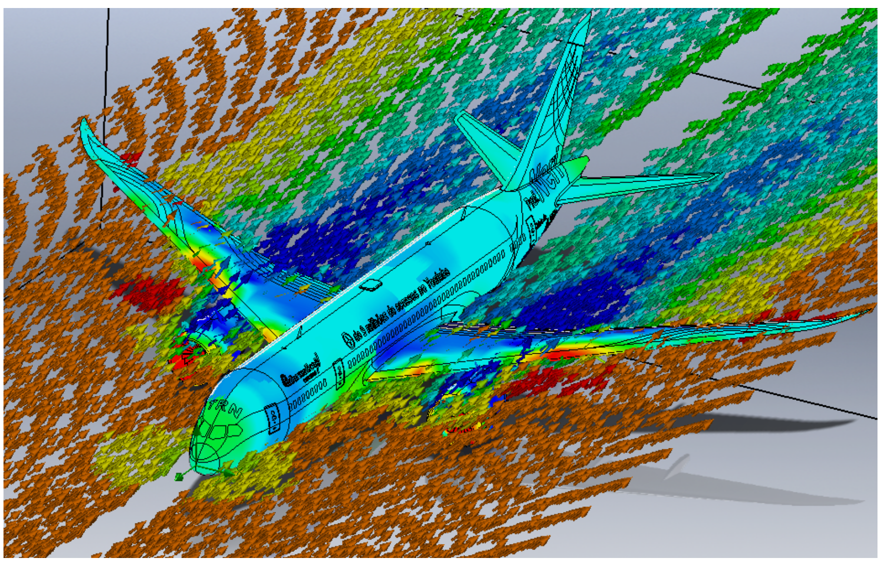
ASEN 2004

Experimental Laboratory 1: Low-Speed Aerodynamics of the Lockheed Martin F-16 and the Boeing 787-8 Dreamliner

Group 09



Forrest Barnes[*], Christopher Leighton[†], Michael Patterson[‡], and Nicholas Renninger[§]

University of Colorado at Boulder, Boulder, CO, 80310

The purpose of this lab was to examine the flight characteristics of a Boeing 787-8 and a Lockheed Martin F-16. To this end, three models (clean F-16, dirty F-16, and 787-8) were placed in a wind tunnel. The reaction forces and moment of the models in response to air flow were plotted against angle of attack ranging from -8° to 20°. The reaction forces and moments were then thoroughly analyzed to parse out various flight characteristics of the F-16 and the 787-8. We found that the 787-8 was statically and dynamically stable, and the F-16 was not. The S.M. of the F-16 was about -15.7%, while the S.M. of the 787-8 was 65%. Just as expected, we found that the dirty model of the F-16 had higher drag and lower lift than the clean model, though the difference diminished at high angles of attack. This may have been influenced by ground effects due to the small size of ITLL Low-Speed wind tunnel and the relatively large size of our models. Unexpectedly, we found that the $(L/D)_{max}$ of the clean F-16 was greater than the $(L/D)_{max}$ of the 787-8, though this result was caused by ground effects due to the small size of the wind tunnel, among other factors. Interestingly, the dirty F-16 model didn't even come close to matching the clean F-16 model or the 787 model in $(L/D)_{max}$ (See Figure 3 for details). For these basic characterizations, the ITLL wind tunnel was adequate to determine major flight characteristics, but would be ill-suited for a more detailed analysis.

Nomenclature

α = Angle of Attack [°]
$\bar{c}$ = Mean Aerodynamic Chord length [m]
δ = uncertainty in a calculated quantity
ρ_∞ = Local atmospheric density [kg/m^3]
σ = Standard Deviation
σ = variance in a measured quantity
A = Axial Force [N]
C_D = Coefficient of Drag
C_L = Coefficient of Lift
COL = Aerodynamic Center of Lift [N]
COM = Center of Mass
D = Drag [N]
d = Distance between model center of gravity and sting endpoint [m]
G = number of groups that took the same data point, for a given model
K = Scaling factor from real plane to model [N]
L = Lift [N]
N = Normal Force [N]
N = number of data points per α taken by the LabVIEW VI, in this case it was 20
P_m = Pitching Moment [N m]
$S.M.$ = Static Margin [%$\bar{c}$]
V_{min} = Minimum landing velocity [m/s]
V_{stall} = Stall velocity [m/s]
X = measured quantity like force, α, or pressure

I. Introduction

Wind tunnels are an important tool in determining the aerodynamic properties of airfoils, wings, and the entire aircraft. In this lab, the ITLL wind tunnel was used to analyze three different aircraft models: one of a "dirty" F-16, which includes under-wing payload such as weapons and fuel tanks; one of a "clean" F-16, devoid of any add-ons; and a third of the Boeing 787-8 commercial transport . Ultimately, our goal was to aerodynamically characterize and compare each of these three aircraft. The models were separately affixed to

*104388401
†102267575
‡107475537
§105492876

a sting balance inside of the wind tunnel's test section as shown in Figure 1, which measured the normal and axial forces, along with the moment, acting on the model in question. We then gathered forces and moments data vs. various angles of attack and airspeeds. With this data we were able to model each design's lift, drag, and moment coefficients as a function of angle of attack. From these, we were able to investigate several aspects of each aircraft's aerodynamic performance, such as their longitudinal stability, their lift to drag ratios, their static margins (S.M.), and the overall relationship between their lift and drag. Through these sorts of analyses, we were able to characterize and compare different facets of each aircraft's performance, which ultimately brought us to a more complete understanding of why the F-16 and the 787 are designed differently and how they've been optimized to match their mission requirements. Our investigations also gave us an opportunity to determine the specific effects of "dirtying" the F-16, and to make some conclusions about why the F-16's add-ons are designed and placed as they are.

II. Theory

The first step towards finding an aircraft's lift, drag, and moment coefficients is to calculate the total lift, drag, and pitching moment based on data taken from the sting balance. The sting is designed to measure only the moment and normal and axial forces acting on it. When completely horizontal, the measured normal force is equivalent to the lift and the axial forces is equivalent to the lift, but at any non-zero angle of attack, the total lift and drag become trigonometric functions of the measured normal and axial forces.

$$D = N * sin\alpha + A * cos\alpha \tag{1}$$
$$L = N * cos\alpha - A * sin\alpha \tag{2}$$

From the calculations of lift and drag, and from other atmospheric measurements directly recorded by the wind tunnel, the coefficients of lift and drag corresponding to a given aircraft configuration can be calculated as:

$$C_D = \frac{2D}{\rho_\infty V_\infty^2 S} \tag{3}$$
$$C_L = \frac{2L}{\rho_\infty V_\infty^2 S} \tag{4}$$

The pitching moment acting on a model attached to the sting can be calculated in a similar vein using Eqn. 6, albeit with one complication: the point about which the sting measures moments is offset a known distance from the center of gravity of the model aircraft. As such, prior to analysis, the raw pitching moment recorded by the wind tunnel must be transformed using Eqn. 5, which is a formulation of the Parallel Axis Theorem:

$$P_{m,cg} = P_m - N * d \tag{5}$$

The adjusted pitching moment can be used to determine the aircraft's moment coefficient at a given angle of attack.

$$C_M = \frac{2P_{m,cg}}{\rho_\infty V_\infty^2 S\bar{c}} \tag{6}$$

Having calculated a given aircraft's lift, drag, and moment coefficients at known angles of attack, we are in a position to perform several critical analyses on the aircraft.

First, we can now estimate the aircraft's *longitudinal stability*, which is essentially a measure of an aircraft's tendency to rotate back toward or continue away from zero angle of attack when disturbed from a perfectly horizontal position. Quantitatively, we determine longitudinal stability by examining how pitching moment varies with attack angle. If the moment acting on the aircraft is negative when its angle of attack is positive (and vice versa), the pitching moment effectively torques the plane back to a horizontal state, in which case it can generally be said to have longitudinal stability. Conversely, if the moment acting on the aircraft is positive when the attack angle is also positive, the aircraft will tend to continue rotating away from a zero degree angle of attack. Graphically, this discussion can be summarized as follows: given a plot describing the pitching moment acting on an aircraft as a function of attack angle (as in Figure. 2d), a negative slope will generally correspond to designs that are stable and positive slopes will correspond to designs that are unstable.

Second, we are able to estimate a given aircraft's *static margin*, $S.M.$, which is the horizontal distance between the center of mass/gravity and the center of aerodynamic lift divided by the $\bar{c}$. A negative $S.M.$ happens when the center of lift is closer to the nose of the airplane than the center of gravity.

In knowing C_L, we are also able to calculate an aircraft's minimum landing velocity. for civilian and military aircraft respectively, this is determined as follows.

$$V_{stall} = \sqrt{\frac{2W}{\rho_\infty S C_{L,max}}} \tag{7}$$

$$V_{land,min,civilian} = 1.3 * V_{stall} \tag{8}$$

$$V_{land,min,military} = 1.2 * V_{stall} \tag{9}$$

Here, $C_{L,max}$ is found by plotting C_L against angle of attack and simply choosing the highest lift coefficient, which generally occurs at some positive attack angle. It should be noted that our analysis does not account for the presence of flaps, which dramatically increase the lift coefficient.

The ratio between the lift coefficient and the drag coefficient, L/D, is another incredibly important property of a given design which is a measure of the aerodynamic efficiency of the aircraft. In general, a designer aims to maximize L/D; one way this can be done is by increasing the wings' aspect ratio, and thus we would expect the 787 to have a higher $(L/D)_{max}$ than the F-16. The coefficients we have calculated will enable us to determine if our expectation is correct, calculating L/D using Eqn. 10:

$$L/D = \frac{C_L}{C_D} \tag{10}$$

Our analysis involves applying Eqns. 1 - 10 to every set of data collected by every group in the class. Consequently, for a given model aircraft at a given angle of attack, we encountered multiple data points recorded by different groups. To make further analysis possible, we averaged the data from different groups taken at the same attack angle and the same aircraft using a weighted mean. From each data set taken for a particular model, we first averaged down the 20 measurements taken for each angle of attack using Eqn. 11, and computed σ for this average using 12. Then, to combine multiple groups' data, we needed a weighted mean to give more weight to a group's data if it had a smaller variance (a greater W), done as follows in Eqn. 14[3] :

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{11}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{12}$$

$$W_j = \frac{1}{\sigma_j^2} \tag{13}$$

$$\bar{X} = \frac{\sum_{j=1}^{G} \bar{x_j} * W_j}{\sum_{j=1}^{G} W_j} \tag{14}$$

$$\tag{15}$$

This weighted average measurement, $\bar{X}$, was the final measurement value used for each α measured. This weighted average was computed for every measures quantity, including N force, Axial Force, Airspeed, angle of attack, and other atmospheric variables. Now, to compute the uncertainty of each of these measurements, $\sigma_{\bar{X}}$, we used the weighted variance as calculated in Eqn. 16:[3]

$$\sigma_{\bar{X}} = \frac{1}{\sqrt{\sum_{j=1}^{G} W_j}} \tag{16}$$

In order to compute the uncertainty in calculated quantities like C_L, C_D, and C_M, we used the general error propagation formula as follows from the Taylor Uncertainty Analysis Text used in ASEN 2012:[3]

$$\delta_{calculated\,variable} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 \delta_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \delta_y^2 + \left(\frac{\partial f}{\partial z}\right)^2 \delta_z^2 + ...} \tag{17}$$

where x, y, and z are all parameters involved in the formulation of the calculated variable. The process of taking partials and summing in quadrature, as in Eqn. 17, was done symbolically in MATLAB for every calculated quantity. The implementation of the averaging, variance, and uncertainty calculation algorithm can be found in the code included in the appendix.

III. Experimental Apparatus and Procedure

Each team was instructed to take measurements from one of the three model aircraft; our team was assigned the Boeing 787. The sting balance, which is a tool capable of measuring normal and axial forces as well as moments, was attached to the floor of the ITLL wind tunnel's test section; each model was then attached to the sting such that its nose pointed toward the front end of the wind tunnel as seen in Figure 1. The angle of attack of the sting balance and the model is variable and can be precisely tuned with the help of the LabVIEW VI. The center of the sting was offset a known distance from each model's center of gravity, the implications of which are described by Eqn. 5.

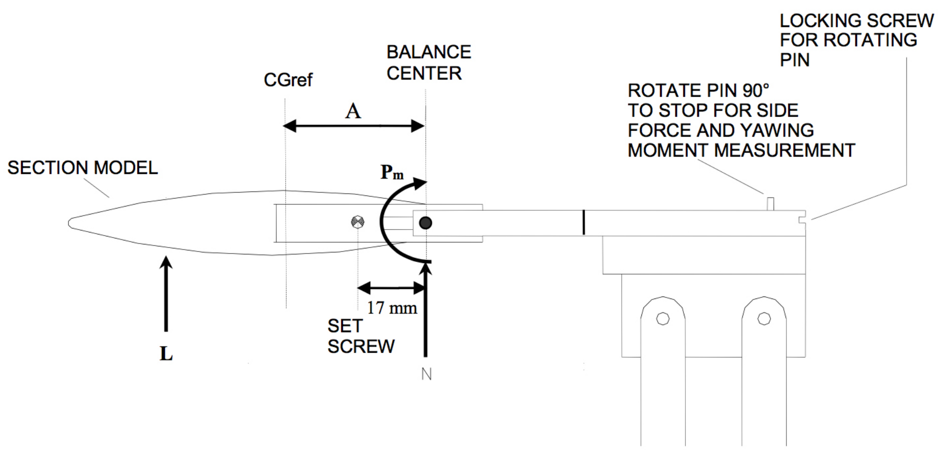


Figure 1: Diagram of the experimental setup,[1] oriented such that the wind would blow from left to right. The model is depicted as the curved oblong region at the left, and the two vertical bars at the bottom of the sting are attached to the floor of the test section.

