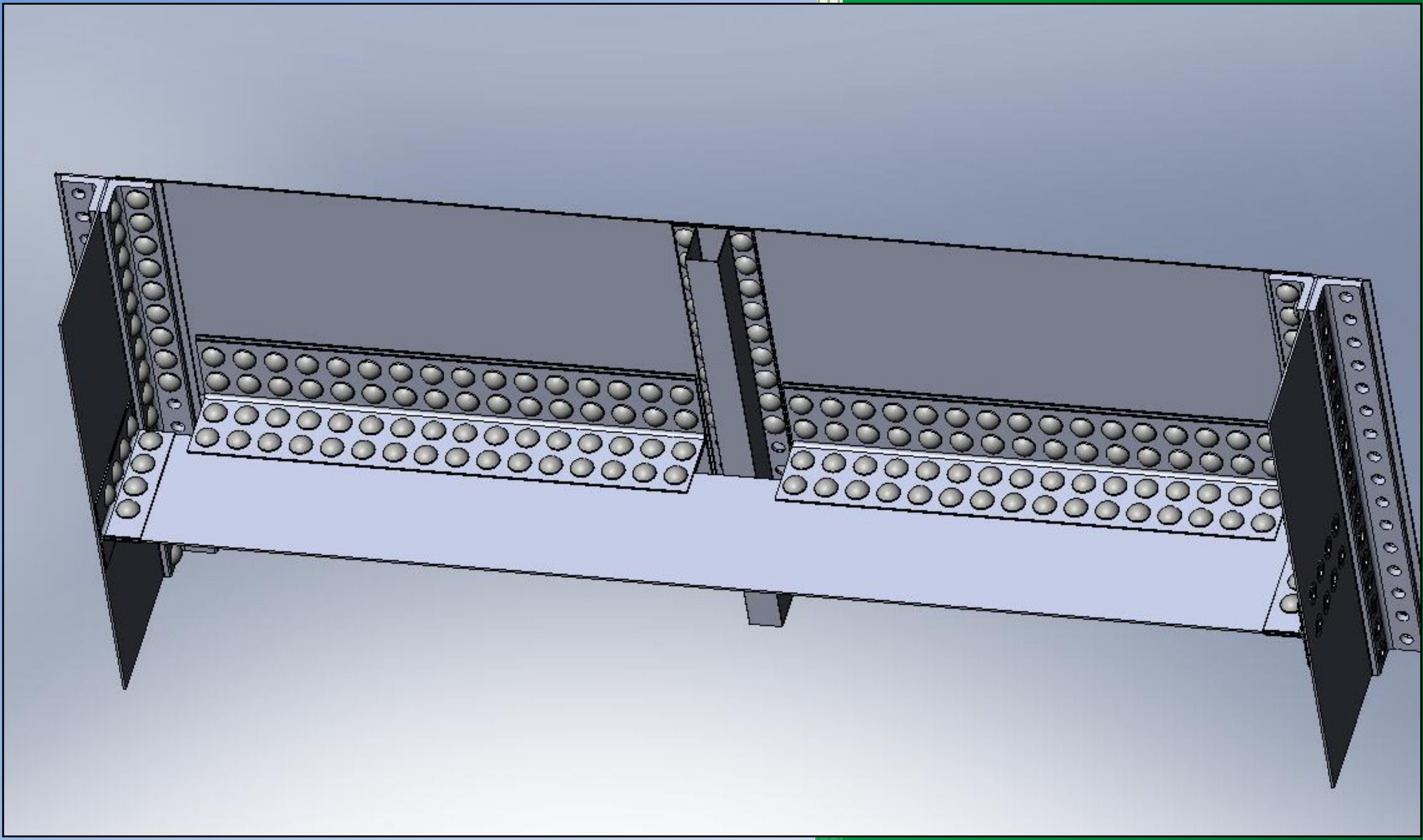
MAE 154B

Final Design Report



Team 1 Members:

June 1, 2009
MAE 154A Design Report

FDR List of Revisions & Additions Since the Revised CDR

- Pg. 4-5: Updated Introduction to match context of final report and include final division of duties section and timeline
- Pg. 26, 28: Moved the discussion on Von Mises Failure Check to a new subsection of the "Failure Criteria" section.
- Pg. 28-32: Moved Buckling section to a new subsection of the "Failure Criteria" section.
- Pg 33: Added Fracture Section
- Pg 34: Added Fatigue Section
- Pg 34: Added Wing Divergence Section
- Pg 35: Added Aileron Reversal Section
- Pg 36: Added Flutter Section
- Pg 38-45: Expanded Monte Carlo Optimization Section to include optimization results, post-processing details, and correlations for geometry parameters.
- Pg 46: Removed "Preliminary Calculations" section of CDR and replaced it with "Final Design" section.
- Pg 47-50: Added Calculation Run-Through Section
- Pg 51-60: Expanded CAD section to include more discussion in CAD assembly. Added rivet design and FEA work.
- Pg 61: Added Conclusion
- Pg 61: Removed "Future Work" section from CDR and moved contents and work timeline to "Division of Duties" section of introduction.
- Added Appendix A (CAD Pictures)
- Added Appendix B (Wing Geometry Correlations)
- Added Appendix C (Geometry, Inertia, and Stress Correlations)
- Added Appendix D (Matlab Code)

Table of Contents

FDR List of Revisions & Additions Since the Revised CDR 1
1) Introduction 4
1.1) Project Description 4
1.2) Division of Duties 4
2) Aerodynamic Loads 5
2.1) Lift Analysis & JavaFoil Data 5
2.2) V-N Diagram 7
2.2.1) Maneuvering Envelope 7
2.2.2) Gust Envelope 8
2.2.3) V-N Diagrams 9
2.3) Load Distributions 11
2.3.1) Lift Distribution 11
2.3.2) Drag Distribution 13
3) Preliminary Sizing 14
3.1) Double Cantilever Beam Approximation 16
4) Stresses 17
4.1) Section Centroid and Moments of Inertia 18
4.2) Bending Stresses 19
4.3) Shear Flow 19
4.3.1) Panel Idealization 20
4.3.2) Combined Flexural and Torsional Shear Flow 21
4.3.3) Pure Torsional Flow 24
4.3.4) Shear Center 25
4.4) Von Mises Stresses 26
5) Deformation Distributions 26
5.1) Deflection Distribution 26
5.2) Angle of Twist Distribution 27
6) Failure Criteria 28
6.1) Von Mises Failure Criteria 28
6.2) Buckling Analysis & Calculations 28
6.2.1) Compressive Column Buckling 29
6.2.2) Compressive Plate Buckling 31

6.2.3) Plate Shear Buckling 32
6.3) Fracture 33
6.4) Fatigue 34
7) Aeroelasticity 34
7.1) Wing Divergence 34
7.2) Aileron Reversal 35
7.3) Flutter 36
8) Monte Carlo Optimization 38
8.1) Randomized Wing Geometry 38
8.2) Analysis Procedure 39
8.3) Optimization Results 39
8.4) A Study of Geometry, Inertia, and Stress Correlations 43
9) Final Design 46
10) Calculation Run-Through for Final Design 47
11) CAD and FEA 51
11.1) CAD Details 51
11.2) Riveted Joints 52
11.3) FEA Structural Verification 55
12) Conclusion 61
13) References 62
Appendix A: CAD Pictures 63
Appendix B: Optimization Scatter Plots: 70
Appendix C: Geometry, Inertia, and Stress Correlations 73
Appendix D: Matlab Code 94

1) Introduction

This report outlines the work completed as part of the MAE 154B wing design project. The goal of this project is to design the structural components for a wing for a lightweight utility aircraft while maintaining strict adherence to Federal Aviation Administration requirements. The analysis presented focuses on conveying the completed work on the project, including the aerodynamic analysis, stresses, failure criteria, aeroelastic considerations, optimization, CAD, and FEA. Possible future work is also presented, including more rigorous analytical approaches and more in-depth and detailed CAD and finite element analysis to verify the structural capabilities before releasing for production. In discussing these aspects of the project, this report aims to elucidate the robust theory that is being implemented in order to meet the ambitious requirements set forth for this project.

1.1) Project Description

The primary objective of this project is to design a wing for a utility aircraft which will satisfy Federal Aviation Administration airworthiness requirements (FAR 23) while maintaining a minimum weight of the wing structure. The wing is built up from a rib-and-spar structure that is covered by a thin aluminum skin with stringers running down its length spanwise. These components will serve to maintain proper geometry and structural integrity throughout the wing over the complete range of operating conditions for the aircraft. The challenges of this project, then, are twofold. The first is to develop an effective method for analyzing the load-bearing structural phenomena inside the wing. The second is to develop a methodology that will allow the rapid convergence of the design onto a structural geometry that optimizes the wing's performance under the given loading conditions.

1.2) Division of Duties

The bulk of the work for this project was split up into three different task areas: Analysis, Programming and Optimization, and Solid Modeling/Finite Element Analysis. Since there are three members in the group, one team member had primary responsibilities in each task area, though the heavy workload in the analysis and programming/optimization subsections led the group to divide some of the work. In terms of solid modeling, while James has had the most experience, Francesco was interested in learning CAD and hence was made primarily responsible for the solid modeling and finite element analysis, though James and Ian would provide instruction should it become necessary. James was primarily in charge of analysis, and Ian was tasked with most of the optimization and post-processing work. All members actively contributed to the programming aspect. A detailed layout of the completed tasks is presented in Table 5.1.

Week	Completed Work
Week 1	- Javafoil Data (I)
Week 2	- V-n Diagram (J) - Load Distributions and preliminary sizing (F) - Complete preliminary design report and presentation (All)
Week 3	- PDR Presentation (F) - Research theory on bending, torsion, and buckling using Von-Mises Yield criteria (All)
Week 4	- Bending Stress Computation (F) - Shear flow and shear center Computation (J) - Buckling analysis (I) - Begin development of Monte Carlo Optimization program (All)
Week 5	- Continue Monte Carlo Optimization programming, including randomly-generated spar and stringer locations (All) - Von-Mises yield criteria (F) - Begin Solid Modeling (F)
Week 6	- Continue Optimization and modifying bending, shear, and buckling calculations to accommodate random wing layouts. (All) - Deflection Distribution (F) - Angle of Twist Distribution (J) - Complete critical design report and presentation (All)
Week 7	- CDR Presentation (J) - Continue CAD & FEA (F) - Fracture and Fatigue (I) - Divergence analysis (J) - Continue Optimization & Begin Post-Processing (I)
Week 8	- Flutter Analysis & Calculations (J) - Continue CAD (including Rivet Design) & FEA (F) - Continue Optimization & Post Processing (I,J)
Week 9	- Address Aileron Reversal (F) - Finalize CAD & FEA (F) - Finalize Design, Post-Processing, Calculation Run-through (I,J) - Complete final design report and presentation (All)
Week 10	- FDR Presentation (I)

Table 5.1: Schedule for Remaining Work and Completed Work. The member that is primarily responsible for each task is given by the following key:

2) Aerodynamic Loads

2.1) Lift Analysis & JavaFoil Data

Before any structural analysis can be attempted, the aerodynamic loads on the aircraft must be analyzed and understood. This analysis was completed by evaluating the airfoil lift characteristics under conditions of viscous flow corresponding to several different flight altitudes at maneuvering velocity. The vast majority of this work was completed using an online Java-language airfoil analysis program called JavaFoil. Initial analysis was run simultaneously in both JavaFoil and a similar command line program called Xfoil. However Xfoil was found to be more difficult to work with in addition to offering limited analysis capabilities for negative angles of attack. For this reason, all analysis presented here uses JavaFoil in order to ensure consistent results. It is important to also note that there is a known degree of inaccuracy in these results. Analysis in both Xfoil as well as JavaFoil produced a linear portion of the lift curve with slope greater than 2π . This would suggest that the rest of the lift curve data may be flawed, because the ideal lift curve slope limit is 2π .

The analysis of the flight envelope was performed at three different altitudes. The net effect of altitude difference on the performance of the airfoil came from viscous effects and is represented in the form of the dimensionless Reynolds Number, which is defined by Equation 2.1.

$$\mathrm{Re} = \frac{\rho U c}{\mu} \tag{2.1}$$

Here, ρ is the air density, U is the airspeed, c is the wing chord length, and μ is the dynamic viscosity. As altitude increases, the density of air decreases (tables providing the ambient density as a function of altitude are provided in many texts, such as White [1]) while dynamic viscosity remains fairly unchanged. The result is that larger Reynolds numbers are observed at low altitudes. Numerically, Reynolds numbers of 8,200,000; 7,400,000; and 6,300,000 were calculated based on cruise velocities of 168 mph, 196 mph, and 256 mph corresponding to altitudes of 0 ft; 10,000 ft; and 25,000 ft respectively. This data was then input into JavaFoil where the plot seen in Figure 2.1 was produced. JavaFoil shows a very small dependence on altitude for the lift curve, however this dependence is so slight that the analysis presented here will assume that the lift curve is constant for all altitudes.

