clo",9x,"cdo",9x,"s",10x,"k1",11x,"k2")
71     format(2x,f5.2,4x,f8.5,4x,f8.4,4x,f8.5,4x,f8.5,4x,f8.5)
90     format(" subsonic equations",52x,"r^2")
91     format(" ")
       stop
       end
c
       subroutine invert_two_by_two(x, xi)
       implicit none
       real x(2,2), xi(2,2)
       real mult
       mult=1/((x(1,1)*x(2,2)) - (x(1,2)*x(2,1)))
       xi(1,1)=x(2,2)
       xi(1,2)= -1.0*x(1,2)
       xi(2,1)= -1.0*x(2,1)
       xi(2,2)=x(1,1)
       xi=xi*mult
       return
       end
c
       subroutine invert_three_by_three(x, xi)
       implicit none
       real x(3,3), xi(3,3)
       real a,b,c,d,e,f,g,h,k,z
       a = x(1,1)
       b = x(1,2)
       c = x(1,3)
```

```
       d = x(2,1)
       e = x(2,2)
       f = x(2,3)
       g = x(3,1)
       h = x(3,2)
       k = x(3,3)
       z=a*(e*k-f*h) - b*(d*k-f*g) + c*(d*h - e*g)
       xi(1,1) = (e*k - f*h)/z
       xi(1,2) = -1.*(b*k - c*h)/z
       xi(1,3) = (b*f - c*e)/z
       xi(2,1) = -1.*(d*k - f*g)/z
       xi(2,2) = (a*k - c*g)/z
       xi(2,3) = -1.*(a*f - c*d)/z
       xi(3,1) = (d*h - e*g)/z
       xi(3,2) = -1.*(a*h - b*g)/z
       xi(3,3) = (a*e - b*d)/z
       return
       end
c
      subroutine sgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      real a(lda,1)
c     sgefa factors a real matrix by gaussian elimination.
c
c     on entry
c        a       real(lda, n)
c                the matrix to be factored.
c        lda     integer
c                the leading dimension of the array  a .
c        n       integer
c                the order of the matrix  a .
c     on return
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that sgesl or sgedi will divide by zero
c                     if called.  use  rcond  in sgeco for a reliable
c                     indication of singularity.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas saxpy,sscal,isamax
c
c     internal variables
c
      real t
      integer isamax,j,k,kp1,l,nm1
c
c     gaussian elimination with partial pivoting
c
      info = 0
      nm1 = n - 1
```

```
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
c
c        find l = pivot index
c
         l = isamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
c
c        zero pivot implies this column already triangularized
c
         if (a(l,k) .eq. 0.0e0) go to 40
c
c           interchange if necessary
c
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
c
c           compute multipliers
c
            t = -1.0e0/a(k,k)
            call sscal(n-k,t,a(k+1,k),1)
c
c           row elimination with column indexing
c
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call saxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0e0) info = n
      return
      end
c
      integer function isamax(n,sx,incx)
c
c     finds the index of element having max. absolute value.
c     jack dongarra, linpack, 3/11/78.
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      real sx(*),smax
      integer i,incx,ix,n
c
      isamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      isamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
c
```

```
c        code for increment not equal to 1
c
      ix = 1
      smax = abs(sx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(abs(sx(ix)).le.smax) go to 5
         isamax = i
         smax = abs(sx(ix))
    5    ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
   20 smax = abs(sx(1))
      do 30 i = 2,n
         if(abs(sx(i)).le.smax) go to 30
         isamax = i
         smax = abs(sx(i))
   30 continue
      return
      end
c
      subroutine saxpy(n,sa,sx,incx,sy,incy)
c
c     constant times a vector plus a vector.
c     uses unrolled loop for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      real sx(*),sy(*),sa
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (sa .eq. 0.0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        sy(iy) = sy(iy) + sa*sx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sy(i) = sy(i) + sa*sx(i)
   30 continue
      if( n .lt. 4 ) return
```

```
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        sy(i) = sy(i) + sa*sx(i)
        sy(i + 1) = sy(i + 1) + sa*sx(i + 1)
        sy(i + 2) = sy(i + 2) + sa*sx(i + 2)
        sy(i + 3) = sy(i + 3) + sa*sx(i + 3)
   50 continue
      return
      end
c
      subroutine sscal(n,sa,sx,incx)
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to 1.
c     jack dongarra, linpack, 3/11/78.
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      real sa,sx(*)
      integer i,incx,m,mp1,n,nincx
c
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        sx(i) = sa*sx(i)
   10 continue
      return
c
c        code for increment equal to 1
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sx(i) = sa*sx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        sx(i) = sa*sx(i)
        sx(i + 1) = sa*sx(i + 1)
        sx(i + 2) = sa*sx(i + 2)
        sx(i + 3) = sa*sx(i + 3)
        sx(i + 4) = sa*sx(i + 4)
   50 continue
      return
      end
c
      subroutine sgedi(a,lda,n,ipvt,det,work,job)
      integer lda,n,ipvt(1),job
      real a(lda,1),det(2),work(1)
c
c     sgedi computes the determinant and inverse of a matrix
c     using the factors computed by sgeco or sgefa.
c
c     on entry
c
c        a       real(lda, n)
```