Our team began by using the LabVIEW VI to ensure that the sting balance and the model were initially set to an attack angle of zero degrees. The pressures being measured were zeroed, as were the forces and moment being measured by the sting balance. In order to eventually subtract them from our analysis, we left the wind tunnel off during our first set of measurements in order to determine the force and moment contributions created by gravity acting on the model and on the sting. Our group was assigned every odd integer attack angle between -8° and 20°, as well as 0°. Next, we set the wind tunnel to a flow velocity of 25 m/s and recorded the moment and forces at each of the same set of angles. Each group followed the basic same procedure, though half of them collected data at odd angles of attack and the other half collected data at even angles, ensuring a complete range of data. Additionally, the class was effectively divided up into three groups, each of which investigated only one of the three models. Ultimately, the class collected multiple complete sets of data describing the forces acting on all three models at every integer angle of attack between -8° and 20°; it is this superset of data that we reduced using Eqns. 11 - 17, and analyzed using Eqns. 1 - 10 and the analytic procedure outlined in Section II.

IV. Summary of Results

The results of the analysis of the wind tunnel data are summarized here in two forms: in Tables 1 - 3 and in the sub-figures of Figure 2. Tables 1 - 3 present the Static Margin, the Longitudinal Stability, the Lift Slope, and the landing speeds of the model and actual plane for each model tested. The landing speeds presented are calculated in two different ways, and are presented to illustrate the differences that can arise from the use of different assumptions. The fourth column (model landing speed w/ actual model weight) represent the landing speed as calculated using Eqn. 7, with the weight of the model calculated by finding the maximum lift force exerted on the model (this happens at $C_{L,max}$). Then, the sixth column (landing speed of the real plane scaled up from the landing speed of the model based on the model's actual weight) is calculated by scaling up the landing velocity of the model calculated using the actual weight of the model (column four in each table):[2]

$$V_{land,model} = V_{land,full-scale} * \sqrt{K} \implies V_{land,full-scale} = \frac{V_{land,model}}{\sqrt{K}} \tag{18}$$

The fifth column (model landing speed w/ scaled model weight) is calculated again using Eqn. 7, but with the weight of the model scaled down from the actual plane using the following:[2]

$$W_{model} = W_{full-scale} * K^3 \tag{19}$$

Eqn. 19, while accurately giving the mass of a perfect model of the full-size plane, does not necessarily accurately reflect the properties of the model used in lab, as it is not a perfect mass model. Despite this inaccuracy, the landing speeds of the actual aircraft would appear to be more reasonable when using the scaled down weight of the real plane as the weight of the model aircraft (cols. five and seven).

The figures contained in Figure 2 demonstrate, C_L, C_D, and C_M for the models, as well as the drag polar for each model. L/D, plotted in Figure 3, can be found in the Appendix and gives a good overview of the aerodynamic efficiency of each model. Below each plot is a description of the relevant information and conclusions that can be drawn from the analysis of the plot.

Table 1: Tabulation of Results of Analysis for Clean LM F-16

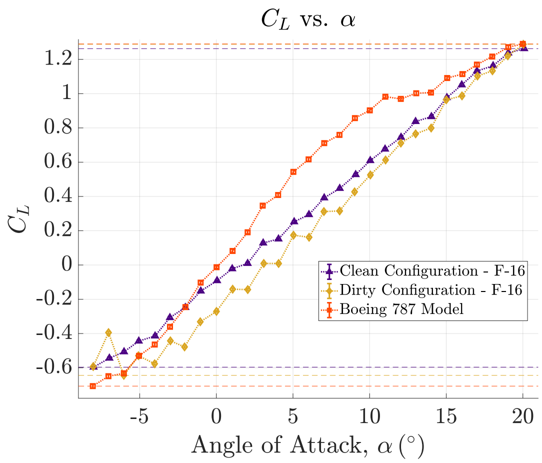
S.M. ($\alpha = 0$) [$\%\bar{c}$]	$\frac{dC_M}{d\alpha}$ ($\alpha = 0$) [$\frac{1}{\circ}$]	$\frac{dC_L}{d\alpha}$ ($\alpha = 0$) [$\frac{1}{\circ}$]	**Landing Speed** *Model* Actual Model Weight [$knots$]	**Landing Speed** *Model* Scaled Full-Scale Weight [$knots$]	**Landing Speed** *Real Plane* Actual Model Weight [$knots$]	**Landing Speed** *Real Plane* Scaled Full-Scale Weight [$knots$]
-15.5	9.9E-3	63.6E-3	58.3	30.1	404.0	208.6

Table 2: Tabulation of Results of Analysis for Dirty LM F-16

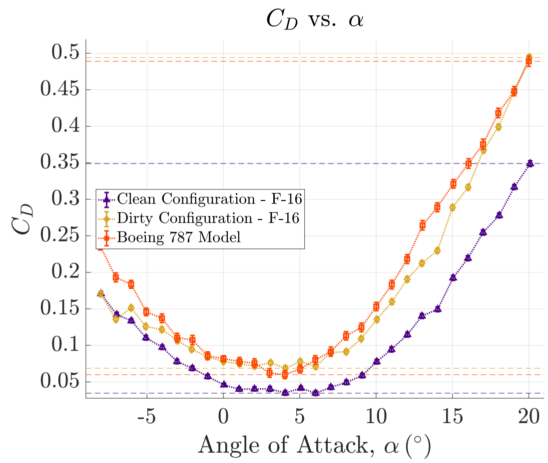
S.M. ($\alpha = 0$) [$\%\bar{c}$]	$\frac{dC_M}{d\alpha}$ ($\alpha = 0$) [$\frac{1}{\circ}$]	$\frac{dC_L}{d\alpha}$ ($\alpha = 0$) [$\frac{1}{\circ}$]	**Landing Speed** *Model* Actual Model Weight [$knots$]	**Landing Speed** *Model* Scaled Full-Scale Weight [$knots$]	**Landing Speed** *Real Plane* Actual Model Weight [$knots$]	**Landing Speed** *Real Plane* Scaled Full-Scale Weight [$knots$]
-15.9	14.3E-3	89.9E-3	58.4	29.7	404.3	205.5

Table 3: Tabulation of Results of Analysis for Boeing 787-8

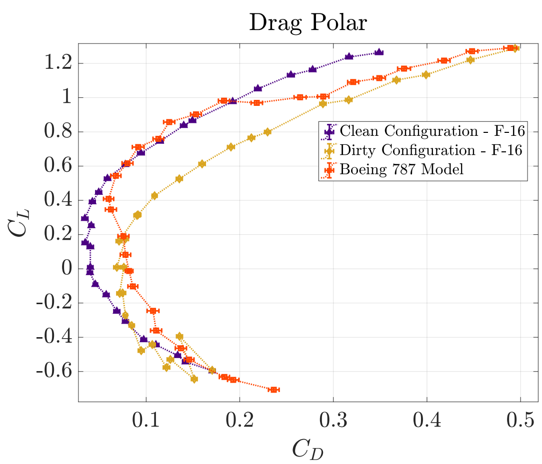
S.M. ($\alpha = 0$) [$\%\bar{c}$]	$\frac{dC_M}{d\alpha}$ ($\alpha = 0$) [$\frac{1}{\circ}$]	$\frac{dC_L}{d\alpha}$ ($\alpha = 0$) [$\frac{1}{\circ}$]	**Landing Speed** *Model* Actual Model Weight [$knots$]	**Landing Speed** *Model* Scaled Full-Scale Weight [$knots$]	**Landing Speed** *Real Plane* Actual Model Weight [$knots$]	**Landing Speed** *Real Plane* Scaled Full-Scale Weight [$knots$]
65.0	-57.6E-3	88.7E-3	63.1	15.0	945.9	225.3



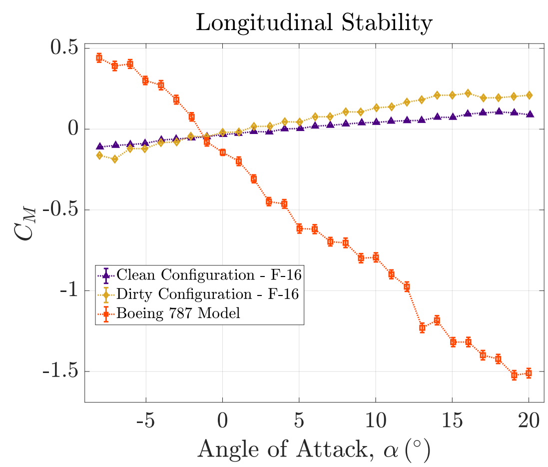
(a) Plot showing the lift coefficients tested at $\mathbf{V_\infty = 25\,m/s}$. Because of the small throat diameter of the wind tunnel, the models actually constrict and accelerate the flow at high α. *This means the as the plane approaches stall, the air is effectively reattached due to the constriction of the tunnel by the model, leading to artificially high value of C_L. For the 787, the wind tunnel effect is most visible starting at approximately α = 10°, where a jump in the lift slope happens at about $C_L = 1$. For the F-16 models, the effect of the wind tunnel can be seen to a lesser degree than with the 787 as α approaches 10-15°, where the lift slopes stall and then jump back up.*



(b) Plot showing the drag coefficients tested at $\mathbf{V_\infty = 25\,m/s}$. *As expected, the clean model has a lower C_D than did the dirty F-16 model across the entire α range. However, the dirty and the clean F-16 models experienced similar behavior at lower α, most likely because at higher α more of the weapons payload is exposed to the flow and thus mainly increases the profile drag. The 787 surprisingly had the highest C_D, which goes against expectations of its extremely streamlined design. This discrepancy between our expectations and the experimental results above is likely due to the much rougher surface (on account of the 3D printed material), which would greatly increase the profile drag on the 787 model, and the difference in Reynold's number.*



(c) Plot showing the Drag Polars for the models tested at $\mathbf{V_\infty = 25\,m/s}$. *The effect of the very narrow wind tunnel throat is clearly visible in the 787 data. For small values of C_D, where $C_L \leq 1$ as seen in Fig. 2a, the 787 experiences a drag polar similar to that of the F-16. However, once the tunnel effect comes into play at higher α (and thus at higher C_D), the 787's drag polar switches regimes to match the Dirty F-16 at about $C_D = 0.3$. Otherwise, the data from both F-16 models largely matches expectations, with the dirty F-16 experiencing higher C_D at the same C_L.*



(d) Plot showing the Longitudinal Stability, C_M vs. α, for the models tested at $\mathbf{V_\infty = 25\,m/s}$. The Static Longitudinal Stability, $\frac{dC_M}{d\alpha}$, is monotonically negative for the 787, and monotonically positive for both of the F-16 models. *As the static longitudinal stability is positive for the F-16 models, their aerodynamic performance can be classified as unstable. As the static longitudinal stability is more positive in the Dirty F-16, it is more aerodynamically unstable. The 787, on the other hand, has positive static longitudinal stability and thus its aerodynamic performance can be classified as very stable.*

Figure 2: Results of Analysis

V. Discussion

The dirty F-16 model consistently had a greater C_D, lower C_L, and lower dynamic stability. This caused it to have a lower L/D. Our drag polar (Figure 2c) also shows that the drag of the dirty F-16 increases more than the drag of the clean F-16 at higher lift values. This is consistent with expectations. The dirty F-16 is still far better than a hypothetical F-16 with over-wing stores rather than the under-wing stores used on our model. Adding stores to the bottom of the wing has far lower impact on the performance of the aircraft that stores on top of the wing. Adding stores to the top of the wing disturbs laminar flow and decreases C_L by increasing the pressure on the top of the wing.

From Eqn. 9, the landing speed of the F-16 is 1.2 times the V_{stall} value. As seen in Table 1 and Table 2 the landing speed of the dirty model is slightly lower than the than that of the clean model, with the dirty F-16 model landing at 29.7 knots and the clean F-16 model landing at 30.1 knots. Here, we decided to use the landing speed calculated using the scaled-down weight of the full-size aircraft as the weight of the model. The results seem to a contradict our expectations for the F-16 models, as the dirty model should weigh more than the clean model and should have a theoretically smaller $C_{L,max}$; both of these factors should increase the landing speed for the dirty configuration as compared to the clean configuration. To explain this contradiction we can look at Figure 2a. At about an angle of attack of 13 degrees, the model appears to stall then regain lift. Before this point the C_L for the dirty model is significantly lower than the clean, however past this point the values converge and the dirty C_L eventually passes the clean C_L, a result that doesn't agree with the theoretical decrease in C_L that should be more evident for the dirty model. This difference between the experimental data and theoretical considerations can be explained by the "ground effect" due to the limited throat area of the wind tunnel, where at high α the blockage of the tunnel causes reduced induced drag and flow reattachment on the models. These landing speeds are extremely high, but they also represent an F-16 landing with no flaps, gear drag or reverse thrust. In effect this represents the speeds that would be seen in an emergency landing without landing gear. It is also worth noting that these models are not mass-modeled, potentially causing issues calculating V_{stall}.