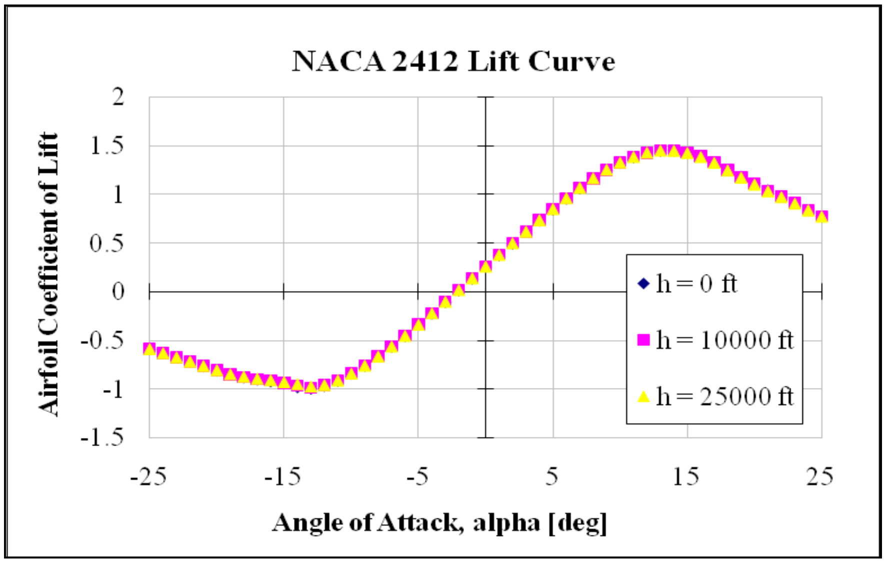


Figure 2.1: NACA 2412 Airfoil Lift Curve Slope for flight altitudes of 0 ft, 10000 ft, and 25000 ft

Figure 2.1 gives the C_{lmax} and C_{lmin} values for the airfoil by selecting the highest and lowest C_l values respectively. To correct this 2-D curve for a 3-D wing it was assumed that the zero lift angle of attack $\alpha_{L=0}$ and the angle of attack at stall were identical between the airfoil and 3-D wing. In reality this is not a completely accurate approximation as the angle of attack at stall does indeed shift, however the somewhat large aspect ratio of the wing for this aircraft will minimize these effects. Then, using Equation 2.2, the lift curve slope of the 3-D wing can be calculated from that of the 2-D airfoil.

$$C_{L,\alpha} = \frac{C_{l,\alpha}}{1+\frac{C_{l,\alpha}}{\pi Ae}} \tag{2.2}$$

Where $C_{L,\alpha}$ is the 3-D lift curve slope of the wing, $C_{l,\alpha}$ is the lift curve slope of the 2-D airfoil, A is the aspect ratio of the wing, and e is the Oswald efficiency factor of the wing. Following this 3-D lift curve slope and extrapolating from the zero lift angle of attack up to the airfoil stall angle of attack gives the stall conditions for the wing. The C_L value at this location is assumed to be the 3-D wing angle of attack at stall, which is used in later calculations.

2.2) V-N Diagram

The V-n Diagram details the load limits that the aircraft must be designed to withstand. The aircraft is safe to operate anywhere within the maneuvering envelope without the risk of permanent structural damage. The gust envelope takes into account the loads imposed by a sudden vertical gust. These will be calculated at flight altitudes of *h=0 ft, 10000 ft, and 25000 ft*.

2.2.1) Maneuvering Envelope

For utility-category aircraft, the positive limit load factor is $n=4.4$ while the negative limit load factor is *0.4* times the positive limit, as outlined in FAR 23.337 [2]. This represents the Operational Load Limit (OLL). Exceeding this value during flight may induce permanent structural damage, more specifically yielding. A factor of safety of *1.5* has also been implemented as detailed in FAR 23.303 [2]. This represents the Structural Load Limit (SLL) or "Ultimate" load limit, beyond which fracture may occur.

The lift constrained load factor is given by Equation 2.3. This was used for both the positive and negative lift constrained load factor. $C_{L,max}$ was used for the positive limit load factor while $C_{L,min}$ was used for the negative limit, both of which were computed using JavaFoil data as shown in section 2.1. Note that to compute $C_{L,max}$ and $C_{L,min}$, an iterative process with Java-foil is necessary. The maneuvering speed

V_A represents the velocity at which the positive lift constrained load factor (Eqn 2.3) is equivalent to the positive limit load factor (*n=4.4*). This requires knowledge of $C_{L,max}$ and $C_{L,min}$. But, since $C_{L,max}$ and $C_{L,min}$ vary with flight velocity, iteration must be performed to converge to a maneuvering speed that agrees with the lift coefficient computed through JavaFoil.

$$n_{LiftCons} = \frac{V^2 \rho S C_{L,\max/\min}}{2W_{TO}} \tag{2.3}$$

V is the flight velocity, ρ is the air density, S is the wing area, and W_{TO} is the takeoff weight. For the equation, all units are in normal English or Metric (sl-ft-sec in the calculations presented here).

The maneuvering envelope is constructed by implementing Equation 2.3 up to the ultimate load limits for both positive and negative loads separately. Horizontal lines are created to represent the structural and ultimate load limits for both positive and negative loads as well. These are shown in Figures 2.2-2.4 in Section 2.2.3.

2.2.2) Gust Envelope

As specified by FAR 23.333 [2], we will assume a vertical gust velocity of $U_g=25$ *ft/s* at the dive speed V_D, $U_g=50$ *ft/s* at the cruise speed V_C, and $U_g=66$ *ft/s* at the maneuvering speed V_A. The gust load factors as a function of flight velocity are computed using Equation 2.4, obtained from FAR 23.341 [2].

$$n_{gust} = 1 + \frac{K_g a_w U_g V}{498\left(\frac{W_{TO}}{S}\right)} \tag{2.4}$$

It should be noted that for Equation 2.4 the velocity V must be supplied in knots. This is due to the correction factor of *498*. Meanwhile, U_g is the gust velocity (this, however, is in feet per second), a_w is the lift-curve slope of the wing, and the gust alleviation factor K_g is given by Equation 2.5.

$$K_g = \frac{0.88\mu}{5.3+\mu} \tag{2.5}$$

The airplane mass ratio μ is given by Equation 2.6.

$$\mu = \frac{2\left(\frac{W_{TO}}{S}\right)}{\rho c a_w g} \tag{2.6}$$

c is the wing chord, and *g* is the acceleration due to gravity.

The gust envelope, as described in Megson's *Aircraft Structure for Engineering Students* [3], is constructed by implementing Equation 2.4 for the V_A gust velocity ($U_g=66\ ft/s$) for velocities ranging from *0* to V_A is reached. This is repeated for the V_C gust velocity and V_D gust velocity in a similar fashion, and lines connect the successive end points to form the outer envelope. These can be seen in Figures 2.2-2.4 in the following section.

2.2.3) V-N Diagrams

Implementation of equations 2.3-2.6 for flight altitudes of *h=0 ft, 10000 ft, and 25000 ft* produce the plots shown in Figures 2.2-2.4. From Figures 2.2-2.4, it can be seen that the maximum lift load factor curves are stretched out as flight altitude increases. This indeed makes sense since at higher altitudes, density is reduced which thus reduces the amount of lift produced. It can also be seen that at 25,000 ft, the positive lift produced at the dive speed does not reach the structural load limit and the maneuvering speed exceeds the cruise speed, thus producing a discontinuity in the gust envelope as seen in Figure 2.4.

For purposes of structural sizing, analysis is performed at four extreme loading conditions on the V-n diagram. The Positive High Angle of Attack (PHAA) is the loading condition represented by the intersection between the positive operational load limit line and the positive maximum lift curve. The Positive Low Angle of Attack (PLAA) is at the intersection between the positive operational load limit line and the dive speed. The Negative High Angle of Attack (NHAA) and Negative Low Angle of Attack (NLAA) are defined similarly except are for the negative loads. Should the gust envelope extend beyond the maneuvering envelope in any of these four locations, the load factor of the gust envelope is instead used for the extreme loading condition. The high angle of attack conditions are characterized by a high coefficient of lift and high drag. The low angle of attack conditions are characterized by a high lift force. Designing to accommodate these four extreme loading conditions will guarantee that the wing will not undergo structural damage so long as operational load limits are not exceeded.

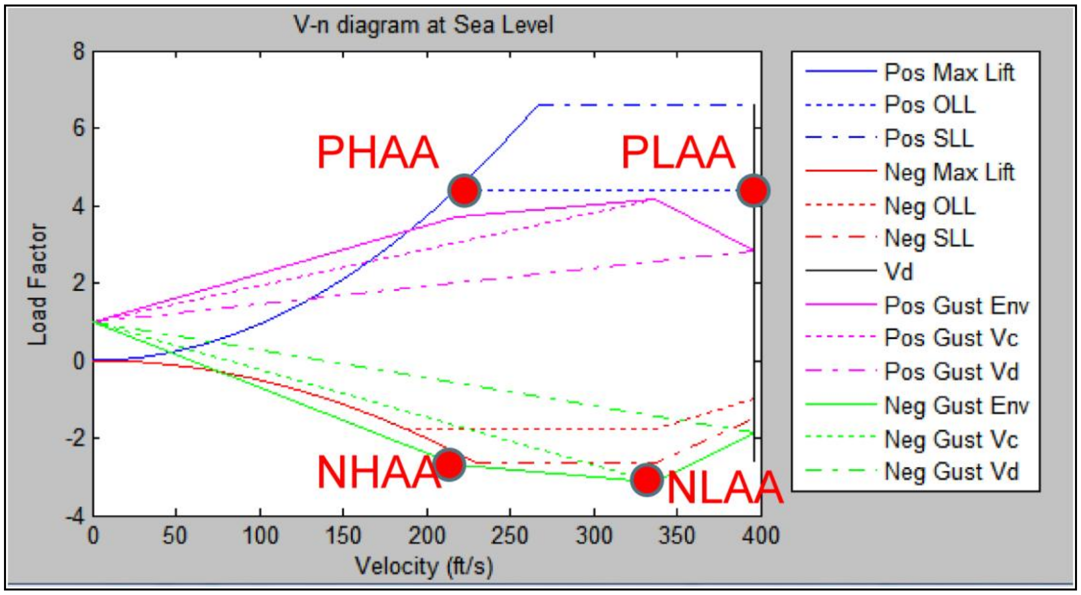


Figure 2.2: V-n Diagram at Sea Level, along with the locations of the four extreme loading conditions