```
c                the output from sgefa.
c        lda     integer
c                the leading dimension of the array  a .
c        n       integer
c                the order of the matrix  a .
c        ipvt    integer(n)
c                the pivot vector from sgeco or sgefa.
c        work    real(n)
c                work vector.  contents destroyed.
c        job     integer
c                = 11   both determinant and inverse.
c                = 01   inverse only.
c                = 10   determinant only.
c     on return
c        a       inverse of original matrix if requested.
c                otherwise unchanged.
c        det     real(2)
c                determinant of original matrix if requested.
c                otherwise not referenced.
c                determinant = det(1) * 10.0**det(2)
c                with  1.0 .le. abs(det(1)) .lt. 10.0
c                or  det(1) .eq. 0.0 .
c     error condition
c
c        a division by zero will occur if the input factor contains
c        a zero on the diagonal and the inverse is requested.
c        it will not occur if the subroutines are called correctly
c        and if sgeco has set rcond .gt. 0.0 or sgefa has set
c        info .eq. 0 .
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas saxpy,sscal,sswap
c     fortran abs,mod
c
c     internal variables
c
      real t
      real ten
      integer i,j,k,kb,kp1,l,nm1
c
c     compute determinant
c
      if (job/10 .eq. 0) go to 70
         det(1) = 1.0e0
         det(2) = 0.0e0
         ten = 10.0e0
         do 50 i = 1, n
            if (ipvt(i) .ne. i) det(1) = -det(1)
            det(1) = a(i,i)*det(1)
c        ...exit
            if (det(1) .eq. 0.0e0) go to 60
   10       if (abs(det(1)) .ge. 1.0e0) go to 20
               det(1) = ten*det(1)
               det(2) = det(2) - 1.0e0
            go to 10
   20       continue
   30       if (abs(det(1)) .lt. ten) go to 40
               det(1) = det(1)/ten
               det(2) = det(2) + 1.0e0
```

```
            go to 30
   40       continue
   50    continue
   60    continue
   70 continue
c
c     compute inverse(u)
c
      if (mod(job,10) .eq. 0) go to 150
         do 100 k = 1, n
            a(k,k) = 1.0e0/a(k,k)
            t = -a(k,k)
            call sscal(k-1,t,a(1,k),1)
            kp1 = k + 1
            if (n .lt. kp1) go to 90
            do 80 j = kp1, n
               t = a(k,j)
               a(k,j) = 0.0e0
               call saxpy(k,t,a(1,k),1,a(1,j),1)
   80       continue
   90       continue
  100    continue
c
c        form inverse(u)*inverse(l)
c
         nm1 = n - 1
         if (nm1 .lt. 1) go to 140
         do 130 kb = 1, nm1
            k = n - kb
            kp1 = k + 1
            do 110 i = kp1, n
               work(i) = a(i,k)
               a(i,k) = 0.0e0
  110       continue
            do 120 j = kp1, n
               t = work(j)
               call saxpy(n,t,a(1,j),1,a(1,k),1)
  120       continue
            l = ipvt(k)
            if (l .ne. k) call sswap(n,a(1,k),1,a(1,l),1)
  130    continue
  140    continue
  150 continue
      return
      end
c
      subroutine sswap (n,sx,incx,sy,incy)
c
c     interchanges two vectors.
c     uses unrolled loops for increments equal to 1.
c     jack dongarra, linpack, 3/11/78.
c     modified 12/3/93, array(1) declarations changed to array(*)
      real sx(*),sy(*),stemp
      integer i,incx,incy,ix,iy,m,mp1,n
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c       code for unequal increments or equal increments not equal
c         to 1
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
```

```
        stemp = sx(ix)
        sx(ix) = sy(iy)
        sy(iy) = stemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c       code for both increments equal to 1
c
c       clean-up loop
c
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        stemp = sx(i)
        sx(i) = sy(i)
        sy(i) = stemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,3
        stemp = sx(i)
        sx(i) = sy(i)
        sy(i) = stemp
        stemp = sx(i + 1)
        sx(i + 1) = sy(i + 1)
        sy(i + 1) = stemp
        stemp = sx(i + 2)
        sx(i + 2) = sy(i + 2)
        sy(i + 2) = stemp
   50 continue
      return
      end
c
      subroutine rsquare(y,n,x,k,b,rsq)
       real y(n),x(n,k),b(k),rsq
       real yhat(n),yavg, sum, ssr, ssto
       integer i,j
c  calculate average of the components of the y matrix
       sum=0.0
      do i=1,n
         sum=sum+y(i)
       end do
       yavg=sum/n
c caclulate yhat matrix (the fitted values of y)
       yhat=matmul(x,b)
c calculate ssr (regression sum of error squares)
       sum=0.0
       do i=1,n
          sum=sum+((yhat(i)-yavg)**2)
       end do
       ssr=sum
c calculate ssto (total sum of error squares, not single stage to orbit!)
       sum=0.0
       do i=1,n
          sum=sum+((y(i)-yavg)**2)
       end do
       ssto=sum
c calculate r-squared, the coefficient of determination
       rsq=ssr/ssto
       return
       end
```