As can be seen in Table 1, the landing speed of the full size clean F-16 is 208.6 knots with a landing weight of 30,000lbs, at wind tunnel speeds, at standard atmosphere conditions, and at 5430 ft. The slope of the moment coefficient is slightly positive (static longitudinal stability), which implies a tendency to move away from a zero angle of attack once disturbed. The static longitudinal stability of the clean F-16 model is very low at 0.0093/°, resulting in the instability you would expect of fighter aircraft, and the lift slope is 0.0636/°. As seen in Table 2, the longitudinal static stability for the dirty configuration is 0.0143/°and the lift slope is 0.0899/°.

The static margin of the clean F-16 (as seen in Table 1) is $-15.5\,\%\bar{c}$ which is almost the same as the static margin of the dirty F-16 which is $-15.9\,\%\bar{c}$. These values are too negative to be completely accurate, but they still allow us to characterize the general stability of the aircraft. Their magnitudes are too large due to the problems with the wind tunnel, and also because of the fact that the F-16 models are not precisely mass-modeled to the real airplanes. These very negative static margins result in the instability seen in Figure 2d. This is due to the fact that as the angle changes the moment produced by the aerodynamic center about the center of mass pushes the plane further from a 0 angle of attack. Also, as expected, the slightly more negative static margin of the dirty F-16 means it is slightly more unstable than the clean F-16.

From Eqn. 8, we are able to estimate the minimum landing speed of the model 787. Based on a $C_{L,max}$ of 1.29, we determined the minimum landing speed of our model 787 to be 15.0 knots for the model weight scaled from the full size 787 case.

The minimum landing speed of the full size 787 at a weight of 360,000lb and no flaps, gear, or reverse thrust is 225.3 knots. This is a very large number, but as with the F-16, this is representative of an emergency landing with no flaps or gear. The size of the wind tunnel and the mass/mass distribution of the model affected this measurement as well, in largely the same way as described before for the F-16.

In contrast to the F-16, our model of pitching moment vs. angle of attack for the 787 indicates remarkable longitudinal stability. Where the F-16 had a positive, and therefore unstable, $\frac{dC_M}{d\alpha}$ of 9.9E-3/°, the 787 had a negative slope of -57.6E-3/°. From Figure 2d, we can see that the 787's equilibrium angle of attack is somewhere around -1.5°, and the lift slope is 88.7E-3/°. The stability of the 787 as compared to the F-16 was to be expected, and makes sense given their very different missions. This very negative longitudinal stability means the 787 is a very stable airplane, much more so than the F-16 configurations. A large, civilian transport jet like the 787 would need to be stable to ensure smooth and resilient operation in all conditions. On the other hand, the F-16 needs to be at best neutrally stable so that control input gain is large and so the aircraft responds rapidly to the pilot in high-G dogfighting maneuvers, ensuring air superiority over less nimble aircraft.

As can be seen in Table 3, the static margin for the 787 is $65.0\,\%\bar{c}$. These values are too large to be completely accurate, but they still allow for a general characterization of the stability of the aircraft. Their magnitudes are too large due to the problems with the wind tunnel, and also because of the fact that the 787 model is not precisely mass-modeled to the real airplane. The large, positive static margin results in a statically and dynamically stable aircraft. The moment of the aerodynamic center about the center of gravity will torque the plane back to a stable angle of attack.

Contrary to expectations, $(L/D)_{max}$ for the clean F-16 was greater than for either the dirty variant or for the 787. This is seen in Figure 3, where we can also observe that L/D peaks at around 5° for the 787, but not until around 7° for the clean F-16. While the F-16 had a higher $(L/D)_{max}$, in the α range that it normally flies in, from [-5, 5]°, the 787 has a greater L/D than either F-16 configuration (as seen in Figure 3). Also of note in the same figure, the effect of the external payload on the aerodynamic efficiency of the dirty F-16 is massive. The L/D for the dirty F-16 at $\alpha \geq 5°$ is less than half that of the L/D of the clean F-16 in the same α range, which is clearly shown in Figure 3.

VI. Conclusion

Through the wind tunnel analysis of the the Boeing 787-8 Dreamliner and the Lockheed Martin F-16 Falcon, we found that the basic performance identified in our analysis would allow each plane to meet its specific design requirements. The 787 proves to be a very stable aircraft with low induced drag and high L/D, all characteristics beneficial to a commercial transport. The F-16 Falcon on the other hand showed itself to be an unstable/agile aircraft with a high maximum angle of attack and low profile drag, all characteristics of a good fighter jet.

In analyzing the wind tunnel data, we found that when analyzing a model in a small wind tunnel, such as the ITLL wind tunnel, undesirable forces are exerted on the model at high angles of attack. These forces are induced by the "ground effects" of the tunnel floor, walls, and ceiling, along with the physical restriction the large cross section places on the airflow inside the test section. These effects led to artificially high values of $C_{L,max}$, which ended up affecting the minimum landing speeds calculated for each airplane. Also, the fact that the models were not exact mass models contributed to the inaccurately high values calculate for the S.M. and the longitudinal stability.

Due to these undesirable effects, if the lab is to be repeated it should be performed in a larger wind tunnel. Along with a larger wind tunnel some other improvements can be made to improve the overall accuracy of the lab. To closer resemble the flight conditions of these aircraft the airspeed should be increased to better match the Reynold's number of the larger aircraft. The relatively low airspeed and low Reynolds number can introduce some undesirable effects such as the lammer air flow bubble. To more accurately account for the coefficient of drag, models that better resemble the skin drag of the actual aircraft should be used, as the 3D printed surface of the 787 model is far rougher than that of the actual plane. These changes would all help to remove the inconsistencies seen in this lab and any similar labs we may perform in the future.

Appendix

Additional Figures

Title page image: CFD image of the flow over the wings of the 787-8 Dreamliner, along with the pressure distribution over the surface of the plane, that we calculated in Solidworks

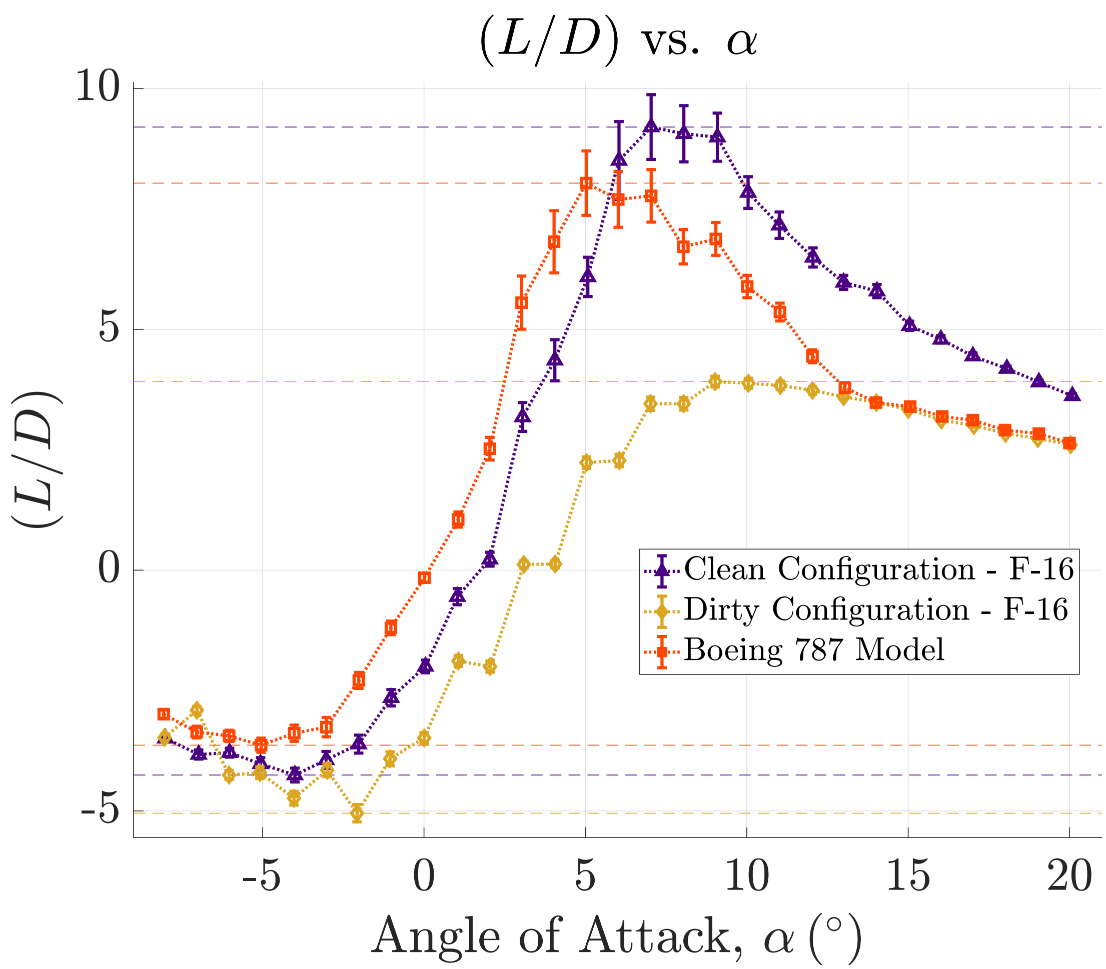


Figure 3: L/D vs. angle of attack tested at $V_\infty = 25\,\mathrm{m/s}$. *The 787 has a higher L/D up until $\alpha = 5°$, where the clean F-16 takes over and exhibits the maximum L/D of all of the models. The dirty F-16 is also shown to be much less efficient than the clean F-16 model, as once $\alpha \geq 5°$, the drag on the dirty F-16 begins to cause it to diverge from the clean F-16 model and have less than half the L/D_{max}.*

Acknowledgments

We would like to thank Dr. Gerren and the TAs for all of their help and guidance in completing this lab. Without them, we would be lost.

References

[1] Gerren, Donna. ASEN 2004 Experimental Laboratory 1: Low-Speed Aerodynamics of the Lockheed Martin F-16 and the Boeing 787 Dreamliner. CU Boulder, 2017. PDF.

[2] Powers, Bradford W. "About the Size of It". Model Aviation: 8-12

[3] Baton, U. N., An introduction to error analysis by John R. Taylor, Strain, vol. 33, 1997, pp. 133134.