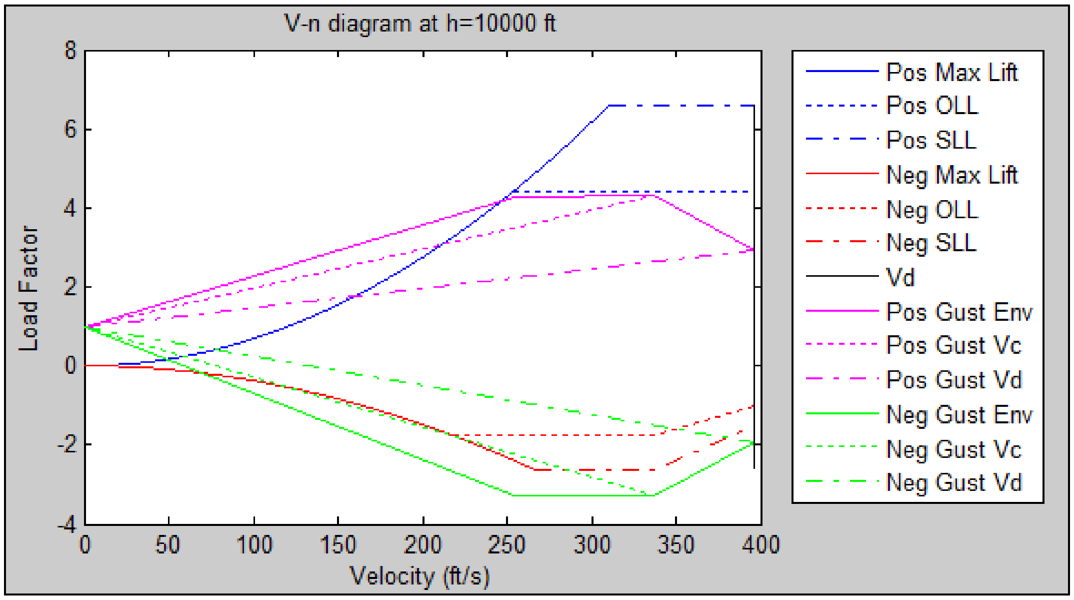


Figure 2.3: V-n Diagram at flight altitude h=10,000 ft

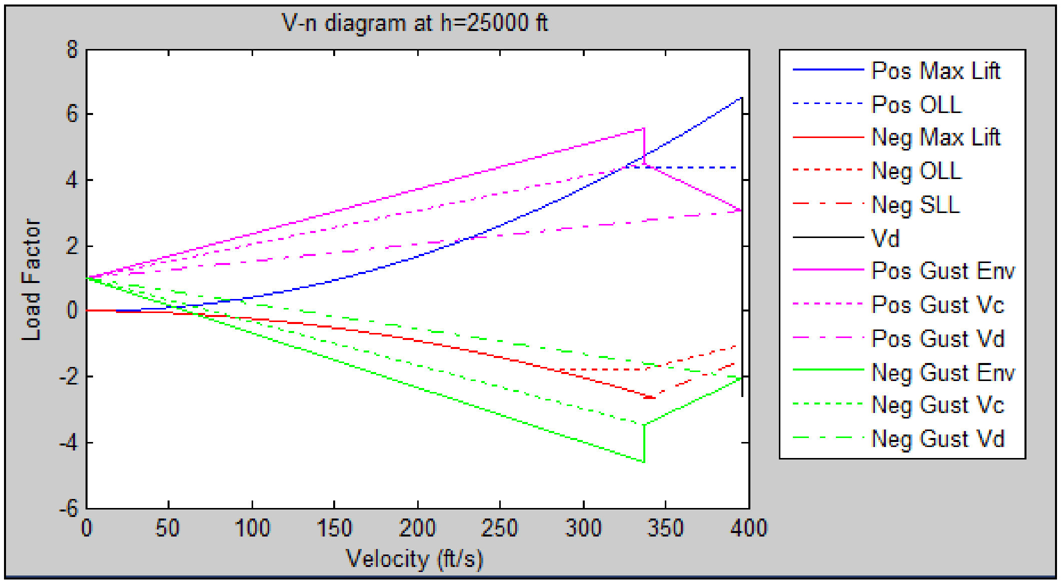


Figure 2.4: V-n Diagram at flight altitude h=25,000 ft

2.3) Load Distributions

The first step in sizing a structure is to identify the loads that act on it. For a wing these are mainly aerodynamic forces, referring to the lift and the drag. These forces can be computed using data from wind tunnel tests or from CFD analyses as it was done in this case (via JavaFoil). Since the wing needs to sustain loading not only for level flight, but also for maneuvering and wind gusts, the lift calculated was multiplied by a loading factor obtained from the V-N diagrams. Four critical conditions (PLAA, PHAA, NLAA, NHAA) are identified in every diagram at each of three different altitude conditions. A total of twelve different conditions and relative load distributions are evaluated and the wing is sized for the worst one.

2.3.1) Lift Distribution

The lift force is produced by the airflow around the wing shape and along the span. Therefore, it cannot be considered a concentrated force applied in a single point of the structure. The lift is better described as a distributed load with a particular shape along the wing. The Kutta-Jukowski theorem shows this expressing the lift per unit span with Equation 2.7.

$$L'(y) = \rho v \Gamma(y) \tag{2.7}$$

ρ is the air density, v is the stream velocity, and Γ is the local circulation as a function of the spanwise coordinate y. The result of this wing theory is that the most efficient wing distribution is elliptical as described by Equation 2.8.

$$L'(y) = \rho v \Gamma_0 \sqrt{1-\left(\tfrac{2y}{b}\right)^2} \tag{2.8}$$

Here, *b* is the total wing span. This ideal distribution is attainable only with an elliptical wing geometry. For straight rectangular wings, like the one in this case, a correction is needed in order to find more realistic results. The correction that was used in our case consists of the ideal elliptical distribution averaged with a rectangular distribution (given by Equation 2.9).

$$L'(y) = \frac{L}{b} \tag{2.9}$$

L is the total lift acting on the airframe which is known from the airplane weight, *W*, and load factor, *n*. Figure 2.5 shows a plot of the rectangular and elliptical lift distributions spanwise from tip to root.

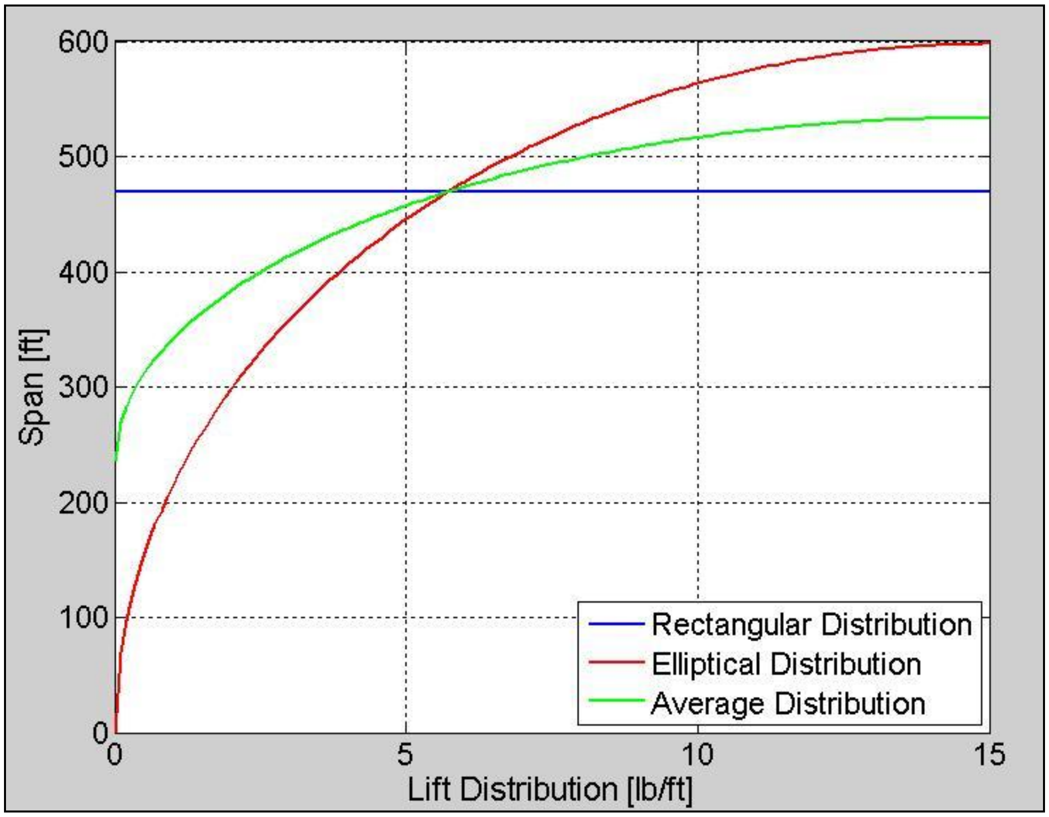


Figure 2.5: Rectangular, Elliptical, and averaged lift distribution over wing half-span from tip to root

The load factor, *n*, is given by Equation 2.10.

$$L = nW \tag{2.10}$$

Finally, two different expressions for total lift can be used to calculate the lift distribution (Right and left sides of Equation 2.11). Equation 2.11 give lift as a function of Γ_0 by integrating *L'(y)* with respect to *y*.

$$L = \frac{1}{2}\rho v^2 S C_L = \frac{\pi}{4}\rho v b \Gamma_0 \tag{2.11}$$

Here, *S* is the wing surface area and C_L is the wing lift coefficient. Solving for Γ_0, the result of this equation can then be combined with the Kutta-Jukowski equation and used to complete the expression of *L'(y).* The resultant lift load plot is shown in Figure 2.5.

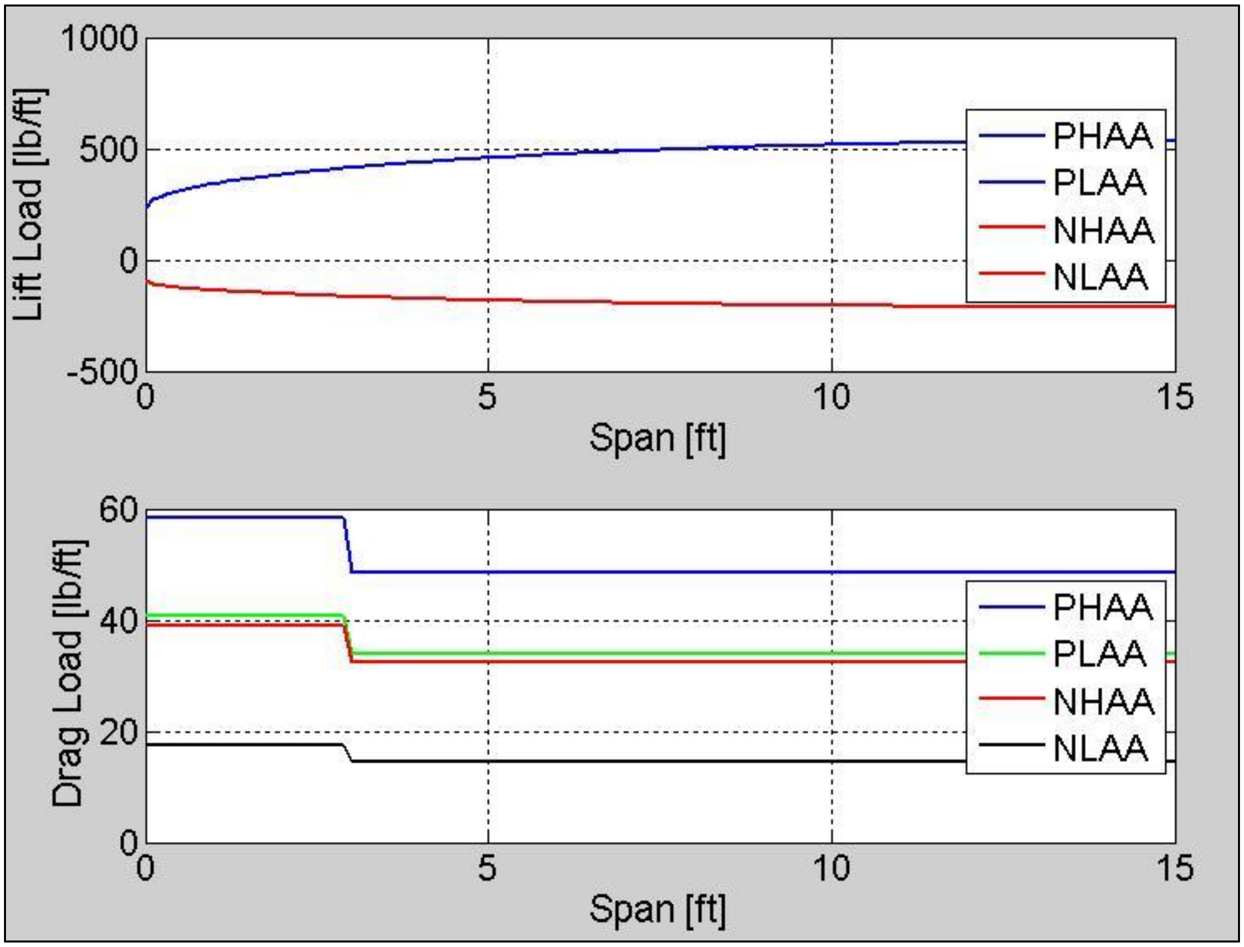


Figure 2.6: Lift and Drag Distribution as a function of spanwise distance, measured from the tip of the wing and proceeding towards the root (half-span shown)

2.3.2) Drag Distribution

Similarly to the lift force, the drag force is also a distributed load. Its function is somewhat linear along the span, however it suddenly increases towards the wing tips. To simulate this larger drag at the wing ends a step function is used: the drag is constant up to 80% of the span where it jumps up by 20%

reaching a larger constant value. The total drag value was computed using the traditional aerodynamic relations for the drag coefficients, shown in Equations 2.12-2.14.

$$C_D = C_{D,p} + C_{D,i} \tag{2.12}$$

$$C_{D,i} = \frac{C_L^2}{\pi AR \cdot e} \tag{2.13}$$

$$AR = \frac{b^2}{S} \tag{2.14}$$

C_D is the total drag coefficient, C_{Dp} is the parasite drag coefficient given by CFD analyses, C_{Di} is the induced drag coefficient, *AR* is the wing's aspect ratio, and *e* is the wing's Oswald's efficiency factor. The total drag is then obtained using Equation 2.15.

$$D = \frac{1}{2}\rho v^2 S C_D \tag{2.15}$$

In order to make sure that the total drag matches the integral of the drag distribution, a geometric analysis was performed. Using this process the values of the two constant functions were found to be those shown in Equation 2.16.

$$\begin{aligned} D_1' &= 0.9615\frac{D}{b} \\ D_2' &= 1.154\frac{D}{b} \end{aligned} \tag{2.16}$$

D'_1 is the amount of drag per unit span acting on the inner wing and D'_2 is at the wing tips. The resultant drag load plot is shown in the previous section in Figure 2.5.