MATLAB Code

Main Script and Setup

```matlab
%%% !! README !!
%%% The code MUST HAVE that the following directory structure exits so that
%%% it can find and save all of the data, results, and plots. The following
%%% directories must also all be in the same parent directory (for example
%%% a directory called "Lab 1" might contain the Code/, Figures/, and Data/
%%% directories as such:
%%%
%%%
%%% ../Lab 1/Code/ - contains all of the .m files
%%% ../Lab 1/Figures/ - save location for .pdf figure files
%%% ../Lab 1/Data/ - where all data is saved. This directory must contain
%%%                  the following directories:
%%%
%%%                  ../Lab 1/Data/Test Data/S_011 - section 1 data
%%%                  ../Lab 1/Data/Test Data/S_012 - section 2 data
%%%                  ../Lab 1/Data/Test Data/S_013 - section 3 data
%%%
%%%
%%%
%%% Last Modified: 3/8/17
%%% ASEN 2004 Lab #1
%%% Author: Nicholas Renninger -  Lab 011
%%%


%% Housekeeping

close all
clear variables
clc


%% Wing Geometry Constants %%%
[S_F16_MODEL, MAC_F16_MODEL, ...
 S_787_MODEL, MAC_787_MODEL, ...
 W_F16_MODEL, W_787_MODEL, ...
 DIST_CG_F16_CLEAN, ...
 DIST_CG_F16_DIRTY, ...
 DIST_CG_787, SCALE_F16, SCALE_787] = Define_Model_Geometry;


%% Plot Constants %%%
set(0, 'defaulttextinterpreter', 'latex')
FONTSIZE = 30;
LINEWIDTH = 2;
MARKERSIZE = 7;


% Read in Data from Excel Files
[test_matrix] = ASEN_2004_Lab_1_readInput();

% Breaking up cell array into its constituent structs for readability
```

```matlab
clean = test_matrix{1};
dirty = test_matrix{2};
seven87 = test_matrix{3};


% Lift & Drag Analysis

%% Clean F-16 Model %%%
fprintf('\nAnalyzing Clean F-16 Data...')
clean = LiftDragMoment(clean, DIST_CG_F16_CLEAN,...
                              S_F16_MODEL, MAC_F16_MODEL);
fprintf('\nSuccessfully Analyzed Data.')

%% Dirty F-16 Model %%%
fprintf('\n\nAnalyzing Dirty F-16 Data...')
dirty = LiftDragMoment(dirty, DIST_CG_F16_DIRTY,...
                              S_F16_MODEL, MAC_F16_MODEL);
fprintf('\nSuccessfully Analyzed Data.')

%% 787 Model %%%
fprintf('\n\nAnalyzing 787 Data...')
seven87 = LiftDragMoment(seven87, DIST_CG_787,...
                              S_787_MODEL, MAC_787_MODEL);
fprintf('\nSuccessfully Analyzed Data.')


%% Calculate Static Longitudinal Stability, dCL/dAoA, S.M., & L/D
fprintf(['\nCalculating Static Longitudinal Stability,', ...
         'dCL/dAoA, S.M., & L/D ...\n']);

isMilitary = true;
clean = auxilaryCalculations(clean, S_F16_MODEL, W_F16_MODEL, ...
                             SCALE_F16, isMilitary);

dirty = auxilaryCalculations(dirty, S_F16_MODEL, W_F16_MODEL, ...
                             SCALE_F16, isMilitary);

isMilitary = false;
seven87 = auxilaryCalculations(seven87, S_787_MODEL, W_787_MODEL, ...
                               SCALE_787, isMilitary);

fprintf('Done\n');


%% Plotting
fprintf('\nPlotting...\n')

colorVecs = [0.294118 0 0.509804; % indigo
             0.854902 0.647059 0.12549; % goldenrod
             1 0.270588 0]; % orange red


%%%%%%%%%%%%% C_L vs. AoA %%%%%%%%%%%%%
plot_CL_CD_AoA(clean, dirty, seven87, ...
               FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)
```

```
%%%%%%%%%%%%% C_M vs. AoA %%%%%%%%%%%%%
plot_CM_AoA(clean, dirty, seven87, ...
            FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE);


%%%%%%%%%%%%% C_L vs. C_D %%%%%%%%%%%%%
plot_drag_polar(clean, dirty, seven87, ...
                FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)

%%%%%%%%%%%%% Lift over Drag vs. AoA %%%%%%%%%%%%%
plot_L_over_D_AoA(clean, dirty, seven87, ...
                  FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)


%% Printing Results of Analysis

rowNames = {'Static Margin @ AoA = 0 [%MAC]:', ...
             'Longitudinal Stability @ AoA = 0 [1/deg]:', ...
             'Lift Slope @ AoA = 0 [1/deg]:', ...
             'Min. Landing Speed - Model (actual Model) [knots]:', ...
             'Min. Landing Speed - Model (theor. Mass Model) [knots]:', ...
             'Min. Landing Speed - Actual Plane (from actual Model) [knots]:', ...
             'Min. Landing Speed - Actual Plane (from theor. Mass Model) [knots]:'};

filePath = '../results.txt';
fid = fopen(filePath, 'w+');
m_p_s_to_knots = 1.94384;

fprintf('Printing Results of Analysis to %s\n', filePath);

% print SM
data1 = clean.SM_at_0;
data2 = dirty.SM_at_0;
data3 = seven87.SM_at_0;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
             rowNames{1}, data1, data2, data3);

% print Longitudinal Stability
data1 = clean.longit_stability_at_0;
data2 = dirty.longit_stability_at_0;
data3 = seven87.longit_stability_at_0;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
             rowNames{2}, data1, data2, data3);

% print dCL / dAoA
data1 = clean.dCL_dAoA;
data2 = dirty.dCL_dAoA;
data3 = seven87.dCL_dAoA;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
             rowNames{3}, data1, data2, data3);

% Print landing speeds
```

```matlab
data1 = clean.V_land_min_real_model * m_p_s_to_knots;
data2 = dirty.V_land_min_real_model * m_p_s_to_knots;
data3 = seven87.V_land_min_real_model * m_p_s_to_knots;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
            rowNames{4}, data1, data2, data3);

data1 = clean.V_land_min_theoretical_model * m_p_s_to_knots;
data2 = dirty.V_land_min_theoretical_model * m_p_s_to_knots;
data3 = seven87.V_land_min_theoretical_model * m_p_s_to_knots;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
            rowNames{5}, data1, data2, data3);

data1 = clean.V_land_realPlane * m_p_s_to_knots;
data2 = dirty.V_land_realPlane * m_p_s_to_knots;
data3 = seven87.V_land_realPlane * m_p_s_to_knots;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
            rowNames{6}, data1, data2, data3);

data1 = clean.V_land_realPlane_theory * m_p_s_to_knots;
data2 = dirty.V_land_realPlane_theory * m_p_s_to_knots;
data3 = seven87.V_land_realPlane_theory * m_p_s_to_knots;
fprintf(fid, '%s \r\nclean: %f \t dirty: %f \t 787: %f \r\n\r\n', ...
            rowNames{7}, data1, data2, data3);

fclose(fid);
```

```matlab
function [S_F16_MODEL, MAC_F16_MODEL, ...
          S_787_MODEL, MAC_787_MODEL, ...
          W_F16_MODEL, W_787_MODEL, ...
          DIST_CG_F16_CLEAN, ...
          DIST_CG_F16_DIRTY, ...
          DIST_CG_787, SCALE_F16, SCALE_787] = Define_Model_Geometry

    %% [S_F16_MODEL, MAC_F16_MODEL, ...
    %%  S_787_MODEL, MAC_787_MODEL, ...
    %%  W_F16_MODEL, W_787_MODEL, ...
    %%  DIST_CG_F16_CLEAN, ...
    %%  DIST_CG_F16_DIRTY, ...
    %%  DIST_CG_787, SCALE_F16, SCALE_787] = Define_Model_Geometry
    %%
    %% Defines the geometry of the scale models based on the geometry of
    %% the full-size airplane, and the scale factor between the model and
    %% the full-size airplane.
    %%
    %% Returns the neccessary geometry parameters needed to compute C_L,
    %% C_D, and C_M for the scale models in SI units.
    %%
    %%
    %% Author: Nicholas Renninger
    %% Date Created: 2/5/17
    %% Last Modified: 3/6/17

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % define an uncertainty multiplier in the estimated MAC and S of models
    UNCERTAINTY_K = 1e-4;
```

```matlab


% this is the scaling from the real plane to the model
% e.g. the F-16 model is 1/48 scale, 787 is 1/225
SCALE_F16 = 1/48;
SCALE_787 = 1/225;

%% Find Weight of the Model
W_F16 = 30000 * 4.448221628254617; % convert to [N]
W_787 = 360000 * 4.448221628254617; %% convert to [N]

% Scale weight of model
W_F16_MODEL = W_F16 * SCALE_F16^3;
W_787_MODEL = W_787 * SCALE_787^3;


%% Real Plane Geometry

% Wing surface area of the actual plane
S_F16 = 27.87; % [m^2]
S_787 = 325.2897752; % [m^2]

% Define Taper Ratio and Root Chord Length of actual F-16 Wing
t_ratio_F16 = 0.21;
C_root_F_16 = 5.04; % [m]

% Wing Mean Aerodynamic Chord (M.A.C.) Length of actual plane
MAC_F16 = (2/3) * C_root_F_16 * ( (1 + t_ratio_F16 + t_ratio_F16^2) ...
                                / (1 + t_ratio_F16) ); % [m]
MAC_787 = 6.437376; % [m], found online source


%% Model Plane Geometry

% Wing surface area of the model planes. Square scale factor, as scale
% factor is based on length ratio, area is essentially a length^2 ratio
S_F16_MODEL.data = SCALE_F16^2 * S_F16; % [m^2]
S_787_MODEL.data = SCALE_787^2 * S_787; % [m^2]

% define uncertainty in this measurement
S_F16_MODEL.error = S_F16_MODEL.data * UNCERTAINTY_K; % [m^2]
S_787_MODEL.error = S_787_MODEL.data * UNCERTAINTY_K; % [m^2]




% M.A.C. of the model planes. Use unadultered scale factor, as M.A.C.
% is a function of the lenggth of root chord and taper ratio for the
% trapezoidal wing planforms of the F-16 and the 787.
MAC_F16_MODEL.data = SCALE_F16 * MAC_F16; % [m]
MAC_787_MODEL.data = SCALE_787 * MAC_787; % [m]

% define uncertainty in this measurement
MAC_F16_MODEL.error = MAC_F16_MODEL.data * UNCERTAINTY_K; % [m^2]
MAC_787_MODEL.error = MAC_787_MODEL.data * UNCERTAINTY_K; % [m^2]
```

```
    % defining distance from sting balance CG to CG of model
    DIST_CG_F16_CLEAN.data = 14.4 / 1000; % [m]
    DIST_CG_F16_DIRTY.data = 15.5 / 1000; % [m]
    DIST_CG_787.data = 63.0 / 1000; % [m]

    % define uncertainty in this measurement
    DIST_CG_F16_CLEAN.error = UNCERTAINTY_K; % [m]
    DIST_CG_F16_DIRTY.error = UNCERTAINTY_K; % [m]
    DIST_CG_787.error = UNCERTAINTY_K; % [m]


end
```

Input Reading

```
function [velocity_test_matrix] = ASEN_2004_Lab_1_readInput()

    %%% function [velocity_test_matrix] = ASEN_2004_Lab_1_readInput()
    %%%
    %%% Inputs: nothing, but needs the directory structure from the .m
    %%% files to be set like this:
    %%%
    %%% ..\Data\All_Group_Data\VelocityVoltage\S_01x\
    %%% ..\Data\All_Group_Data\BoundryLayer\S_01x\
    %%%
    %%% In these folders, the data must be saved as a .csv, and organized
    %%% such that each folder contains only data from the correct section.
    %%% The data files must be organized alphabetically, and the data taken
    %%% must be taken in such a way that it follows the ASEN 2004 Lab 1
    %%% test matrix.
    %%%
    %%% Outputs: cell array containing filtered expiremental data for each
    %%% model. Each cell array contains two structures containing the data
    %%% and variance for the two test velocities. There are seven columns
    %%% for each velocity tested, per model. They columns are ordered as
    %%% follows:
    %%%
    %%% 1: Atmospheric Density [kg/m^3]
    %%% 2: Airspeed [m/s]
    %%% 3: Pitot Dynamic Pressure [N/m^2] -or- [Pa]
    %%% 4: Angle of Attack (AoA) [degrees]
    %%% 5: Sting Normal [N]
    %%% 6: Sting Axial Force [N]
    %%% 7: Sting Pitching Moment [Nm]
    %%%
    %%%
    %%% Author: Nicholas Renninger
    %%% Last Modified: 2/18/17
    %%% ASEN 2004 Lab #1

    %%% Reads all data files from set directory structure, sorts them, and
    %%% amalgamates totalized data matricies containing all test data ready
    %%% for analysis.
```

```
%% Command Window I/O
disp('Reading in data files...')

%% Set File I/O constants

% Set relevant columns of data to extract from data files
dataCols = [3:5, 23:26];

% sort data by AoA (3rd column)
AOA_col = 4;


%% set path of velocity data files

% section 011
path_Velocity{1} = '../Data/Test Data/S_011/';
vel_folder_path{1} = cat(2, path_Velocity{1}, '*.csv');

% section 012
path_Velocity{2} = '../Data/Test Data/S_012/';
vel_folder_path{2} = cat(2, path_Velocity{2}, '*.csv');

% section 013
path_Velocity{3} = '../Data/Test Data/S_013/';
vel_folder_path{3} = cat(2, path_Velocity{3}, '*.csv');


%% Get Dir. for each set of data

% section 011
dir_test{1} = dir(vel_folder_path{1});

% section 012
dir_test{2} = dir(vel_folder_path{2});

% section 013
dir_test{3} = dir(vel_folder_path{3});

%% Reading Velocity Data out of Spreadsheets and into Cell Arrays

% intialize beginning indices of saving processes
index_clean = 1;
index_dirty = 1;
index_787 = 1;

% initialize all cell arrays
[data_clean, data_dirty, ...
 data_787, error_clean, ...
 error_dirty, error_787] = deal(cell(1, 2));

% initialize regex expression
cleanExpression = 'clean(\w*)';
dirtyExpression = 'loaded(\w*)';
seven87Expression = '787(\W*)';
```

```
    % read in velocity data from data files into cell arrays corresponding
    % to the type of measurements taken by the pressure transducer. Each
    % Cell entry contains the contents of an entire group's data.
    for j = 1:length(dir_test)

        for k = 1:length(dir_test{j})

            current_directory = dir_test{j};
            filename = cat(2, path_Velocity{j}, current_directory(k, 1).name);
            current_spreadsheet = load(filename);

            % average data for each AoA measured
            % 500 measurements per
            [current_spreadsheet,...
             curr_STD] = average_data_points(current_spreadsheet, 20);

            [num_data_points, ~] = size(current_spreadsheet);

            % separating data into zero and 25 m/s data, and their errors
            current_zero_data = current_spreadsheet(1:num_data_points/2,...
                                                    dataCols);
            current_zero_error_data = curr_STD(1:num_data_points/2,...
                                                    dataCols);


            current_25_data = current_spreadsheet(num_data_points/2 + 1:...
                                                num_data_points, ...
                                                dataCols);
            current_25_error_data = curr_STD(num_data_points/2 + 1:...
                                                num_data_points, ...
                                                dataCols);



            % determine which model is being tested: the F-16 clean, F-16
            % Dirty, or the 787 model by matching a regex with the filename

            if ~isempty( regexpi(filename, cleanExpression) ) % clean

                % 0 m/s goes in 1st col, 25 m/s goes in 2nd col
                data_clean{index_clean, 1} = current_zero_data;
                data_clean{index_clean, 2} = current_25_data;

                error_clean{index_clean, 1} = current_zero_error_data;
                error_clean{index_clean, 2} = current_25_error_data;

                index_clean = index_clean + 1;
                %disp('clean')

            elseif ~isempty( regexpi(filename, dirtyExpression) ) % dirty

                % 0 m/s goes in 1st col, 25 m/s goes in 2nd col
                data_dirty{index_dirty, 1} = current_zero_data;
                data_dirty{index_dirty, 2} = current_25_data;