3) Preliminary Sizing

The structure's preliminary sizing was performed focusing mainly on the bending moment acting on it. For this reason the bending moment distribution along the wing in the spanwise direction (y-coordinate) is required so that the stress in the wing itself (nothing more than a cantilever beam) can be evaluated. Equation 3.1 is used to obtain the shear force, *V(y)*, and bending moment, *M(y)*, given the generic load function, *p(y)*, all as a function of the spanwise coordinate.

$$\frac{\partial^2 M(y)}{\partial y^2} = \frac{\partial V(y)}{\partial y} = -p(y) \tag{3.1}$$

So far only the load functions due to lift and drag were computed. They are expressed in the lift-drag frame of reference that has the axes parallel and perpendicular to the air stream respectively. It becomes necessary to rotate the loads to the structural coordinate system that was chosen to be with the "x" axis along the airfoil chord, the "y" axis along the wing span and the "z" perpendicular to them. The two reference frames are tilted one with respect to the other by an angle, α, called 'angle of attack', which is actually a key parameter in the aerodynamic analysis. Once this rotation was performed the load functions in the "x" and "z" direction were available. From Equation 3.1, it can be easily seen that a double integration is necessary to determine the bending moment. This integration is performed using two different methods. The first is a simple numerical integration which implements the trapezoidal method. The second one, used to double check the numerical integration results, uses a polynomial that curve fits the load function and then analytically integrates the resulting polynomial. The two methods give very similar results, and hence the numerical trapezoidal method was used. The loading, shear, and bending moment distributions in the x and z directions are shown in Figures 3.1 and 3.2 respectively for all four extreme loading conditions at sea level.

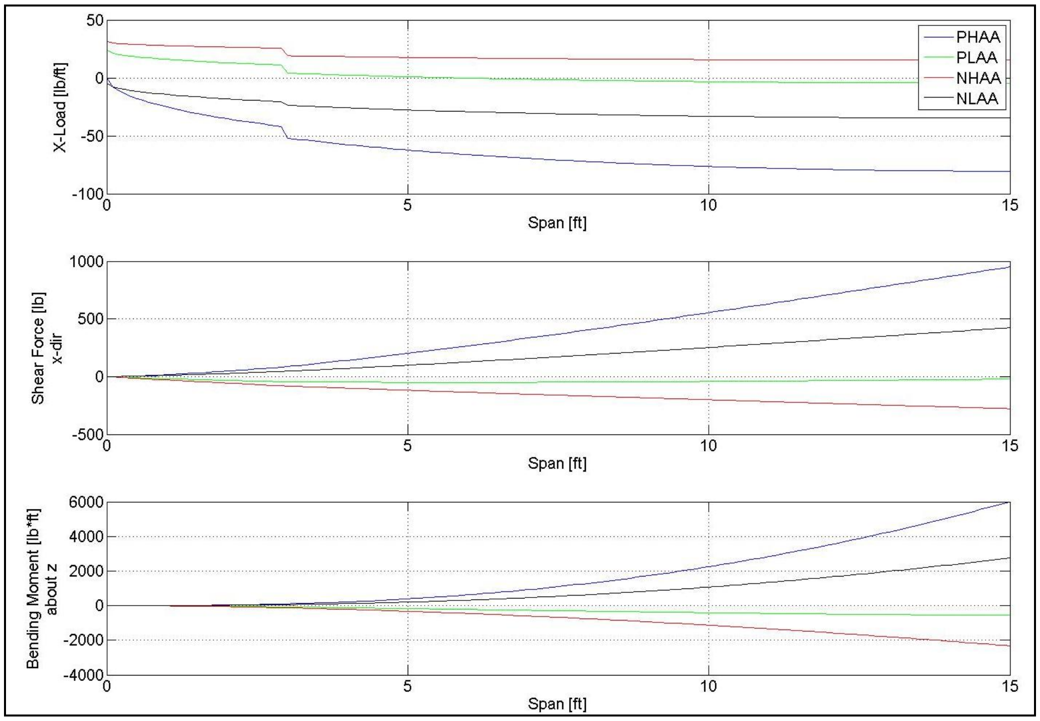


Figure 3.1: Load, Shear Force, and Bending Moment distributions in the X-direction on the wing half-span, measured from the tip of the wing and proceeding toward the root

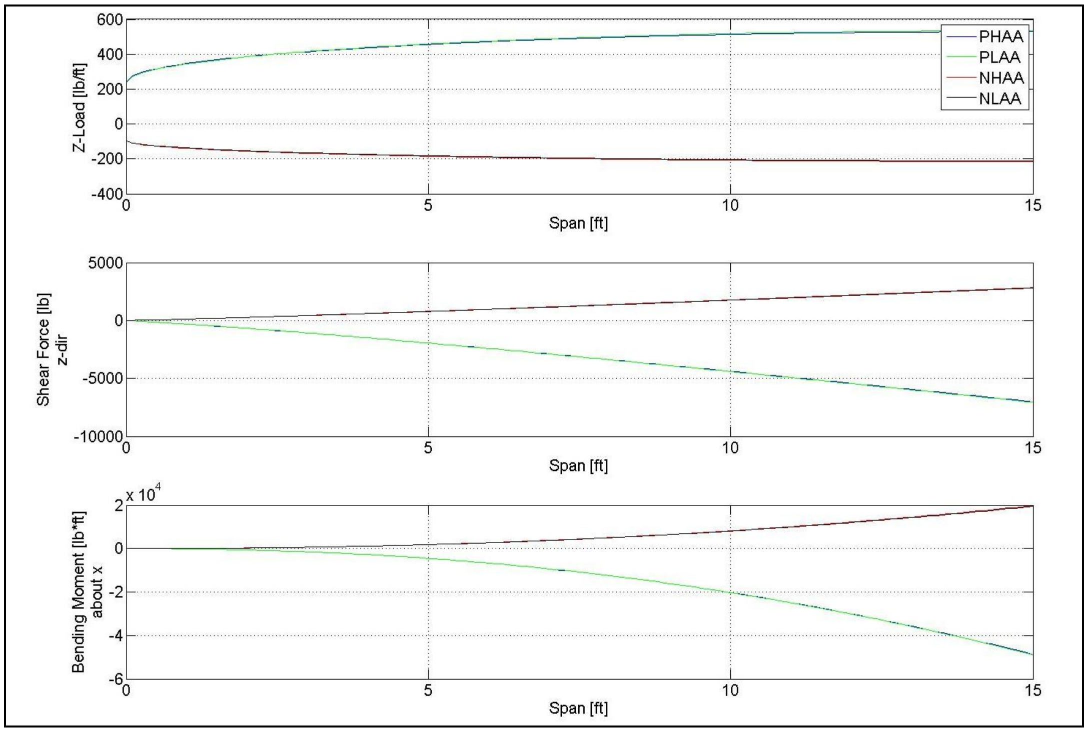


Figure 3.2: Load, Shear Force, and Bending Moment distribution in the Z-direction on the wing half span, measured from the tip of the wing and proceeding toward the root

3.1) Double Cantilever Beam Approximation

The preliminary sizing of the wing structure is based on an important approximation. Since the spars carry the majority of the bending load these are the objects of the first sizing process. They are assumed to be two I-Section cantilever beams whose height is determined by the airfoil's thickness and whose width is to be determined. The location where the maximum bending moment acts is the wing root section. The starting point for the analysis is the de Saint Venant relation for composed bending on symmetric section beams, shown in Equation 3.2.

$$\sigma_y = \frac{M_z x}{I_z} \pm \frac{M_x z}{I_x} \tag{3.2}$$

σ_y is the tensile or compressive stress in a particular beam location, M_z and M_x are the bending moments at the root chord and I_z and I_x are the section's moments of inertia. The moments of inertia used at this point are the resultants of the contribution of both beams. They can be expressed as a function of the spar width, t. The highest stress is located at the farthest point from the section's centroid that means and that is the position where the yield has to be evaluated. The maximum allowable value of $\sigma_{y,max}$ is found from the aluminum yield stress (assumed to be 71000 psi for aluminum 7475-T61) and the factor of safety, *FOS*, given by Equation 3.3.

$$\sigma_{y,\max} = \frac{\sigma_{yield}}{\text{FOS}} \qquad \text{FOS} = 1.25 \tag{3.3}$$

It is now possible to express the spar thickness as a function of other known parameters (spar dimensions, maximum allowable stress and bending moments). Solving for *t* gives the preliminary sizing of the spars. This process has to be repeated for all twelve conditions given by the V-N diagrams, giving the final result shown below.

$$t = 0.56 \text{ in}$$
$$h_1 = 6.83 \text{ in}$$
$$h_2 = 5.42 \text{ in}$$

where h_1 and h_2 are the first and second spar's height respectively. The spar web thickness *t* still looks pretty high, but further optimizations will lead to better values.

4) Stresses

The stress calculations are performed in the same conventional Cartesian coordinate system:

x – Chord-wise axis pointing towards the trailing edge

y – Span-wise axis oriented from wing tip to wing root

z – Perpendicular axis pointing upwards to complete the right-hand frame of reference

The axis origin is usually located at the airfoil leading edge. The text will specifically say if the system is centered at the airfoil's centroid.

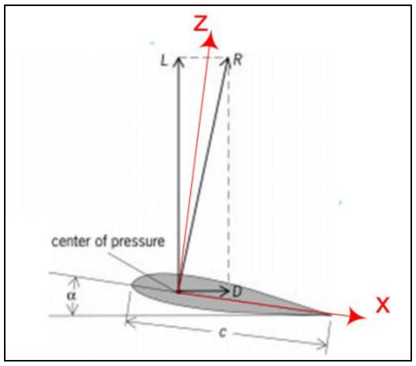


Figure 4.1: Body-fixed coordinate system used in calculations

4.1) Section Centroid and Moments of Inertia

For a more accurate sizing of the wing the section centroid and the moments of inertia have to be computed for the actual wing section. This task is performed splitting the section's skin in many small rectangular panels. The area of these elements is calculated and then attributed to point masses whose distance from the axes origin (the airfoil leading edge) is the same as that of the rectangles. With a sum of these distances weighted on the areas gives the centroid location. This calculation must be performed for both the x and z-directions, as indicated by Equations 4.1 and 4.2.

$$x_c = \frac{\sum_{i=1}^{n} x_i A_i}{\sum_{i=1}^{n} A_i} \tag{4.1}$$

$$z_c = \frac{\sum_{i=1}^{n} z_i A_i}{\sum_{i=1}^{n} A_i} \tag{4.2}$$

Here, x_c and z_c represent the x and z-coordinates of the centroid location while *n* represents the number of elements over which the centroid is to be computed, and x_i, z_i, and A_i represent the coordinates and area of the elements, respectively.

Also, the moments of inertia were calculated. Since each small section of area was considered to be a point area, the actual moment of inertia of that section is negligible. However, a non-negligible contribution comes when the moment of inertia is moved from the element centroid to the centroid of the whole structure. This contribution is found using the parallel axis theorem, defined by Equations 4.3 and 4.4.

$$I_x = I_{x',i} + A_i z_i^2 \tag{4.3}$$

$$I_z = I_{z',i} + A_i x_i^2 \tag{4.4}$$

Where $I_{x,i}$ and $I_{z,i}$ are the area moments of inertia about the centroid, and $I_{x',i}$ and $I_{z',i}$ are the area moments of inertia of the elements with respect to their own centroid. Also, the parallel moment of inertia is calculated. Again the contribution of the elements themselves are negligible, and the parallel axis theorem is defined by Equation 4.5 [6].

$$I_{xz} = I_{x'z'} + A_i xz \tag{4.5}$$

Accounting for the negligible element area moment of inertia, the area moments of inertia of the whole wing about the centroid are given by Equation 4.6.

$$\begin{aligned} I_x &= \sum_{i=1}^{n} A_i z_i^2 \\ I_z &= \sum_{i=1}^{n} A_i x_i^2 \\ I_{xz} &= \sum_{i=1}^{n} A_i x_i z_i \end{aligned} \tag{4.6}$$

4.2) Bending Stresses

The stresses, σ_y, due to the bending moments in the structure are oriented in the span-wise direction 'y'. These are therefore normal stresses, perpendicular to the wing cross section. Since the wing section is not exactly symmetric about both axes a more complicated equation is required to compute these σ_y. Along with the moments of inertia about the 'x' and 'z' axis, I_x and I_z respectively, also the product of inertia, I_{xz}, is included in the expression in order to account for the structure's asymmetry, as shown in Equation 4.7.

$$\sigma_y = -\frac{M_z I_x + M_x I_{xz}}{I_x I_z - I_{xz}^2} \cdot x + \frac{M_x I_z + M_z I_{xz}}{I_x I_z - I_{xz}^2} \cdot z \tag{4.7}$$

where M_x and M_z are the bending moments in the structure (about 'x' and 'z' axis respectively) and x and z are the coordinates (relative to the centroid) of the position that has to be evaluated [4].