```

```
                error_dirty{index_dirty, 1} = current_zero_error_data;
                error_dirty{index_dirty, 2} = current_25_error_data;

                index_dirty = index_dirty + 1;
                %disp('dirty')

            elseif ~isempty( regexpi(filename, seven87Expression) ) % 787

                % 0 m/s goes in 1st col, 25 m/s goes in 2nd col
                data_787{index_787, 1} = current_zero_data;
                data_787{index_787, 2} = current_25_data;

                error_787{index_787, 1} = current_zero_error_data;
                error_787{index_787, 2} = current_25_error_data;

                index_787 = index_787 + 1;
                %disp('787')

            end

        end

    end


    %%% sort rows based on voltage to build array of test data vs. increasing
    %%% Voltages


    %%% build up test matrix of velocity data %%%

    % initialize matrix with first sets of test data and error
    clean_test_matrix.zero.data = data_clean{1, 1};
    clean_test_matrix.twentyFive.data = data_clean{1, 2};

    clean_test_matrix.zero.error = error_clean{1, 1};
    clean_test_matrix.twentyFive.error = error_clean{1, 1};

    dirty_test_matrix.zero.data = data_dirty{1, 1};
    dirty_test_matrix.twentyFive.data = data_dirty{1, 2};

    dirty_test_matrix.zero.error = error_dirty{1, 1};
    dirty_test_matrix.twentyFive.error = error_dirty{1, 2};

    seven87_test_matrix.zero.data = data_787{1, 1};
    seven87_test_matrix.twentyFive.data = data_787{1, 2};

    seven87_test_matrix.zero.error = error_787{1, 1};
    seven87_test_matrix.twentyFive.error = error_787{1, 2};


    % build up rest of data
    for k = 2:length(data_clean)

        clean_test_matrix.zero.data = cat(1, clean_test_matrix.zero.data, ...
```

```
                                        data_clean{k, 1});
        clean_test_matrix.twentyFive.data = cat(1, ...
                                                clean_test_matrix.twentyFive.data,
                                                    ...
                                                data_clean{k, 2});

        clean_test_matrix.zero.error = cat(1, ...
                                              clean_test_matrix.zero.error, ...
                                              error_clean{k, 1});
        clean_test_matrix.twentyFive.error = cat(1, ...
                                      clean_test_matrix.twentyFive.error, ...
                                      error_clean{k, 2});


    end

    % build up rest of data
    for k = 2:length(data_dirty)

        dirty_test_matrix.zero.data = cat(1, dirty_test_matrix.zero.data, ...
                                        data_dirty{k, 1});
        dirty_test_matrix.twentyFive.data = cat(1, ...
                                                dirty_test_matrix.twentyFive.data,
                                                    ...
                                                data_dirty{k, 2});

        dirty_test_matrix.zero.error = cat(1, ...
                                              dirty_test_matrix.zero.error, ...
                                              error_dirty{k, 1});
        dirty_test_matrix.twentyFive.error = cat(1, ...
                                      dirty_test_matrix.twentyFive.error, ...
                                      error_dirty{k, 2});

    end

    % build up rest of data
    for k = 2:length(data_787)

        seven87_test_matrix.zero.data = cat(1, seven87_test_matrix.zero.data,
            ...
                                        data_787{k, 1});
        seven87_test_matrix.twentyFive.data = cat(1, ...
                                              seven87_test_matrix.twentyFive.data,
                                                  ...
                                                data_787{k, 2});


        seven87_test_matrix.zero.error = cat(1, ...
                                              seven87_test_matrix.zero.error, ...
                                              error_787{k, 1});
        seven87_test_matrix.twentyFive.error = cat(1, ...
                                      seven87_test_matrix.twentyFive.error, ...
                                      error_787{k, 2});

    end
```

```
    [clean_test_matrix.zero.data, idx] = sortrows(clean_test_matrix.zero.data, AOA_col);
    clean_test_matrix.zero.error = clean_test_matrix.zero.error(idx, :);

    [clean_test_matrix.twentyFive.data, idx] = sortrows(clean_test_matrix.twentyFive.data, AOA_col);
    clean_test_matrix.twentyFive.error = clean_test_matrix.twentyFive.error(idx, :);


    [dirty_test_matrix.zero.data, idx] = sortrows(dirty_test_matrix.zero.data, AOA_col);
    dirty_test_matrix.zero.error = dirty_test_matrix.zero.error(idx, :);

    [dirty_test_matrix.twentyFive.data, idx] = sortrows(dirty_test_matrix.twentyFive.data, AOA_col);
    dirty_test_matrix.twentyFive.error = dirty_test_matrix.twentyFive.error(idx, :);

    [seven87_test_matrix.zero.data, idx] = sortrows(seven87_test_matrix.zero.data, AOA_col);
    seven87_test_matrix.zero.error = seven87_test_matrix.zero.error(idx, :);

    [seven87_test_matrix.twentyFive.data, idx] = sortrows(seven87_test_matrix.twentyFive.data, AOA_col);
    seven87_test_matrix.twentyFive.error = seven87_test_matrix.twentyFive.error(idx, :);


    % filter out duplicate data points
    [clean_test_matrix.zero.data, ...
     clean_test_matrix.zero.error] = filterAndAverage(clean_test_matrix.zero.data, ...
                                                      clean_test_matrix.zero.error, ...
                                                      AOA_col);
    [clean_test_matrix.twentyFive.data, ...
     clean_test_matrix.twentyFive.error] = filterAndAverage(clean_test_matrix.twentyFive.data, ...
                                                      clean_test_matrix.twentyFive.error, ...
                                                      AOA_col);



    [dirty_test_matrix.zero.data, ...
     dirty_test_matrix.zero.error] = filterAndAverage(dirty_test_matrix.zero.data, ...
                                                      dirty_test_matrix.zero.error, ...
                                                      AOA_col);
    [dirty_test_matrix.twentyFive.data, ...
     dirty_test_matrix.twentyFive.error] = filterAndAverage(dirty_test_matrix.
```

```
            twentyFive.data, ...
                                                    dirty_test_matrix.
                                                        twentyFive.error, ...
                                                    AOA_col);


    [seven87_test_matrix.zero.data, ...
     seven87_test_matrix.zero.error] = filterAndAverage(seven87_test_matrix.
         zero.data, ...
                                                    seven87_test_matrix.zero
                                                        .error, ...
                                                    AOA_col);
    [seven87_test_matrix.twentyFive.data, ...
     seven87_test_matrix.twentyFive.error] = filterAndAverage(
         seven87_test_matrix.twentyFive.data, ...
                                                    seven87_test_matrix.
                                                        twentyFive.error, ...
                                                    AOA_col);


    velocity_test_matrix = {clean_test_matrix,...
                            dirty_test_matrix, ...
                            seven87_test_matrix};
    %% Command Window I/O
    disp('Successfully read in data.')

end
```

```
function [averaged_data, STD_mat] = average_data_points(current_data, ...
                                        numDataPointsPerMeasurement)

    % average data for each variable measured
    % numDataPointsPerMeasurement measurement per measurement
    AoA_indices = linspace(1, length(current_data) + 1, ...
                            length(current_data) / ...
                            numDataPointsPerMeasurement + 1);

    % for every voltage measured, average numDataPointsPerMeasurement
    % values together
    for k = 1:length(AoA_indices) - 1

        current_V_index_range = AoA_indices(k):AoA_indices(k + 1) - 1;
        averaged_data(k, :) = mean(current_data(current_V_index_range, :), 1);
        STD_mat(k, :) = std(current_data(current_V_index_range, :));

    end

end
```

```
function [filtered_data, filtered_error] = filterAndAverage(input_mat,
    error_mat, AOA_index)

    %%% Filter out and average any duplicate data points

    avg_index = 1;
    new_airspeed_index = 1;
```

```
    % pull AoA out of matrix
    AoA = input_mat(:, AOA_index);

    for i = 1:length(input_mat)

        if i ~= 1

            current_AoA = round(AoA(i));
            prev_AoA = round(AoA(i - 1));

            if current_AoA == prev_AoA

                similar_mat(avg_index, :) = input_mat(i, :);
                sim_error_mat(avg_index, :) = error_mat(i, :);
                avg_index = avg_index + 1;

            else % new AoA

                [r, ~] = size(similar_mat);

                if r == 1
                    new_data_matrix(new_airspeed_index, :) = similar_mat;
                    new_error_matrix(new_airspeed_index, :) = sim_error_mat;
                else
                    [weightedMean, errorMean] = weightedMean_and_Variance(similar_mat, sim_error_mat);
                    new_data_matrix(new_airspeed_index, :) = weightedMean;
                    new_error_matrix(new_airspeed_index, :) = errorMean;
                end

                new_airspeed_index = new_airspeed_index + 1;
                avg_index = 1;
                similar_mat = [];
                sim_error_mat = [];
                similar_mat(avg_index, :) = input_mat(i, :);
                sim_error_mat(avg_index, :) = error_mat(i, :);
                avg_index = avg_index + 1;
            end

        elseif i == 1 % first iteration

            similar_mat(avg_index, :) = input_mat(i, :);
            sim_error_mat(avg_index, :) = error_mat(i, :);
            avg_index = avg_index + 1;

        end

    end

    if new_data_matrix(avg_index - 1, end) ~= current_AoA % if have not added last AoA into matrix
        [weightedMean, errorMean] = weightedMean_and_Variance(similar_mat, sim_error_mat);
        new_data_matrix(new_airspeed_index, :) = weightedMean;
```

```
            new_error_matrix(new_airspeed_index, :) = errorMean;
        end

        filtered_data = new_data_matrix;
        filtered_error = new_error_matrix;

end
```

```
function [weightedMean, errorMean] = weightedMean_and_Variance(data, error)

    % calculate weights based on statistical variance of data (error mat.)
    weights = 1 ./ error.^2;

    % check if any of the weights is zero and fix it
    if ~isempty(find(weights == inf, 1))

        [r, c] = find(weights == inf);

        for i = 1:length(r)

            % re make weight based on super small error for zero error
            non_zero_errors = error(error(:, c(i)) ~= 0, c(i));
            if isempty(non_zero_errors)
                new_error = data(r(i), c(i)) * 1e-3;
                weights(r(i), c(i)) = 1 / new_error^2;
            else
                new_error = min(non_zero_errors(:, 1)) * 1e-3;
                weights(r(i), c(i)) = 1 / new_error^2;
            end

        end
    end

    %% calculate sums for the weighted mean of the matrix
    [numPoints, numVars] = size(data);

    [sumOfAvgPoints, sumOfWeights] = deal(zeros(1, numVars));

    for i = 1:numVars
        for j = 1:numPoints

            sumOfAvgPoints(i) = sumOfAvgPoints(i) + weights(j, i) .* data(j, i
                );
            sumOfWeights(i) = sumOfWeights(i) + weights(j, i);
        end
    end

    weightedMean = sumOfAvgPoints ./ sumOfWeights;



    %% calculate the variance of the weighted mean

    errorMean = sqrt(1 ./ sumOfWeights);


```

```
end
```

Finding C_L, C_D, and C_M

```
function all_data = LiftDragMoment(all_data, Dist_CG, S, MAC)

    %% all_data = LiftDragMoment(all_data, Dist_CG, S, MAC)
    %%
    %% Function Computes the coefficients of lift, drag, and pitching
    %% moment for every angle of attack measured, and
    %%
    %%
    %% Inputs:
    %%
    %%          alldata: struct containing expiremental data and variance
    %%                   for each data point. From one model, it contain
    %%                   expiremental data and the statistical variance of
    %%                   each point of data, tested at both zero (airspeed
    %%                   = 0 m/s) and twentyFive (airspeed = 25 m/s). Each
    %%                   of these matrices contains the filtered data from
    %%                   the tests as a matrix in which the columns are:
    %%
    %%                   1: Atmospheric Density, rho [kg/m^3]
    %%                   2: Airspeed, V_infinity [m/s]
    %%                   3: Pitot Dynamic Pressure, q [N/m^2] -or- [Pa]
    %%                   4: Angle of Attack (AoA), AoA [degrees]
    %%                   5: Sting Normal, N [N]
    %%                   6: Sting Axial Force, A [N]
    %%                   7: Sting Pitching Moment, M [Nm]
    %%
    %%
    %%                   Ex. usage: "alldata.zero.data" and
    %%                   "alldata.zero.error" to get both the expiremental
    %%                   data and the corresponding uncertainty of each
    %%                   measurement for the current model tested at 0
    %%                   m/s.
    %%
    %%         Dist_CG: dist. from balance center to CG & error [m]
    %%
    %%         S: wing planform area of scale model & error [m^2]
    %%
    %%         MAC: mean aerodynamic wing chord of scale model & error [m]
    %%
    %%
    %% Outputs:
    %%
    %%         Update all_data with the following updated fields:
    %%
    %%         AoA: struct containing each AoA tested and the uncertainty
    %%              in the measurement of each AoA.
    %%
    %%         C_L: struct containing the value of C_L computed at every
    %%               AoA, with
    %%               the uncertainty of each value of C_L computed.
    %%               Uncertainty is computed using the general error
    %%               propagation formula, with each intermediary value
```

```
%%                    having its uncertainty computed, and then combined
%%                    again using the general formula to find the
%%                    uncertainty in C_L.
%%
%%              C_D: struct containing the value of C_D computed at every
%%                   AoA, with
%%                    the uncertainty of each value of C_D computed.
%%                    Uncertainty is computed using the general error
%%                    propagation formula, with each intermediary value
%%                    having its uncertainty computed, and then combined
%%                    again using the general formula to find the
%%                    uncertainty in C_D.
%%
%%              C_M: struct containing the value of C_M computed at every
%%                   AoA, with
%%                    the uncertainty of each value of C_M computed.
%%                    Uncertainty is computed using the general error
%%                    propagation formula, with each intermediary value
%%                    having its uncertainty computed, and then combined
%%                    again using the general formula to find the
%%                    uncertainty in C_M. Computed using the Moment measured
%%                    about the model's CG, then C_M is computed for the
%%                   M.A.C.
%%
%%
%%
%% Author: Nicholas Renninger
%% Created: 2/8/17
%% Last modified: 3/1/17



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% pulling out data for analysis
zero.data = all_data.zero.data;
zero.error = all_data.zero.error;
twentyFive.data = all_data.twentyFive.data;
twentyFive.error = all_data.twentyFive.error;

%% pull out each column of data from constituent matrix

% q [Pa]
data_col = 3;

q.data = twentyFive.data(:, data_col);
q.error = twentyFive.error(:, data_col);

% AoA [deg]
data_col = 4;

AoA.data = twentyFive.data(:, data_col);
AoA.error = twentyFive.error(:, data_col);
all_data.AoA = AoA; % save AoA data to struct
```

```matlab
    % N [N]
    data_col = 5;

    N.data = twentyFive.data(:, data_col) - zero.data(:, data_col);
    N.error = sqrt( twentyFive.error(:, data_col).^2 + ...
                    zero.error(:, data_col).^2 ); % use gen.error formula

    % A [N]
    data_col = 6;

    A.data = twentyFive.data(:, data_col) - zero.data(:, data_col);
    A.error = sqrt( twentyFive.error(:, data_col).^2 + ...
                    zero.error(:, data_col).^2 ); % use gen.error formula

    % M [Nm]
    data_col = 7;

    M.data = twentyFive.data(:, data_col) - zero.data(:, data_col);
    M.error = sqrt( twentyFive.error(:, data_col).^2 + ...
                    zero.error(:, data_col).^2 ); % use gen.error formula



    %% Analyze Data - Calc L, D, M, then C_L, C_D, C_M

    % declare sybolic variables for the calculation of the uncertainty in
    % C_L, C_D, and C_M
    syms N_sym A_sym M_sym AoA_sym Dist_CG_sym L_sym ...
         q_sym S_sym D_sym C_sym M_CG_sym MAC_sym


    % 1) Calc L = N*cos(AoA) - A*sin(AoA)
    % 2) Calc D = N*sin(AoA) + A*cos(AoA)
    % 3) Calc C_L = L / (q * S)
    % 4) Calc C_D = D / (q * S)
    % 5) Calc P_moment = M - (N*Dist_CGref)
    % 6) Calc C_M = P_moment / (q * S * C)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Lift [N]
    L_fcn = @(N_sym, A_sym, AoA_sym) N_sym .* cos( AoA_sym * (pi/180) ) ...
                                    - A_sym .* sin( AoA_sym * (pi/180) );

    L.data = L_fcn(N.data, A.data, AoA.data);

    % find error in L using general error propagation
    for i = 1:length(N.data)


        L.error(i) = ErrorProp(L_fcn, [N_sym, A_sym, AoA_sym], ...
                               [N.data(i), A.data(i), AoA.data(i)], ...
                               [N.error(i), A.error(i), AoA.error(i)]);

    end

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Drag [N]
D_fcn = @(N_sym, A_sym, AoA_sym) N_sym .* sin( AoA_sym * (pi/180) ) ...
                                 + A_sym .* cos( AoA_sym * (pi/180) );

D.data = D_fcn(N.data, A.data, AoA.data);

% find error in D using general error propagation
for i = 1:length(N.data)


    D.error(i) = ErrorProp(D_fcn, [N_sym, A_sym, AoA_sym], ...
                           [N.data(i), A.data(i), AoA.data(i)], ...
                           [N.error(i), A.error(i), AoA.error(i)]);

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Moment about CG [Nm]
M_CG_fcn = @(M_sym, N_sym, Dist_CG_sym) M_sym - ...
                                       (N_sym .* Dist_CG_sym);

M_CG.data = M_CG_fcn(M.data, N.data, Dist_CG.data);

% find error in M about CG using general error propagation
for i = 1:length(N.data)


    M_CG.error(i) = ErrorProp(M_CG_fcn, [M_sym, N_sym, Dist_CG_sym],...
                           [M.data(i), N.data(i), Dist_CG.data], ...
                           [M.error(i), N.error(i), Dist_CG.error]);

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% C_L
C_L_fcn = @(L_sym, q_sym, S_sym) L_sym ./ (q_sym .* S_sym);

all_data.C_L.data = C_L_fcn(L.data, q.data, S.data);

% find error in M about CG using general error propagation
for i = 1:length(L.data)


    all_data.C_L.error(i) = ErrorProp(C_L_fcn, [L_sym, q_sym, S_sym], ...
                           [L.data(i), q.data(i), S.data], ...
                           [L.error(i), q.error(i), S.error]);

end
```

```matlab



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% C_D
    C_D_fcn = @(D_sym, q_sym, S_sym) D_sym ./ (q_sym .* S_sym);

    all_data.C_D.data = C_D_fcn(D.data, q.data, S.data);

    % find error in M about CG using general error propagation
    for i = 1:length(D.data)


        all_data.C_D.error(i) = ErrorProp(C_D_fcn, [D_sym, q_sym, S_sym], ...
                                [D.data(i), q.data(i), S.data], ...
                                [D.error(i), q.error(i), S.error]);

    end



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% C_M
    C_M_fcn = @(M_CG_sym, q_sym, S_sym, MAC_sym) M_CG_sym ./ ...
                                    (q_sym .* S_sym .* MAC_sym);

    all_data.C_M.data = C_M_fcn(M_CG.data, q.data, S.data, MAC.data);

    % find error in M about CG using general error propagation
    for i = 1:length(M_CG.data)


        all_data.C_M.error(i) = ErrorProp(C_M_fcn, ...
                        [M_CG_sym, q_sym, S_sym, MAC_sym], ...
                        [M_CG.data(i), q.data(i), S.data, MAC.data], ...
                        [M_CG.error(i), q.error(i), S.error, MAC.error]);

    end



end
```

```matlab
function totError = ErrorProp(func, dependents, vals, dependentError)

    %% totError = ErrorProp(func, dependents, vals, dependentError)
    %%
    %% Uses a numeric formulation of the general error propogation formula
    %% to calculate the total uncertainty of calculated value based on
    %% the uncertainty in the measurements used to compute the value.
    %%
    %% Ex. function call:
    %%
    %%     syms ho hn
    %%     e = @(ho,hn) ((hn/ho).^(1/2));
    %%     sigma_ho = 0.1;
```

```
    %%      sigma_hn = 0.2;
    %%      sigma_e = ErrorProp(e,[hn ho], [10, 8], [sigma_ho, sigma_hn]);
    %%
    %% Inputs:
    %%        - func: inline function that maps the measures values to the
    %%                calculated value.
    %%
    %%        - dependents: symbolic arguments to func, i.e. the
    %%                      measurement variables that might have error in
    %%                      them.
    %%
    %%        - vals: values of the dependent variables when used to
    %%                calculate the independen variable.
    %%
    %%        - dependentError: the uncertainty in the measured values of
    %%                          the dependent variables.
    %%
    %% Author: Jeffrey Mariner Gonzalez & Nicholas Renninger
    %% Date Created: 2/5/17
    %% Last Modified: 2/15/17
    %%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    x = numel(dependents);
    temp = vpa(ones(1,x));

    % Compute Partials
    for i = 1:x
       temp(i) = diff(func,dependents(i),1);
    end

    % Sum in quadrature to compute general error
    totError = sqrt( ( sum( (subs(temp,dependents,vals).^2) .* ...
                            (dependentError.^2) ) ) );
    totError = double(totError);

end
```

Finding Stability Derivatives and L/D

```
function struct_out = auxilaryCalculations(struct_in, ...
                                           S_model, W_model_theory, ...
                                           SCALE, isMilitary)

    %% struct_out = auxilaryCalculations(struct_in, ...
    %%                                   S_model, W_theory, SCALE, ...
    %%                                   isMilitary)
    %%
    %% Function calculates the
    %%
    %% Inputs:
    %%
    %%        "struct_in" Struct Containing:
    %%
    %%           Two structs containing data at 25 and 0 m/s with the
```

```
%%%          following data:
%%%
%%%                  1: Atmospheric Density, rho [kg/m^3]
%%%                  2: Airspeed, V_infinity [m/s]
%%%                  3: Pitot Dynamic Pressure, q [N/m^2] -or- [Pa]
%%%                  4: Angle of Attack (AoA), AoA [degrees]
%%%                  5: Sting Normal, N [N]
%%%                  6: Sting Axial Force, A [N]
%%%                  7: Sting Pitching Moment, M [Nm]
%%%
%%%
%%%                  Ex. usage: "alldata.zero.data" and
%%%                  "alldata.zero.error" to get both the expiremental
%%%                  data and the corresponding uncertainty of each
%%%                  measurement for the current model tested at 0
%%%                  m/s.
%%%
%%%          AoA: struct containing each AoA tested and the uncertainty
%%%             in the measurement of each AoA.
%%%
%%%          C_L: struct containing the value of C_L computed at every
%%%              AoA, with
%%%              the uncertainty of each value of C_L computed.
%%%              Uncertainty is computed using the general error
%%%              propagation formula, with each intermediary value
%%%              having its uncertainty computed, and then combined
%%%              again using the general formula to find the
%%%              uncertainty in C_L.
%%%
%%%          C_D: struct containing the value of C_D computed at every
%%%              AoA, with
%%%              the uncertainty of each value of C_D computed.
%%%              Uncertainty is computed using the general error
%%%              propagation formula, with each intermediary value
%%%              having its uncertainty computed, and then combined
%%%              again using the general formula to find the
%%%              uncertainty in C_D.
%%%
%%%          C_M: struct containing the value of C_M computed at every
%%%              AoA, with
%%%              the uncertainty of each value of C_M computed.
%%%              Uncertainty is computed using the general error
%%%              propagation formula, with each intermediary value
%%%              having its uncertainty computed, and then combined
%%%              again using the general formula to find the
%%%              uncertainty in C_M. Computed using the Moment measured
%%%              about the model's CG, then C_M is computed for the
%%%              M.A.C.
%%%
%%%      - S_model: the wing area of the model in [m^2]
%%%
%%%      - W_model_theory: this is the weight of the model if it were
%%%                  correctly mass modeled. This is calculated using
%%%                  the following: W_F16_MODEL = W_F16 * SCALE_F16^3
%%%
```

```
%%%       - SCALE: length Scale factor between the model and the
%%%                real plane.
%%%
%%%       - isMilitary: bool that changes analysis based on military
%%%                     or civilian reqs.
%%%
%%% Outputs:
%%%
%%%         Update struct_in to struct_out with the
%%%         following updated fields:
%%%
%%%         - V_land_min_real_model: calclated minimum landing
%%%                                  speed for the model plane
%%%                                  assuming its weight is equal
%%%                                  to the maximum lift it
%%%                                  experienced during the test.
%%%
%%%         - V_land_min_theoretical_model: calclated minimum landing
%%%                                  speed for the model plane
%%%                                  assuming its weight is
%%%                                  accurately scaled down from
%%%                                  the real plane (i.e the model
%%%                                  is a mass model).
%%%
%%%         - V_land_realPlane: calclated minimum landing
%%%                             speed for the real-size plane
%%%                             assuming by scaling up
%%%                             V_land_min_real_model using:
%%%
%%%                             V_stall_realPlane = V_stall_model /
%%%                                                 sqrt(SCALE);
%%%
%%%
%%%         - V_land_realPlane_theory: calclated minimum landing
%%%                             speed for the real-size plane
%%%                             assuming by scaling up
%%%                             V_land_min_theoretical_model using:
%%%
%%%                             V_land_realPlane_theory =
%%%                             V_land_min_theoretical_model /
%%%                             sqrt(SCALE);
%%%
%%% Author: Nicholas Renninger
%%% Date Created: 2/18/17
%%% Last Modified: 3/8/17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%% Find Landing Speed of Model

% C_L Max
[CL_max, max_idx] = max(struct_in.C_L.data);