It is easy to see that σ_y is strongly dependant on the distance from the section's centroid. There is a line, called the neutral axis, passing through the centroid on whose points the normal bending stresses are zero. M_x and M_z vary with the location along the wing span and are maximum at the root, which is the critical section where yield or failure has to be evaluated.

4.3) Shear Flow

To compute the shear stresses acting along the plane of any cross-section of the wing, Megson [3] and Sun's [4] thin-web shear flow theory is implemented. Shear flow is simply a shear force per unit length. The shear stress in a skin or spar panel can be computed from the shear flow by dividing by the thickness of the panel. For this theory to be applicable, the wing structure must be idealized into an arrangement of axial stress-carrying booms with finite cross sectional area and tangential shear stress-carrying panels with a very small but finite thickness. So long as the direct-stress carrying capacity of the skin sections are incorporated as additional area to stringers and spar caps, the approximation of constant shear flow between any two adjacent stringers or spar caps can be made. Such an approximation greatly facilitates the computation of the shear flow in the wing section.

The total shear flow computation requires a three-step process. First, the flexural shear flow component of each skin or spar panel due to the shear loads induced by the aerodynamic forces on the wing section is computed. Given these values, the torsional component of this same load is computed. Lastly, the moment about the aerodynamic center due to the wing's camber is incorporated as a pure torque, inducing an additional torsional flow on the wing panels. These three sets of shear flows are then summed to produce the total shear flow distribution in the wing section.

4.3.1) Panel Idealization

A panel with finite thickness is idealized as an infinitely thin web with two finite-area booms on both sides, as shown in Figure 4.1. The effective boom areas must incorporate the effect of a gradient in the direct-stress along the panel by ensuring that the moment about any arbitrary point due to the net bending stress of the actual panel is equivalent to the moment induced by the idealized panel. Doing so, the boom areas B_1 and B_2 shown in Equations 4.8 and 4.9 are obtained [3].

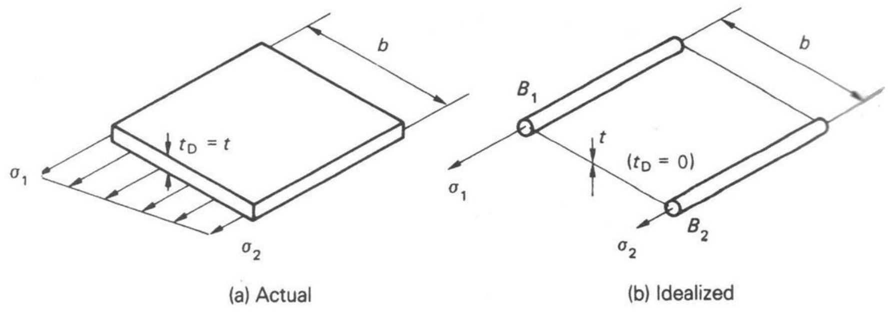


Figure 4.1: Panel Idealization for shear flow calculations [3]

$$B_1 = \frac{t_D b}{6}(2 + \frac{\sigma_2}{\sigma_1}) \tag{4.8}$$

$$B_2 = \frac{t_D b}{6}(2 + \frac{\sigma_1}{\sigma_2}) \tag{4.9}$$

t_D is the panel thickness, b is the panel width, and σ_1 and σ_2 are the axial stresses at booms 1 and 2 respectively.

The approximation of constant shear flow between adjacent booms can be made only when booms are spaced sufficiently close from each other. Since optimizing the wing requires minimizing the number of stringers in the wing section, the spacing between stringers may not be sufficient for this spacing requirement to be met. Thus, the panels between any two adjacent stringers and spar caps are additionally subdivided into 10 sections. The boom areas between panel subdivisions are computed from Equations 4.8 and 4.9 above and panel subdivisions adjacent to stringers or spar caps must incorporate this additional effective boom area into the adjacent stringer or spar cap.

4.3.2) Combined Flexural and Torsional Shear Flow

First, the stresses due to the shear loads on the wing section are computed. The new centroid and the moment of inertias about this centroid are computed for this idealized arrangement of booms using Equations 4.1-4.6 presented previously. The coordinate system used throughout the shear flow calculation is centered at this centroid. Since the wing consists of a closed multi-cell structure, the shear flow computation requires a special two-step process. First, hypothetical "cuts" are made at arbitrary points in the wing section. The number of cuts is equal to the number of cells in the section, and the cuts must be placed in such a way that ensures that no cell is completely enclosed. These cuts effectively convert a closed cell into an open cell and thus the structure can only hold flexural stresses and cannot support torsional stresses. This flexural shear flow component is computed first. Next, the resulting torsional shear flow component is computed by considering the torque induced by the shear load location being offset from the center of twist [4].

The three hypothetical cuts are chosen to be located at the upper right panel of each cell i.e. the panel in between the upper right spar cap of each cell and the boom area immediately to the left, as shown in Figure 4.2.

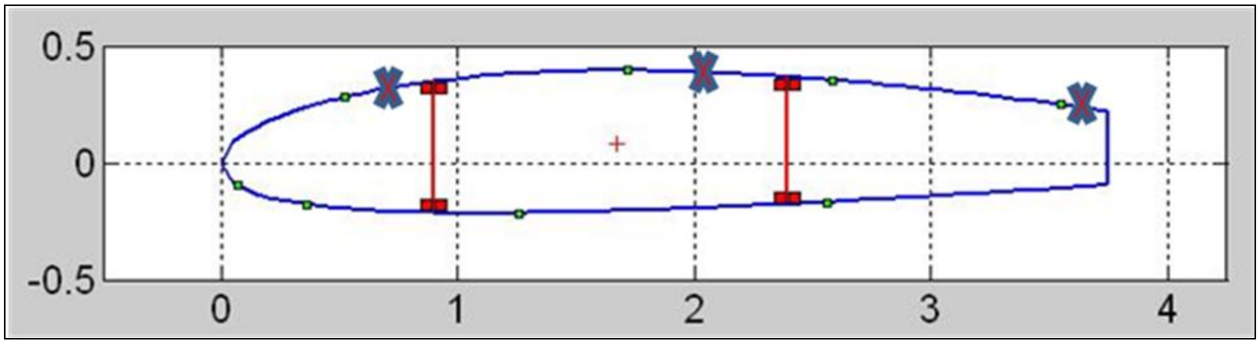


Figure 4.2: An arbitrary set of stringers (green circles) and spar locations, along with location of cuts (red X's) for shear flows.

Since these specific panels contain a cut, the flexural shear flow components are assumed to be 0 in these panels. Moving along in a counterclockwise direction for each cell, each successive panel contains a constant shear flow given by Equation 4.10 and is only a function of the shear flow $q_{b,i-1}$ in the previous panel and the location and area of the boom between the panel in question and the previous panel.

$$q_{b,i} = -\left(\frac{V_x I_{xx} - V_z I_{xz}}{I_{xx} I_{zz} - I_{xz}^2}\right) xA - \left(\frac{V_z I_{zz} - V_x I_{xz}}{I_{xx} I_{zz} - I_{xz}^2}\right) zA + q_{b,i-1} \tag{4.10}$$

I_{xx} and I_{zz} are the area moment of inertias about the x-axis and z-axis respectively, assuming centroid-centered axes, I_{xz} is the polar area moment of inertia, V_x and V_z are the shear loads at this particular cross-section of the wing in the x and z directions respectively, x and z are the x and z-locations of the boom, and A is the area of the boom. At junctions where more than two panels intersect, a slight modification must be made to Equation 4.10. The shear flow entering the junction must be equivalent to the shear flow exiting the junction, ensuring that the boom area at the center of the junction is still incorporated into the total shear flow. Thus, instead of a single $q_{b,i-1}$ term at the end of Equation 4.11, additional shear flow terms from other panels entering or exiting the junction must be included. The signs of these additional shear flows must match the counterclockwise convention for each cell mentioned earlier. Following this process for all panels, the flexural shear flow component throughout the wing structure can be computed.

The torsional shear flow component is constant in each cell except along panels that are common to more than one cell (i.e. the spars). The wing twist angle per unit spanwise length $d\theta / dy$ for a cell is computed using Equation 4.11, which involves the integration of the flexural and torsional shear flow components around the cell with respect to the tangential distance along the panel. Note that the torsional component is initially an unknown in this equation, though it will be solved later in the process. For the idealized wing structure with constant shear flow in the panels, the integration can be broken up into a summation of the shear flow components in all N panels of the cell, also shown in Equation 4.11 [4].

$$\frac{d\theta}{dy} = \frac{1}{2A_R}\oint_R (q_b + q_{s,R})\frac{ds}{Gt} \approx \frac{1}{2A_R}\sum_i^N [(q_{b,i} + q_{s,R})\frac{L_i}{G_i t_i}] \tag{4.11}$$

A_R is the area enclosed by cell R, $q_{s,R}$ is the torsional shear flow component in cell R, G_i is the shear modulus of the material used for panel *i*, t_i is the thickness of panel *i*, and L_i is the arc length of panel *i*. Note that for the spar panels of the cell, the $q_{s,R}$ term involves the difference between the torsional shear flow component of the two cells that the spar panel is common to. The arc length for each panel is computed by breaking up the panel into very small subdivisions so that each subdivision can be approximated as a straight panel. The simple distance formula is used to compute the length of each subdivision, and then these lengths are summed to produce the arc length of the entire panel. Given that the wing section possesses three cells, Equation 4.11 will yield three equations of which there are a total of four unknowns (the torsional shear flows and the twist per unit length).

The fourth equation is obtained by equating the moment produced by the shear loads and the moment produced by the shear flows in all the panels. This moment balance is shown in Equation 4.12 and the moments are taken about the top of the first spar. Here, N represents the total number of panels in the entire wing section while M represents the number of cells.

$$V_z x_{V_z} + V_x z_{V_x} = \sum_{i=1}^{N} 2\bar{A}_i q_{b,i} + \sum_{R=1}^{M} 2A_R q_{s,R} \tag{4.12}$$

x_{V_z} is the x-location of the V_z shear force relative to the top of the first spar, z_{V_x} is the z-location of the V_x shear force relative to the top of the first spar, and $\bar{A}_i$ is the area enclosed by the top of the first spar and the panel in question [4]. The $\bar{A}_i$ term can essentially be interpreted as the area swept by the line connecting the top of the first spar and a point along the panel that traverses the length of the panel. The sweep area is computed by breaking up the panel into very small subdivisions so that each subdivision of the panel can be assumed to be approximately straight. The sweep area over each subdivision can then be reduced to computing the area of a triangle. The lengths of the three sides of the triangle are computed by the simple distance formula, and then the area of the triangle is computed by using Heron's Formula, shown in Equation 4.13. The sweep areas from each subdivision are then summed to produce the entire sweep area of the panel in question.

$$A_{triangle} = \sqrt{s(s-a)(s-b)(s-c)} \tag{4.13}$$

a, *b*, and *c* are the lengths of the three sides of the triangle, and *s* is half of the perimeter of the triangle.

Simultaneously solving this system of four equations will yield the torsional shear flow values for the three cells and the twist angle per unit span induced by the shear loads. The flexural shear flow component and the torsional shear flow component can then be summed for each panel to obtain the total shear flow distribution throughout the wing section.

4.3.3) Pure Torsional Flow

Aerodynamic loads also produce a torsional torque, M_y, to act on the structure. This torque is assumed to be constant throughout the wing span and is computed using Equation 4.14.

$$M_y = \frac{1}{2}\rho v^2 C_{M,0} S c \tag{4.14}$$

where ρ is the air density, *v* is the airplane velocity, *S* is the reference surface area (wing surface area), *c* is the chord length and $C_{M,0}$ is the moment coefficient at the airfoil's aerodynamic center obtained through CFD analysis.

The torque computed this way produces shear stresses in the structure that require a shear flow calculation to estimate them. For shear flow purposes the wing is seen as a three-cell beam. An expression for the twist angle per unit length, $\frac{d\theta_i}{dy}$, can be developed for the generic cell *i*. If the shear flow, *q*, is constant in each cell the twist angle per unit length is given by Equation 4.15 [4].

$$\frac{d\theta_i}{dy} = \frac{q}{2\bar{A}_i G}\oint_i \frac{ds}{t} \tag{4.15}$$

where A_i is the cross sectional area included in the i-th cell, *G* is the material's shear modulus, *q* is the shear flow in the cell and *t* is the skin thickness both functions of the curved coordinate *s*.