```

```
    % Rho_infinity
    rho_inf = struct_in.twentyFive.data(max_idx, 1);

    % V_infinity
    V_inf = struct_in.twentyFive.data(max_idx, 2);

    % Calculate Weight
    W_real = 0.5 * rho_inf * V_inf^2 * S_model.data * CL_max;

    % Calculate the Stall Speed
    V_stall_model = sqrt( (2 * W_real) / ...
                          (rho_inf * S_model.data * CL_max) );
    V_stall_model_theory = sqrt( (2 * W_model_theory) / ...
                                 (rho_inf * S_model.data * CL_max) );

    V_stall_realPlane = V_stall_model / sqrt(SCALE);
    V_stall_realPlane_theory = V_stall_model_theory / sqrt(SCALE);

    % Calculate Landing Speed of the model

    if isMilitary
        land_K = 1.2;
    else % is commercial/civilian
        land_K = 1.3;
    end

    % all in [m/s]
    struct_in.V_land_min_real_model = land_K * V_stall_model;
    struct_in.V_land_min_theoretical_model = land_K * V_stall_model_theory;
    struct_in.V_land_realPlane = land_K * V_stall_realPlane;
    struct_in.V_land_realPlane_theory = land_K * V_stall_realPlane_theory;


    %% Static Longitudinal Stability (dCM / dAoA)
    CM_AoA_slope = diff(struct_in.CM.data) ./ diff(struct_in.AoA.data);
    zero_idx = find(abs(struct_in.AoA.data) == min(abs(struct_in.AoA.data)));
    struct_in.longit_stability_at_0 = mean([CM_AoA_slope(zero_idx - 1), ...
                                            CM_AoA_slope(zero_idx)]);


    %% Static Margin [- (dCM / dAoA) / (dCL / dAoA)]
    CL_AoA_slope = diff(struct_in.CL.data) ./ diff(struct_in.AoA.data);
    CL_AoA_slope_at_0 = mean([CL_AoA_slope(zero_idx - 1), ...
                                            CL_AoA_slope(zero_idx)]);

    struct_in.dCL_dAoA = CL_AoA_slope_at_0;
    struct_in.SM_at_0 = - (struct_in.longit_stability_at_0 / ...
                          CL_AoA_slope_at_0) * 100; % in [%]

    %% Pass Data Out
    struct_out = struct_in;
```

Plotting

```
function plot_CL_CD_AoA(clean, dirty, seven87, ...
                        FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)
```

```
%%% plot_CL_CD_AoA(clean, dirty, seven87, ...
%%%                 FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)
%%%
%%% Function plots C_L vs. AOA and C_D vs. AoA, with 2 * sigma
%%% error bars.
%%%
%%% Inputs:
%%%
%%%         - clean: struct containing the value of C_L and C_D for
%%%                  every AoA (also contained in this struct) for the
%%%                  clean F-16 model.
%%%
%%%
%%%         - dirty: struct containing the value of C_L and C_D for
%%%                  every AoA (also contained in this struct) for the
%%%                  dirty F-16 model.
%%%
%%%         - seven87: struct containing the value of C_L and C_D for
%%%                  every AoA (also contained in this struct) for the
%%%                  787 model
%%%
%%%         - FONTSIZE: sets the fontsize of the text in the figure
%%%
%%%         - LINEWIDTH: sets the thickness of the lines on the plot
%%%
%%%         - colorVecs: contains RGB triplets that set the line colors
%%%
%%%         - MARKERSIZE: sets the size of the plot value markers
%%%
%%%
%%% Author: Nicholas Renninger
%%% Date Created: 2/18/17
%%% Last Modified: 3/6/17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% C_L constants
x_min = floor(min(clean.AoA.data));
x_max = ceil(max(clean.AoA.data));
x_vec = linspace(x_min, x_max, 100);

y_min_CL_clean = min(clean.C_L.data);
y_min_CL_dirty = min(dirty.C_L.data);
y_min_CL_787 = min(seven87.C_L.data);

min_clean_vec = y_min_CL_clean * ones(1, 100);
min_dirty_vec = y_min_CL_dirty * ones(1, 100);
min_787_vec = y_min_CL_787 * ones(1, 100);

y_max_CL_clean = max(clean.C_L.data);
y_max_CL_dirty = max(dirty.C_L.data);
y_max_CL_787 = max(seven87.C_L.data);

max_clean_vec = y_max_CL_clean * ones(1, 100);
```

```
    max_dirty_vec = y_max_CL_dirty * ones(1, 100);
    max_787_vec = y_max_CL_787 * ones(1, 100);

    y_min = min([y_min_CL_clean, y_min_CL_dirty, y_min_CL_787]) * 1.1;
    y_max = max([y_max_CL_clean, y_max_CL_dirty, y_max_CL_787]) * 1.02;


    sigma_multiplier = 2;

    % Calculate uncertainty in each variable
    cleanErrCL = sigma_multiplier .* clean.C_L.error;
    dirtyErrCL = sigma_multiplier .* dirty.C_L.error;
    seven87ErrCL = sigma_multiplier .* seven87.C_L.error;


    %% Plot C_L
    figure_title = sprintf('$C_L$ vs. $\\alpha$');
    SaveTitleName = 'C_L_vs_AoA';
    saveLocation = '../Figures/';
    saveTitleFull = cat(2, saveLocation, sprintf('%s.pdf', SaveTitleName));
    xlabel_string = sprintf('Angle of Attack, $\\alpha \\, (^\\circ)$');
    ylabel_string = sprintf('$C_L$');
    legend_string = {'Clean Configuration - F-16 ', ...
                     'Dirty Configuration - F-16', ...
                     'Boeing 787 Model'};

    hFig = figure('name', sprintf('C_L vs. Angle of Attack'));
    scrz = get(groot,'ScreenSize');
    set(hFig, 'Position', scrz)

    hold on
    % plot min lines
    plot(x_vec, min_clean_vec, '--', 'color', colorVecs(1, :), 'linewidth',
        LINEWIDTH - 1.5)
    plot(x_vec, min_dirty_vec, '--', 'color', colorVecs(2, :), 'linewidth',
        LINEWIDTH - 1.5)
    plot(x_vec, min_787_vec, '--', 'color', colorVecs(3, :), 'linewidth',
        LINEWIDTH - 1.5)

    % plot max lines
    plot(x_vec, max_clean_vec, '--', 'color', colorVecs(1, :), 'linewidth',
        LINEWIDTH - 1.5)
    plot(x_vec, max_dirty_vec, '--', 'color', colorVecs(2, :), 'linewidth',
        LINEWIDTH - 1.5)
    plot(x_vec, max_787_vec, '--', 'color', colorVecs(3, :), 'linewidth',
        LINEWIDTH - 1.5)

    p1 = errorbar(clean.AoA.data, clean.C_L.data, cleanErrCL, ':^', ...
            'Color', colorVecs(1, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    p2 = errorbar(dirty.AoA.data, dirty.C_L.data, dirtyErrCL, ':d', ...
            'Color', colorVecs(2, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