Since the structural integrity has to be guaranteed, a condition to the twist angles is imposed: they have to be the same in each cell. This relation in addition to the summation of moments in the cross section provides a set of three equations in three unknowns, the shear flows in the three cells. These are shown in Equation 4.16.

$$M_y = 2\sum_{i=1}^{3} \overline{A_i} q_i$$
$$\frac{d\theta_1}{dy} = \frac{d\theta_2}{dy}$$
$$\frac{d\theta_2}{dy} = \frac{d\theta_3}{dy} \tag{4.16}$$

Solving this system of linear equations leads to the knowledge of the shear flows in the structure and then the shear stresses, τ_{xz}, are easily estimated with Equation 4.17 [4].

$$\tau_{xz_i} = \frac{q_i}{t} \tag{4.17}$$

where t is again the skin thickness at the location to be evaluated.

4.3.4) Shear Center

The shear center (also known as the "center of twist") is the location at which an applied shear load will not induce any rotation in the section. Since the shear center remains fixed for a set section geometry, the shear center allows for an alternative way of computing the total shear flow throughout the wing section. By moving the applied shear load to the shear center and accounting for the additional torque on the section from moving the shear load, the shear flow can then be broken up into two independent shear flow computations: a pure flexural shear flow and a pure torsional shear flow. Both of these can be computed through equations presented in sections 4.3.2 and 4.3.3.

To determine the location of the shear center, the x and z-components of the shear center must be computed separately. To compute the x-location, a shear load in the z-direction V_z of arbitrary non-zero magnitude is applied at an arbitrary location which is assumed to be the shear center. The flexural shear component due to V_z is computed in a similar fashion to that described in section 4.3.2. The torsional shear flow component is also computed similarly to that shown in section 4.3.2, although the twist angle per unit length is now deemed to be zero (since the shear load is acting at the shear center). Thus, the three equations yielded by Equation 4.11 are sufficient to solve for the torsional shear flow components in the three cells. Then, the flexural and torsional shear flows are plugged into Equation 4.12 to solve for the x-location of the shear center which would be the x_{Vz} term. The z-location is computed similarly except that an arbitrary shear load in the x-direction V_x is applied at the shear center instead and z_{Vx} is now the variable of interest in Equation 4.12.

Since computing the shear center requires the calculation of the shear flows twice (for the x and z-locations), it was deemed to be a less efficient and more time-consuming approach to computing the total shear flow in the section. In consideration of the Monte Carlo iteration process, a low computation time is important to allow for a larger number of trials to be tested. Hence, the method outlined in sections 4.3.2 and 4.3.3 is what is implemented for the present wing design in the computation of the shear stresses. Shear center will be computed after optimization is performed in order to compute parameters such as divergence velocity.

4.4) Von Mises Stresses

Once all the components of the stresses are evaluated they have to be related to the material's yield. This can be done with an appropriate yield criterion like Von Mises' stress for metals. The Von Mises equation gives the value of the normal stress, σ_{VM}, equivalent to the stress tensor considered. In generic Cartesian coordinates (not the structure's principal directions) the expression is shown in Equation 4.18 [4].

$$\sigma_{VM} = \sqrt{\frac{(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_x - \sigma_z)^2 + 6(\tau_{xy}{}^2 + \tau_{yz}{}^2 + \tau_{xz}{}^2)}{2}} \tag{4.18}$$

where σ are normal stresses and τ are shear stresses in the directions shown by the subscripts. In the case analyzed the expression simplifies to Equation 4.19.

$$\sigma_{VM} = \sqrt{\sigma_y{}^2 + 3\tau_{xz}{}^2} \tag{4.19}$$

5) Deformation Distributions

5.1) Deflection Distribution

The wing under the effect of the bending moments deflects and its shape modifies. In order to check this deformation a deflection distribution calculation is performed. The bending moments, M_x and M_z, distribution in the 'y' direction are a function of the second derivative of displacements u and w in the 'x' and 'z' direction respectively. The relationships between these variables are shown in Eqn 5.1 [3].

$$\begin{bmatrix} \frac{\partial^2 u}{\partial y^2} \\ \frac{\partial^2 w}{\partial y^2} \end{bmatrix} = -\frac{1}{E(I_x I_y - I_{xz}{}^2)} \begin{bmatrix} -I_{xz} & I_x \\ I_z & -I_{xz} \end{bmatrix} \begin{Bmatrix} M_x(y) \\ M_z(y) \end{Bmatrix} \tag{5.1}$$

where E is the material's Young's modulus. Numerically integrating twice gives the deflection distribution along the wing span expressed as section's displacements. The boundary conditions are shown in Equation 5.2.

$$\text{if} \quad y = 0 \qquad u(0) = \frac{\partial u(0)}{\partial y} = 0 \quad \text{and} \quad w(0) = \frac{\partial w(0)}{\partial y} = 0 \tag{5.2}$$

The maximum deflection at the wing tip is approximately 0.4-0.8 ft in the 'z' direction and 0.02-0.05 ft in the 'x' direction.

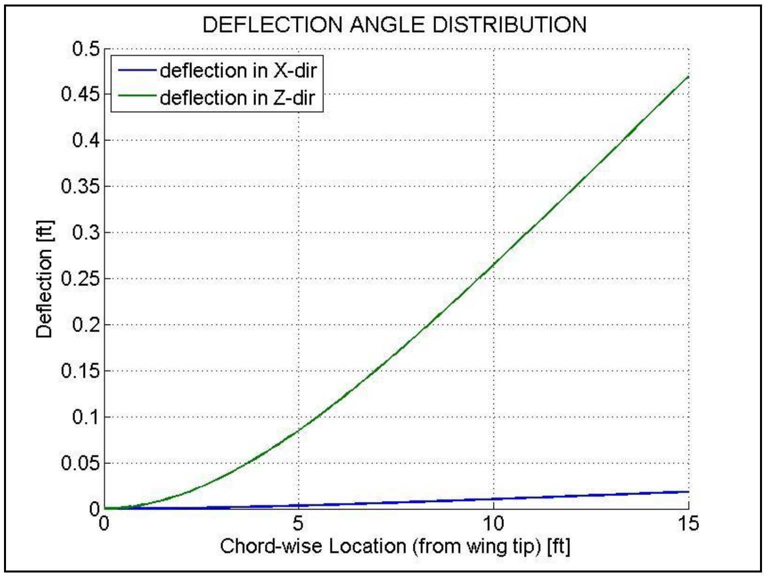


Figure 5.1: Absolute wing deflection over the half span, proceeding from tip to root

5.2) Angle of Twist Distribution

In sections 4.3.2 and 4.3.3, the wing twist angle per unit spanwise distance was computed from the torsional shear load due to both the aerodynamic loads (from Equation 4.5) and the airfoil moment about the aerodynamic center (from Equation 4.9). The wing twist due to the airfoil moment about the aerodynamic center, which is only a function of the airfoil selected, is assumed to be constant throughout the spanwise length. The wing twist due to the aerodynamic loads, however, depends on the magnitude of the shear load, which varies along the span. Hence, the wing twist per unit length was computed in 0.1 ft increments along the span (totaling to 150 subdivisions in the semispan). Numerically integrating this distribution while imposing the boundary condition of zero twist angle at the wing root, the angle of twist distribution is computed and shown in Figure 5.2.

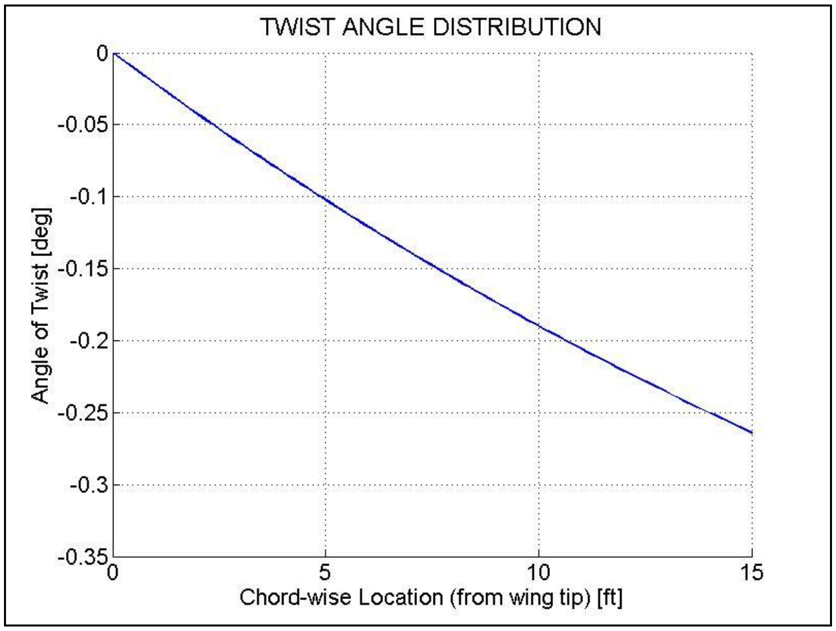


Figure 5.2: Angle of twist distribution along half span, proceeding from wing tip to root

6) Failure Criteria

6.1) Von Mises Failure Criteria

Using the Von Mises stress computed in Section 4.4, the yield and fracture verification can be done using the Factor of Safety (FOS, 1.25 for yield and 1.5 for fracture) and that shown in Equation 6.1.

$$\sigma_{VM} \leq \frac{\sigma_{yield}}{FOS_{yield}} \qquad \sigma_{VM} \leq \frac{\sigma_u}{FOS_u} \tag{6.1}$$

where σ_{yield} and σ_u are the material's yield and ultimate stresses respectively.

6.2) Buckling Analysis & Calculations

Buckling refers to a variety of failure modes wherein a flat plate or column fails at stresses that are less than the material yield stress under conditions of compressive or shear loading. In an aircraft wing, the skin is prone to buckling which tends to impair the aerodynamic properties of the airfoil. Hence stringers are placed strategically within the wing in order to minimize the risk of skin buckling. However, the stringers themselves are prone to buckling, which leads to the inclusion of ribs spaced throughout the wing structure, which stiffen the stringers by reducing their length. Therefore buckling is almost purely a function of wing geometry and material properties. The analysis presented in this section deals with the buckling of wing sections due to both compressive as well as shear loads and puts forward the necessary calculations in order to calculate critical stresses which, if surpassed, result in buckling. This information was then used to develop the criteria for a successful wing design within the optimization procedure.

6.2.1) Compressive Column Buckling

As the wing structure bends due to the aerodynamic forces, compressive loads develop on surfaces (generally this loading is complex, but in a simplified sense the top of the wing is in compression and the bottom in tension). Within the wing, the structures that are most prone to column buckling are the skin and stringers. Since both the skin and the stringers carry this compressive load, they were considered together for column buckling calculations. Skin panels are divided between stringers in the chordwise direction and ribs in the spanwise direction. The geometry of the stringer/skin combination is shown in Figure 1. In it, each stringer was considered individually along with the skin whose chordwise length was determined by half of the distance to the nearest adjacent stringer (or spar cap). For convenience, this analysis is performed in a different coordinate system local to each stringer: the 'x' axis is parallel to the plate connected to the stringer, while 'z' is now running along the span and 'y' is perpendicular to the first two, as shown in Figure 6.1.

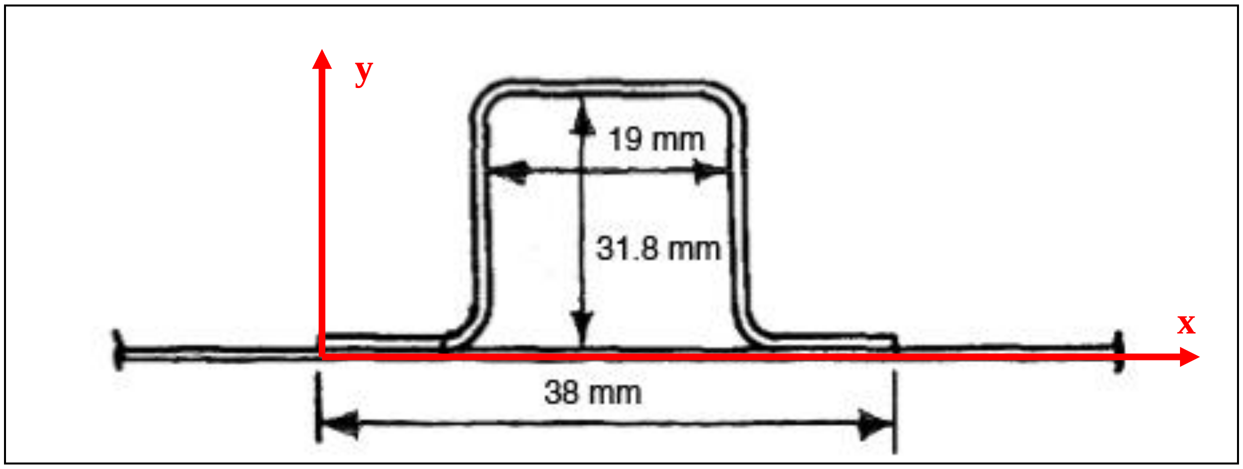


Figure 6.1: Stringer Cross Section and local coordinate system [5]

In order to calculate the column buckling stress, the centroid of the stringer/skin geometry was first calculated, given by Equations 6.2 and 6.3.

$$x_c = \frac{\sum_{i=1}^{7} x_i A_i}{\sum_{i=1}^{7} A_i} \tag{6.2}$$

$$y_c = \frac{\sum_{i=1}^{7} y_i A_i}{\sum_{i=1}^{7} A_i} \tag{6.3}$$

Where x_c and y_c represent the x and y-coordinates of the total centroid, x_i and y_i represent the centroid of each of the seven flat plate components that make up the geometry, and A_i represents the cross-sectional area of each of these flat plate components. Next the area moment of inertia about the local centroid was calculated for each stringer section. This process was simplified by again assuming that the local stringer/skin cross section can be broken into seven distinct flat plates (2 contributions from the skin sections and 5 faces of the stringer). Then the moment of inertia of each of these plates was calculated individually about its local centroid (Equation 6.4) and then moved to the section centroid via the Parallel Axis Theorem (Equation 6.5) and finally all seven moments of inertia were summed linearly to find the area moment of inertia of the whole wing/skin cross-section.

$$I_{i,x'} = \frac{1}{12} b \cdot h^3 \tag{6.4}$$

Where $I_{i,x'}$ is the local rectangular plate area moment of inertia, b is the width of the rectangle along the axis the moment of inertia is taken about, and h is the height of the plate perpendicular to the axis of the area moment of inertia.

$$I_x = I_{i,x'} + A_i y^2 \tag{6.5}$$

Where I_x is the area moment of inertia about the x-axis passing through the stringer/skin cross-sectional centroid, $I_{i,x'}$ is the area moment of inertia of the individual component and A_i is the area of the component, while y is the distance in the y-direction between the component and structure centroids [6].