```

```matlab
    p3 = errorbar(seven87.AoA.data, seven87.C_L.data, seven87ErrCL, ':s', ...
             'Color', colorVecs(3, :), 'LineWidth', LINEWIDTH, ...
             'markersize', MARKERSIZE);

    grid on
    set(gca,'FontSize', FONTSIZE)
    set(gca, 'defaulttextinterpreter', 'latex')
    set(gca, 'TickLabelInterpreter', 'latex')
    xlim([x_min, x_max])
    ylim([y_min, y_max])
    title(figure_title)
    xlabel(xlabel_string)
    ylabel(ylabel_string)
    curr_leg = legend([p1 p2 p3], legend_string, 'location', 'best', ...
                                'interpreter', 'latex');
    set(curr_leg, 'fontsize', round(FONTSIZE * 0.7));

    % setup and save figure as .pdf
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitleFull, 'w+');
    if fid < 1 % check if file is already open.
       error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitleFull);


    %% C_D constants
    x_min = floor(min(clean.AoA.data));
    x_max = ceil(max(clean.AoA.data));
    x_vec = linspace(x_min, x_max, 100);

    y_min_CD_clean = min(clean.C_D.data);
    y_min_CD_dirty = min(dirty.C_D.data);
    y_min_CD_787 = min(seven87.C_D.data);

    min_clean_vec = y_min_CD_clean * ones(1, 100);
    min_dirty_vec = y_min_CD_dirty * ones(1, 100);
    min_787_vec = y_min_CD_787 * ones(1, 100);

    y_max_CD_clean = max(clean.C_D.data);
    y_max_CD_dirty = max(dirty.C_D.data);
    y_max_CD_787 = max(seven87.C_D.data);

    max_clean_vec = y_max_CD_clean * ones(1, 100);
    max_dirty_vec = y_max_CD_dirty * ones(1, 100);
    max_787_vec = y_max_CD_787 * ones(1, 100);

    y_min = min([y_min_CD_clean, y_min_CD_dirty, y_min_CD_787]) * 0.8;
    y_max = max([y_max_CD_clean, y_max_CD_dirty, y_max_CD_787]) * 1.05;

    sigma_multiplier = 2;
```

```
    % Calculate uncertainty in each variable
    cleanErrCD = sigma_multiplier .* clean.C_D.error;
    dirtyErrCD = sigma_multiplier .* dirty.C_D.error;
    seven87ErrCD = sigma_multiplier .* seven87.C_D.error;


    %% Plot C_D
    figure_title = sprintf('$C_D$ vs. $\\alpha$');
    SaveTitleName = 'C_D_vs_AoA';
    saveLocation = '../Figures/';
    saveTitleFull = cat(2, saveLocation, sprintf('%s.pdf', SaveTitleName));
    xlabel_string = sprintf('Angle of Attack, $\\alpha \\, (^\\circ)$');
    ylabel_string = sprintf('$C_D$');
    legend_string = {'Clean Configuration - F-16 ', ...
                     'Dirty Configuration - F-16', ...
                     'Boeing 787 Model'};

    hFig = figure('name', sprintf('C_D vs. Angle of Attack'));
    scrz = get(groot,'ScreenSize');
    set(hFig, 'Position', scrz)

    hold on

    % plot min lines
    plot(x_vec, min_clean_vec, '--', 'color', colorVecs(1, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, min_dirty_vec, '--', 'color', colorVecs(2, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, min_787_vec, '--', 'color', colorVecs(3, :), 'linewidth', LINEWIDTH - 1.5)

    % plot max lines
    plot(x_vec, max_clean_vec, '--', 'color', colorVecs(1, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, max_dirty_vec, '--', 'color', colorVecs(2, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, max_787_vec, '--', 'color', colorVecs(3, :), 'linewidth', LINEWIDTH - 1.5)

    p1 = errorbar(clean.AoA.data, clean.C_D.data, cleanErrCD, ':^', ...
            'Color', colorVecs(1, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    p2 = errorbar(dirty.AoA.data, dirty.C_D.data, dirtyErrCD, ':d', ...
            'Color', colorVecs(2, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    p3 = errorbar(seven87.AoA.data, seven87.C_D.data, seven87ErrCD, ':s', ...
            'Color', colorVecs(3, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    grid on
    set(gca,'FontSize', FONTSIZE)
    set(gca, 'defaulttextinterpreter', 'latex')
```

```
    set(gca, 'TickLabelInterpreter', 'latex')
    xlim([x_min, x_max])
    ylim([y_min, y_max])
    title(figure_title)
    xlabel(xlabel_string)
    ylabel(ylabel_string)
    curr_leg = legend([p1 p2 p3], legend_string, 'location', 'best', ...
                                  'interpreter', 'latex');
    set(curr_leg, 'fontsize', round(FONTSIZE * 0.7));


    % setup and save figure as .pdf
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitleFull, 'w+');
    if fid < 1 % check if file is already open.
       error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitleFull);



end
```

```
function plot_CM_AoA(clean, dirty, seven87, ...
                     FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)

    %%% plot_CM_AoA(AoA, C_M, data_set_string, ...
    %%%                  FONTSIZE, LINEWIDTH, colorVecs, ...
    %%%                  y_limits, MARKERSIZE)
    %%%
    %%% Function plots C_M vs. AoA, with 2 * sigma error bars. Requires
    %%% ../Figures/ folder to exist to save the plots in.
    %%%
    %%% Inputs:
    %%%       - clean: struct containing the value of C_L and C_D for
    %%%                every AoA (also contained in this struct) for the
    %%%                clean F-16 model.
    %%%
    %%%
    %%%       - dirty: struct containing the value of C_L and C_D for
    %%%                every AoA (also contained in this struct) for the
    %%%                dirty F-16 model.
    %%%
    %%%       - seven87: struct containing the value of C_L and C_D for
    %%%                every AoA (also contained in this struct) for the
    %%%                787 model.
    %%%
    %%%       - FONTSIZE: sets the fontsize of the text in the figure
    %%%
    %%%       - LINEWIDTH: sets the thickness of the lines on the plot
    %%%
    %%%       - colorVecs: contains RGB triplets that set the line colors
```

```
%%
%%        - MARKERSIZE: sets the size of the plot value markers
%%
%%
%% Author: Nicholas Renninger
%% Date Created: 2/18/17
%% Last Modified: 3/1/17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Constants
x_min = floor(min(clean.AoA.data));
x_max = ceil(max(clean.AoA.data));

y_min = min(min([clean.C_M.data, dirty.C_M.data, seven87.C_M.data])) * 1.11;
y_max = max(max([clean.C_M.data, dirty.C_M.data, seven87.C_M.data])) * 1.2;

sigma_multiplier = 2;

% Calculate uncertainty in each variable
cleanErrCM = sigma_multiplier .* clean.C_M.error;
dirtyErrCM = sigma_multiplier .* dirty.C_M.error;
seven87ErrCM = sigma_multiplier .* seven87.C_M.error;


%% Plot
figure_title = 'Longitudinal Stability';
SaveTitleName = 'Longitudinal Stability';
saveLocation = '../Figures/';
saveTitleFull = cat(2, saveLocation, sprintf('%s.pdf', SaveTitleName));
xlabel_string = sprintf('Angle of Attack, $\\alpha \\, (^\\circ)$');
ylabel_string = sprintf('$C_M$');
legend_string = {'Clean Configuration - F-16 ', ...
                 'Dirty Configuration - F-16', ...
                 'Boeing 787 Model'};

hFig = figure('name', sprintf('C_M vs. Angle of Attack'));
scrz = get(groot,'ScreenSize');
set(hFig, 'Position', scrz)

errorbar(clean.AoA.data, clean.C_M.data, cleanErrCM, ':^', ...
         'Color', colorVecs(1, :), 'LineWidth', LINEWIDTH, ...
         'markersize', MARKERSIZE);
hold on
errorbar(dirty.AoA.data, dirty.C_M.data, dirtyErrCM, ':d', ...
         'Color', colorVecs(2, :), 'LineWidth', LINEWIDTH, ...
         'markersize', MARKERSIZE);

errorbar(seven87.AoA.data, seven87.C_M.data, seven87ErrCM, ':s', ...
         'Color', colorVecs(3, :), 'LineWidth', LINEWIDTH, ...
         'markersize', MARKERSIZE);
```

```
    grid on
    set(gca,'FontSize', FONTSIZE)
    set(gca, 'defaulttextinterpreter', 'latex')
    set(gca, 'TickLabelInterpreter', 'latex')
    xlim([x_min, x_max])
    ylim([y_min, y_max])
    title(figure_title)
    xlabel(xlabel_string)
    ylabel(ylabel_string)
    curr_leg = legend(legend_string, 'location', 'best', ...
                                     'interpreter', 'latex');
    set(curr_leg, 'fontsize', round(FONTSIZE * 0.7));

    % setup and save figure as .pdf
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitleFull, 'w+');
    if fid < 1 % check if file is already open.
        error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitleFull);



end
```

```
function plot_drag_polar(clean, dirty, seven87, ...
                         FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)

    %%% plot_drag_polar(clean, dirty, seven87, ...
    %%%                 FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)
    %%%
    %%% Function plots C_L vs. C_D (drag polar), with 2 * sigma error bars
    %%%
    %%% Inputs:
    %%%
    %%%        - clean: struct containing the value of C_L and C_D for
    %%%                 every AoA (also contained in this struct) for the
    %%%                 clean F-16 model.
    %%%
    %%%
    %%%        - dirty: struct containing the value of C_L and C_D for
    %%%                 every AoA (also contained in this struct) for the
    %%%                 dirty F-16 model.
    %%%
    %%%        - seven87: struct containing the value of C_L and C_D for
    %%%                 every AoA (also contained in this struct) for the
    %%%                 787 model
    %%%
    %%%        - FONTSIZE: sets the fontsize of the text in the figure
    %%%
    %%%        - LINEWIDTH: sets the thickness of the lines on the plot
    %%%
```

```matlab
%%%         - colorVecs: contains RGB triplets that set the line colors
%%%
%%%         - MARKERSIZE: sets the size of the plot value markers
%%%
%%% Author: Nicholas Renninger
%%% Date Created: 2/18/17
%%% Last Modified: 3/8/17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% C_L, C_D constants
x_min = min(min([clean.C_D.data, dirty.C_D.data, seven87.C_D.data])) * 0.8;
x_max = max(max([clean.C_D.data, dirty.C_D.data, seven87.C_D.data])) * 1.05;

sigma_multiplier = 2;

% Calculate uncertainty in each variable
cleanErrCD = sigma_multiplier .* clean.C_D.error;
dirtyErrCD = sigma_multiplier .* dirty.C_D.error;
seven87ErrCD = sigma_multiplier .* seven87.C_D.error;

y_min = min(min([clean.C_L.data, dirty.C_L.data, seven87.C_L.data])) * 1.1;
y_max = max(max([clean.C_L.data, dirty.C_L.data, seven87.C_L.data])) * 1.02;

sigma_multiplier = 2;

% Calculate uncertainty in each variable
cleanErrCL = sigma_multiplier .* clean.C_L.error;
dirtyErrCL = sigma_multiplier .* dirty.C_L.error;
seven87ErrCL = sigma_multiplier .* seven87.C_L.error;



%% plot
figure_title = sprintf('Drag Polar');
saveLocation = '../Figures/';
saveTitle = cat(2, saveLocation, sprintf('%s.pdf', figure_title));
xlabel_string = sprintf('$C_D$');
ylabel_string = sprintf('$C_L$');
legend_string = {'Clean Configuration - F-16 ', ...
                 'Dirty Configuration - F-16', ...
                 'Boeing 787 Model'};

hFig = figure('name', sprintf('Drag Polar'));
scrz = get(groot,'ScreenSize');
set(hFig, 'Position', scrz)

errorbar(clean.C_D.data, clean.C_L.data, cleanErrCL, ...
         cleanErrCL, cleanErrCD, cleanErrCD, ':^', ...
         'Color', colorVecs(1, :), 'LineWidth', LINEWIDTH, ...
         'markersize', MARKERSIZE);
```

```
    hold on
    errorbar(dirty.C_D.data, dirty.C_L.data, dirtyErrCL, ...
             dirtyErrCL, dirtyErrCD, dirtyErrCD, ':d', ...
             'Color', colorVecs(2, :), 'LineWidth', LINEWIDTH, ...
             'markersize', MARKERSIZE);

    errorbar(seven87.C_D.data, seven87.C_L.data, seven87ErrCL, ...
             seven87ErrCL, seven87ErrCD, seven87ErrCD, ':s', ...
             'Color', colorVecs(3, :), 'LineWidth', LINEWIDTH, ...
             'markersize', MARKERSIZE);


    grid on
    set(gca,'FontSize', FONTSIZE)
    set(gca, 'defaulttextinterpreter', 'latex')
    set(gca, 'TickLabelInterpreter', 'latex')
    xlim([x_min, x_max])
    ylim([y_min, y_max])
    title(figure_title)
    xlabel(xlabel_string)
    ylabel(ylabel_string)
    curr_leg = legend(legend_string, 'location', 'best', ...
                                     'interpreter', 'latex');
    set(curr_leg, 'fontsize', round(FONTSIZE * 0.7));


    % setup and save figure as .pdf
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitle, 'w+');
    if fid < 1 % check if file is already open.
       error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitle);


end
```

```
function plot_L_over_D_AoA(clean, dirty, seven87, ...
                           FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)

    %%% plot_CL_CD_AoA(clean, dirty, seven87, ...
    %%%                FONTSIZE, LINEWIDTH, colorVecs, MARKERSIZE)
    %%%
    %%% Function plots (L/D) vs. AoA, with 2 * sigma
    %%% error bars.
    %%%
    %%% Inputs:
    %%%
    %%%        - clean: struct containing the value of C_L and C_D for
    %%%                 every AoA (also contained in this struct) for the
    %%%                 clean F-16 model.
    %%%
```

```
%%
%%          - dirty: struct containing the value of C_L and C_D for
%%                   every AoA (also contained in this struct) for the
%%                   dirty F-16 model.
%%
%%          - seven87: struct containing the value of C_L and C_D for
%%                   every AoA (also contained in this struct) for the
%%                   787 model
%%
%%          - FONTSIZE: sets the fontsize of the text in the figure
%%
%%          - LINEWIDTH: sets the thickness of the lines on the plot
%%
%%          - colorVecs: contains RGB triplets that set the line colors
%%
%%          - MARKERSIZE: sets the size of the plot value markers
%%
%%
%% Author: Nicholas Renninger
%% Date Created: 2/18/17
%% Last Modified: 3/6/17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Uncertainty Propagation
syms C_L_sym C_D_sym
L_o_D_fcn = @(C_L_sym, C_D_sym) C_L_sym ./ C_D_sym;

L_o_D_clean.data = L_o_D_fcn(clean.C_L.data, clean.C_D.data);
L_o_D_dirty.data = L_o_D_fcn(dirty.C_L.data, dirty.C_D.data);
L_o_D_787.data = L_o_D_fcn(seven87.C_L.data, seven87.C_D.data);

% find error in LoD using general error propagation
for i = 1:length(clean.C_L.data)


    L_o_D_clean.error(i) = ErrorProp(L_o_D_fcn, [C_L_sym, C_D_sym], ...
                           [clean.C_L.data(i), clean.C_D.data(i)], ...
                           [clean.C_L.error(i), clean.C_D.error(i)]);



end

for i = 1:length(dirty.C_L.data)


    L_o_D_dirty.error(i) = ErrorProp(L_o_D_fcn, [C_L_sym, C_D_sym], ...
                           [dirty.C_L.data(i), dirty.C_D.data(i)], ...
                           [dirty.C_L.error(i), dirty.C_D.error(i)]);



end

for i = 1:length(seven87.C_L.data)

```

```
        L_o_D_787.error(i) = ErrorProp(L_o_D_fcn, [C_L_sym, C_D_sym], ...
                            [seven87.C_L.data(i), seven87.C_D.data(i)], ...
                            [seven87.C_L.error(i), seven87.C_D.error(i)]);


    end

    sigma_multiplier = 2;

    % Calculate uncertainty in each variable
    cleanErrLoD = sigma_multiplier .* L_o_D_clean.error;
    dirtyErrLoD = sigma_multiplier .* L_o_D_dirty.error;
    seven87ErrLoD = sigma_multiplier .* L_o_D_787.error;


    %% C_L constants
    x_min = floor(min(clean.AoA.data));
    x_max = ceil(max(clean.AoA.data));
    x_vec = linspace(x_min, x_max, 100);

    y_min_clean = min(L_o_D_clean.data);
    y_min_dirty = min(L_o_D_dirty.data);
    y_min_787 = min(L_o_D_787.data);

    min_clean_vec = y_min_clean * ones(1, 100);
    min_dirty_vec = y_min_dirty * ones(1, 100);
    min_787_vec = y_min_787 * ones(1, 100);

    y_max_clean = max(L_o_D_clean.data);
    y_max_dirty = max(L_o_D_dirty.data);
    y_max_787 = max(L_o_D_787.data);

    max_clean_vec = y_max_clean * ones(1, 100);
    max_dirty_vec = y_max_dirty * ones(1, 100);
    max_787_vec = y_max_787 * ones(1, 100);

    y_min = min([y_min_clean, y_min_dirty, y_min_787]) * 1.1;
    y_max = max([y_max_clean, y_max_dirty, y_max_787]) * 1.101;


    %% Plot Lift over Drag
    figure_title = sprintf('$(L/D)$ vs. $\\alpha$');
    SaveTitleName = 'L_over_D_vs_AoA';
    saveLocation = '../Figures/';
    saveTitleFull = cat(2, saveLocation, sprintf('%s.pdf', SaveTitleName));
    xlabel_string = sprintf('Angle of Attack, $\\alpha \\, (^\\circ)$');
    ylabel_string = sprintf('$(L/D)$');
    legend_string = {'Clean Configuration - F-16 ', ...
                     'Dirty Configuration - F-16', ...
                     'Boeing 787 Model'};

    hFig = figure('name', sprintf('L_over_D vs. Angle of Attack'));
    scrz = get(groot,'ScreenSize');
    set(hFig, 'Position', scrz)
```

```
    hold on
    % plot min lines
    plot(x_vec, min_clean_vec, '--', 'color', colorVecs(1, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, min_dirty_vec, '--', 'color', colorVecs(2, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, min_787_vec, '--', 'color', colorVecs(3, :), 'linewidth', LINEWIDTH - 1.5)

    % plot max lines
    plot(x_vec, max_clean_vec, '--', 'color', colorVecs(1, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, max_dirty_vec, '--', 'color', colorVecs(2, :), 'linewidth', LINEWIDTH - 1.5)
    plot(x_vec, max_787_vec, '--', 'color', colorVecs(3, :), 'linewidth', LINEWIDTH - 1.5)

    p1 = errorbar(clean.AoA.data, L_o_D_clean.data, cleanErrLoD, ':^', ...
            'Color', colorVecs(1, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    p2 = errorbar(dirty.AoA.data, L_o_D_dirty.data, dirtyErrLoD, ':d', ...
            'Color', colorVecs(2, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    p3 = errorbar(seven87.AoA.data, L_o_D_787.data, seven87ErrLoD, ':s', ...
            'Color', colorVecs(3, :), 'LineWidth', LINEWIDTH, ...
            'markersize', MARKERSIZE);

    grid on
    set(gca,'FontSize', FONTSIZE)
    set(gca, 'defaulttextinterpreter', 'latex')
    set(gca, 'TickLabelInterpreter', 'latex')
    xlim([x_min, x_max])
    ylim([y_min, y_max])
    title(figure_title)
    xlabel(xlabel_string)
    ylabel(ylabel_string)
    curr_leg = legend([p1 p2 p3], legend_string, 'location', 'best', ...
                                'interpreter', 'latex');
    set(curr_leg, 'fontsize', round(FONTSIZE * 0.7));

    % setup and save figure as .pdf
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitleFull, 'w+');
    if fid < 1 % check if file is already open.
        error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitleFull);
```

```

end
```