Once the area moment of inertia is known, it can be combined with material properties and the prescribed rib spacing to calculate the critical pressure that can be applied to the structure. Equation 6.6 gives this calculation [3].

$$P_{CR} = \frac{\pi^2 E I_x}{L_{rib}^2} \tag{6.6}$$

Here P_{CR} is the critical pressure, E is the Young's modulus, I_x is the area moment of inertia, and L_{rib} is the rib spacing. The critical pressure can be converted into a stress using Equation 6.7.

$$\sigma_{CR,section} = \frac{P_{CR}}{A_{tot}} \tag{6.7}$$

Where $\sigma_{CR,section}$ is the critical stress and A_{tot} represents the combined cross-sectional area of the stringer and local skin sections.

6.2.2) Compressive Plate Buckling

In addition to checking that the stringer/skin section will not fail from compressive loads, it is necessary to check that the individual plates that make up the stringer will not buckle. For this calculation, it was assumed that each plate was clamped on all four sides. The critical buckling load was then computed using Equation 6.8 [3].

$$N_{CR} = \frac{k\pi^2 E t^3}{12b^2\left(1-\nu^2\right)} \tag{6.8}$$

Where k is the buckling coefficient, E is the Young's modulus, t is the plate thickness, b is the plate length (in cross section), and υ is the Poisson ratio. The buckling coefficient k comes experimentally from section geometry and end conditions and was programmed approximately as a piecewise function. The complete buckling coefficient for compressive loading is shown in Figure 6.2. With the critical load for each section calculated, the total compressive plate buckling stress was then calculated using Equation 6.9 [4].

$$\sigma_{CR,comp} = \frac{\sum_{i=1}^{5} N_{CR,i}}{A_{stringer}} \tag{6.9}$$

Where $\sigma_{CR,comp}$ is the total compressive plate buckling stress allowable, $N_{CR,i}$ is the compressive load allowed on the individual stringer sections, and $A_{stringer}$ is the total cross-sectional area of the stringer. At each section, the critical stress due to axial loads to be considered is the lesser of the column buckling critical stress and the plate compressive critical stress.

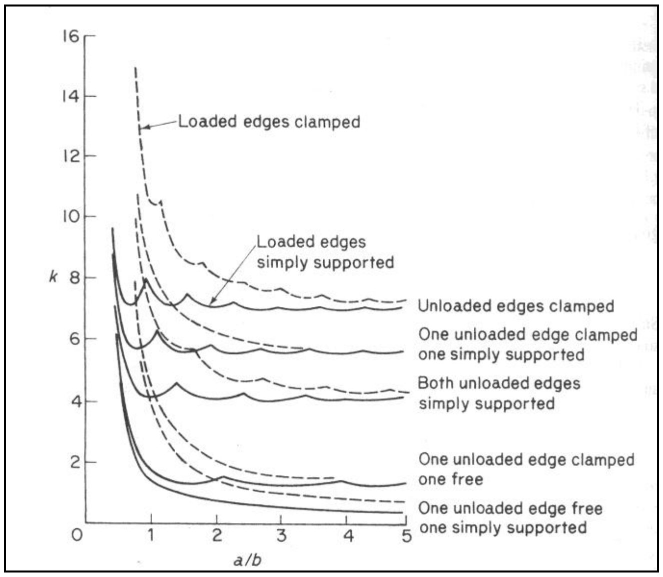


Figure 6.2: Plate buckling coefficients for compressive loading [3]

6.2.3) Plate Shear Buckling

The final buckling problem considered was that of plate shear buckling on the wing's skin. This calculation again uses Equation 6.8, however in this case the buckling coefficient *k* is altered to reflect plate shear buckling [3]. Again, this coefficient is calculated from experimental data and was programmed to be evaluated approximately as a piecewise function. The actual plate buckling coefficients for shear loading are shown in Figure 6.3. For each panel of skin in the wing section, the buckling coefficient and then the buckling stress were calculated. If the calculated shear stresses in the skin section exceeded the buckling shear stress, it would result in failure, and hence was considered unacceptable for any proposed wing geometry.

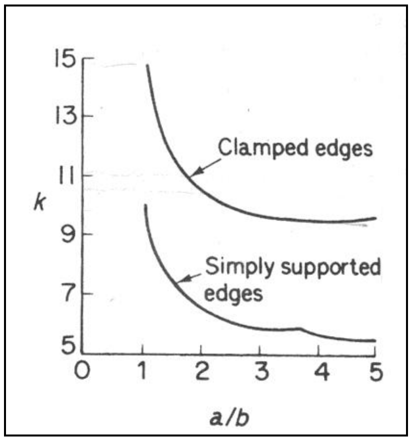


Figure 6.3: Plate buckling coefficients for shear loading [3]

6.3) Fracture

Fracture involves the local separation of a material into two or more pieces due to the action of an applied stress. In ductile materials such as aluminum, stresses exceeding the yield stress induce extensive plastic deformation, resulting in the formation of a crack and eventually fracture. Fracture may also occur below the yield stress if a crack is present in the material. The geometry of a crack induces local stress concentrations near the crack tip, which can be quantified by the stress intensity factor K_I (Equation 6.10) [4].

$$K_I = 1.2\sigma^{\infty}\sqrt{\pi a} \tag{6.10}$$

Where σ^{∞} is the applied tensile stress (assumed to be perpendicular to the crack) and a is the crack length (assuming an edge crack).

The critical stress intensity factor K_{IC} is an inherent property of a particular material and represents the threshold at which fracture occurs. Should the K_I induced from a certain crack size exceed this critical value, unconstrained crack growth occurs, causing failure. The critical crack size a_{CR} at which this unrestrained crack growth occurs is computed using Equation 6.11, which is obtained from Equation 6.10 where K_I is replaced with K_{IC} [4].

$$a_{CR} = \frac{1}{\pi}\left(\frac{K_{IC}}{1.2\sigma^{\infty}}\right)^2 \tag{6.11}$$

6.4) Fatigue

Fatigue is the localized and progressive structural damage due to applied cyclic loading. Should a crack be present in a material, sufficient tensile loads will cause crack growth. When the crack length exceeds the critical crack size computed using Equation 6.11, the localized stress concentration near the crack tip induces unconstrained crack growth resulting in fracture. Using Paris' law and assuming a loading cycle from zero to the maximum tensile stress (since compressive forces do not contribute to crack growth), the number of loading cycles *N* till failure is computed using Equation 6.12 [4].

$$N = \frac{a_{CR}^{\;1-m/2} - a_0^{\;1-m/2}}{C(1-m/2)(\sigma_{\max}\pi)^2} \tag{6.12}$$

Where a_0 is the initial crack size, $\sigma_{\max}$ is the maximum tensile stress (from the panel that produces the largest positive bending stress as computed in Section 4.2), and C and m are both constants inherent to each material.

7) Aeroelasticity

7.1) Wing Divergence

If the aerodynamic center of the wing is located forward of the shear center, the aerodynamic loads (i.e. lift and drag) will apply a pitch-up torsional moment on the wing. The higher the aircraft velocity, the higher this applied moment. When the aircraft hits a certain critical velocity where the torsional rigidity of the wing is no longer able to withstand this moment, the wing will twist uncontrollably causing failure. This occurs because as the wing twists in a pitch-up fashion, the angle of attack effectively increases, which increases the loads and thus the moment, causing more wing twist and repeating the cycle until failure. A similar phenomenon occurs if the aerodynamic center is located behind the shear center, though wing twist will occur in the opposite direction. This positive feedback situation that causes the wing twist to diverge is termed "wing divergence," and the critical velocity at which this occurs is the divergence velocity. To ensure that the wing does not fail through divergence, the wing must be designed to ensure that the divergence velocity exceeds the maximum velocity of flight. In the case at hand, the maximum flight velocity is the dive velocity [3].

The total torsional moment *T* applied on the wing is the sum of the airfoil moment about the aerodynamic center and the torque applied by the lift and drag forces about the shear center. This is shown in Equation 7.1, where the shear center is assumed to be to the upper right of the aerodynamic center and the moment is assumed to be positive counterclockwise.

$$T = -M_{ac} + F_z(x_{sc} - x_{ac}) - F_x(z_{sc} - z_{ac}) \tag{7.1}$$

Where M_{ac} is the moment about the aerodynamic center, F_z and F_x are the net force in the z-direction and x-direction respectively, x_{sc} and z_{sc} are the x and z locations of the shear center (computed in Section 4.3.4), and x_{ac} and z_{ac} are the x and z locations of the aerodynamic center. Knowing the total torque and the wing twist per unit spanwise length on the wing $\frac{d\theta}{dy}$ computed in section 4.3.2 and 4.3.3, the torsional rigidity GJ of the wing can be computed from Equation 7.2.

$$GJ = \frac{T}{\left(\frac{d\theta}{dy}\right)} \tag{7.2}$$

Finally, the divergence velocity of the wing is computed using Equation 7.3.

$$V_{Divergence} = \sqrt{\frac{\pi^2 GJ}{2a_w c^2 e\rho\left(\frac{b}{2}\right)^2}} \tag{7.3}$$

Where a_w is the 3-D lift curve slope, c is the chord, e is the distance between the shear center and aerodynamic center in percent chord, ρ is the air density, and $b/2$ is the semispan.

All these terms are evaluated at the root of the wing at the PHAA sea level loading scenario since this is where the highest aerodynamic loads are obtained.

7.2) Aileron Reversal

Aileron reversal refers to a critical and some times dangerous phenomenon in flight stability and control. It can affect all the major aerodynamic surfaces and here the wing example is presented for simplicity. This event is a consequence of the wing torsional elasticity: the aileron deflection triggers a torsional moment in the wing structure that leads to twisting of the wing. If the aileron moves downward a pitch down deformation of the wing takes place producing a reduction in the wing incidence angle and, consequently, in the wing lift. The reduced lift harms the rolling moment which is a function of the square of the airflow velocity. Also, the structure's torsional stiffness is constant for a constant geometry. The wing twist due to aileron reversal therefore increases with increasing speed. A critical velocity can be defined as the *aileron reversal speed* and it is the value at which an aileron deflection does not cause any rolling moment. If this velocity is exceeded the aileron behavior is opposite to what is expected, leading to a very dangerous and unpredictable aircraft dynamics [3].

From a design point of view, this phenomenon can be avoided by increasing the wing's rigidity. This can be achieved with a proper distribution of the areas in the wing's cross section. The analytical relations that mathematically describe the problem are non trivial and become complex if the complete finite wing is considered. Given the maximum speed in the airplane's flight envelope it is possible to compute the consequent torque in the wing structure and compare it to the torsional stiffness [3]. Since this analysis is beyond the scope of this project, no calculations will be presented in this report.

7.3) Flutter

"Flutter" is an aero-elastic phenomenon where the aerodynamic forces couple with the natural mode of vibration of an elastic body, causing rapid periodic motion. An aircraft wing has flexural and torsional modes of vibration. According to Megson [3], an accurate flutter analysis requires coupling of more than one of these modes of vibration, because single degree of freedom systems cannot go unstable. In aircraft wings, a form of flutter that is commonly taken into consideration involves the coupling of the bending and torsional modes of the wing. Here, a twist on the wing causes a geometric incidence with the airflow and thus an aerodynamic load, inducing a bending motion on the wing. As the twist on the wing reverses, an aerodynamic load is produced in the opposite direction inducing bending motion of the wing in this opposite direction. This cycle repeats, thus causing coupled torsional and flexural oscillatory motion. Problems arise at a critical flight velocity called the "flutter speed," which is the lowest flight speed at which the structure will oscillate with sustained simple harmonic motion. Below this velocity, the damping is sufficient to maintain stable structural oscillation. Exceeding this critical speed, however, will cause divergent oscillation resulting in failure [3].

Two approaches will be discussed in terms of flutter calculations. The first approach involves a considerably simplified model of the wing as a cantilever beam. Note that this analysis only takes into consideration the bending mode of vibration in the z-direction, and hence may not be accurate since single degree of freedom systems cannot go unstable, as mentioned earlier. By computing the maximum deflection of the wing $\delta_{\max}$ (as described in Section 5.1) and considering the total force on the half-span in the z-direction $F_{b/2}$, the effective spring constant k_{eff} of the beam can be computed using Equation 7.4.

$$k_{eff} = \frac{F_{b/2}}{\delta_{\max}} \tag{7.4}$$

Next, the natural frequency f_n of the wing in Hz can be computed using Equation 7.5, which is based on this effective spring constant and the mass of a half-span of the wing m_{wing}.

$$f_n = \frac{1}{2\pi}\sqrt{\frac{k_{eff}}{m_{wing}}} \tag{7.5}$$

An alternate approach involves a more lengthy and comprehensive analysis of the dynamic aspects of the wing and aerodynamic loads. While such an analysis is beyond the scope of this project, a procedure will be briefly discussed that will enable the computation of the critical flutter velocity. Rewriting the lift force and pitch moment equations to account for contributions from the oscillatory motion of the wing and ensuring a moment equilibrium about the aerodynamic center and vertical force equilibrium, Equations 7.6 and 7.7 are obtained [3].

$$(m - L_{\ddot{y}})\ddot{y} - L_{\dot{y}}\dot{y} + (k - L_y)y - (mgc + L_{\ddot{\alpha}})\ddot{\alpha} - L_{\dot{\alpha}}\dot{\alpha} - L_{\alpha}\alpha = 0 \tag{7.6}$$

$$(-mgc - M_{\ddot{y}})\ddot{y} - M_{\dot{y}}\dot{y} + M_y y - (I_O + M_{\ddot{\alpha}})\ddot{\alpha} - M_{\dot{\alpha}}\dot{\alpha} - (k_{\theta} - M_{\alpha})\alpha = 0 \tag{7.7}$$

Where *m* is the mass of the wing half-span, *g* is the distance between the center of gravity and the shear center in percent chord, *c* is the chord, I_O is the moment of inertia about the aerodynamic center, *k* and k_θ are the flexural and torsional stiffnesses of the wing respectively, *y* is in the direction of the lift force, *α* is the angle of attack, and the subscripted terms for the Lift "*L*" and Moment "*M*" indicate derivatives with respect to those variables in the subscripts. The subscripted terms are all functions of velocity and are computed through more rigorous analytical approaches or via experiment. Assuming oscillatory motion of the system as described by Equation 7.8, the critical flutter speed V_f can be computed numerically by determining the velocity at which *δ* is zero, hence representing the point at which stable oscillations transition to unstable conditions [3].

$$y = y_0 e^{(\delta + i\omega)t} \qquad \alpha = \alpha_0 e^{(\delta + i\omega)t} \tag{7.8}$$

Where y_0 and α_0 are arbitrary coefficients, *δ* is the exponential growth rate, *ω* is the frequency of oscillation, and *t* is time.

Again, this analysis is beyond the scope of this project and hence no calculations will be obtained using this approach in this report. However, such an analysis is essential before reaching production stages of the wing, and hence must be performed in the future to compute the critical flutter speed of the wing. Once this is accomplished, one only needs to ensure that the flight speed does not exceed this critical speed in order to prevent disastrous flutter of the wing.

8) Monte Carlo Optimization

Up to this point, material presented in this report has dealt largely with structural analysis required to judge whether a wing design is acceptable or not. However, this analysis is merely the means by which any given design is interpreted in order to generate understanding of its performance. This section presents how these analyses were used in order to develop an optimal design. Many different optimization algorithms exist, however perhaps the simplest is known as the Monte Carlo algorithm, which was implemented in this work. In this optimization the wing geometry is selected randomly and then evaluated for its performance. Those wings with satisfactory performance were then judged based on a cost function, which in this case was based purely on the weight of the proposed wing design. By testing thousands of random designs, this method reveals not only the designs that are most often successful, but also those that will provide satisfactory performance with the lowest weight.

8.1) Randomized Wing Geometry

Although the final wing design will use standardized parts, for simplicity of implementation in Matlab it was chosen instead to use randomized inputs for the Monte Carlo algorithm and then later switch to the standard part dimensions that most closely match the optimal random design. Except for the wing shape (NACA 2412), virtually the entirety of the component arrangement in the wing design is determined randomly. For each degree of freedom in the design, a maximum and minimum range were selected for that parameter and a Matlab script was written that randomly chooses a wing design based on the predefined ranges. For the optimization provided here, the range provided was generally wide in order to never dismiss potentially satisfactory designs by leaving them outside the possibilities of the program. The wing is a three-cell, two-spar design, with a front spar placement ranging from 0.1 to 0.3 of the chord length and the rear spar from 0.4 to 0.65 of the chord length. The spar caps at the ends of the spars range from 250 to 350 mm^2 (0.3875 to 0.5425 in^2) in their surface area. The stringers, which act to stiffen the skin and carry some bending load, range in cross-section area from 50 to 150 mm^2 (0.0775 to 0.2325 in^2). Spar thickness ranges from 0.04 to 0.08 inches and skin thickness ranges from 0.03 to 0.06 inches. Another script was written for the placement of the stringers. Their number ranges from 4 to 14 with 1 to

3 along each of the top and bottom sections of the first cell, 1 to 3 along each of the top and bottom sections of the second cell, and 0 to 2 along the top and bottom sections of the third cell. The script also ensures that stringers are not placed too closely to each other or the spar caps.

8.2) Analysis Procedure

A series of Matlab scripts were written in order to perform the necessary analysis on each of these randomized designs. As shown in Figure 8.1, after the wing section geometry was chosen, the first step was to calculate the centroid of the wing section and the area moment of inertia about the centroid, using the calculations presented in Section 4.1. Next the shear flows, torsional flows, shear stresses, bending stresses, twist, and deflections were calculated. In a parallel process buckling stresses were also considered. The stresses from applied loads were then compared to the material properties and the bending stresses to ensure that the proposed design satisfied safety requirements. Those designs that provided satisfactory safety were then passed to another script to calculate an approximate weight of the wing design.

Since the primary criteria for judging the wings is the minimization of weight, this gave direct results for judging the effectiveness of each design. Regardless of the outcome, this process was repeated over 30,000 times each for aluminums 2024-T3 and 7475-T6, allowing us to develop the optimized wing structure through force of computation as we consider thousands of possible wing designs.

8.3) Optimization Results

Once the Monte Carlo optimization was run for a sufficient number of trials, the results were interpreted using a series of Matlab scripts written to plot and parse in order to differentiate between the satisfactory designs presented by the optimization. For both materials studied the pass rate for wing designs was approximately 8.5%. This low pass rate is desirable because it means that the wing designs proposed by the program exist mostly on the cusp between wings that meet structural requirements and those that do not. Since the lightest wings generally have designs that lie close to the bottom limits in their structural capabilities, the low pass rate means that most of the designs examined lie close to this transition between success and failure.

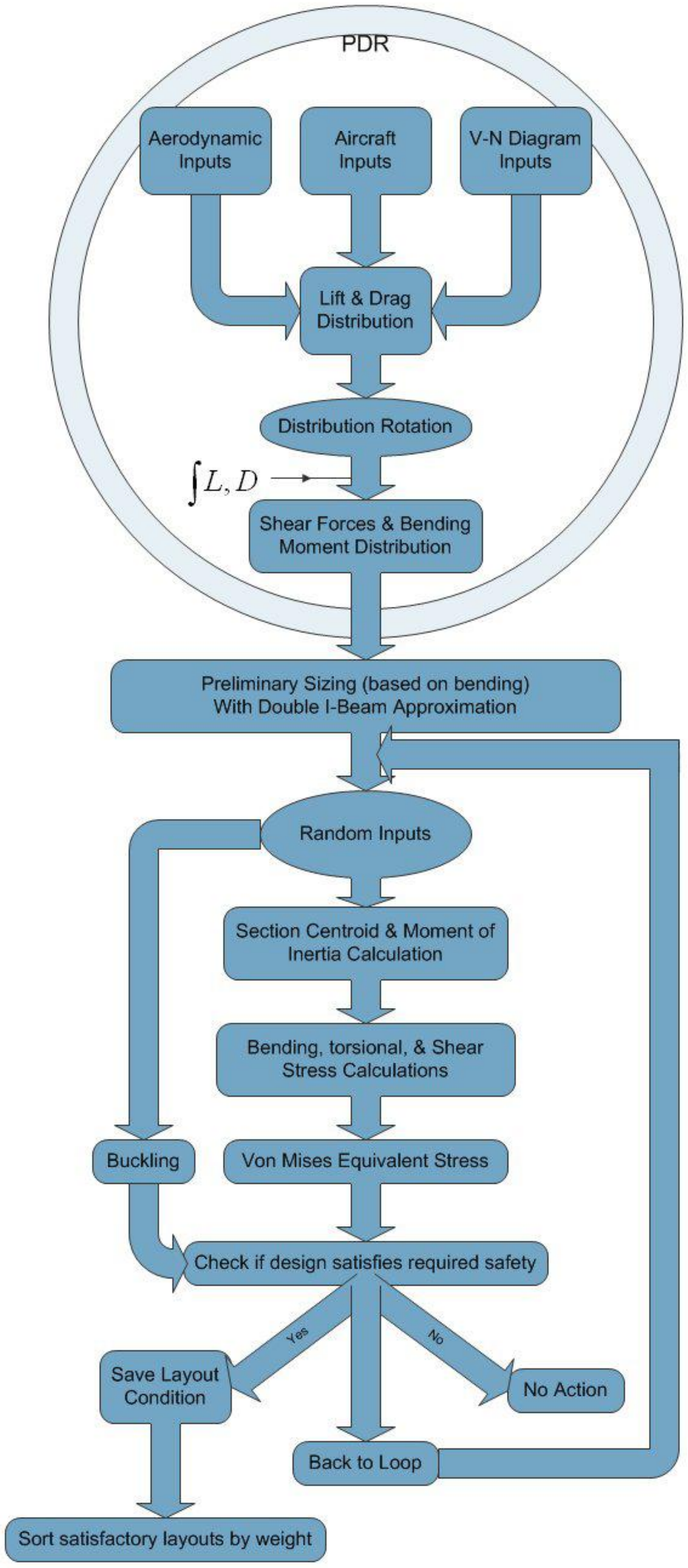

PDR
Aerodynamic Inputs
Aircraft Inputs
V-N Diagram Inputs
Lift & Drag Distribution
Distribution Rotation
$\int L, D$
Shear Forces & Bending Moment Distribution
Preliminary Sizing (based on bending) With Double I-Beam Approximation
Random Inputs
Section Centroid & Moment of Inertia Calculation
Bending, torsional, & Shear Stress Calculations
Buckling
Von Mises Equivalent Stress
Check if design satisfies required safety
Yes
No
Save Layout Condition
No Action
Back to Loop
Sort satisfactory layouts by weight

Figure 8.1: Monte Carlo Optimization Algorithm

Since the cost function used in the optimization is based on weight alone, a histogram showing the weights of the different proposed designs is somewhat useful in understanding how range of successful trials as a whole compare to each other. Figure 8.2 shows this type of plot, from the results of 30,000 trials of aluminum 7475-T6. The weight distribution of successful designs is roughly Gaussian, although our interest lies only with the bottommost of the range, as these are the lightest designs. Several scatter plots were also generated which show successful trials as a function of different input parameters. For each of these plots a correlation coefficient was also calculated to better understand the dependency of weight on each parameter for successful trial. Plots for all eight parameters examined are shown in Appendix B. Figure 8.3 shows the scatter plot of spar cap spacing (the distance between the first and second spars) against half-span wing weight. This plot shows no apparent trend (confirmed by a correlation coefficient of 0.093758). Because of this we can conclude that there is no way to find an optimal design based on the spar cap spacing. On the other hand, Figure 8.4 shows the skin thickness of successful designs against the half-span weight. This plot reveals a distinct correlation between the two parameters. The correlation coefficient for this plot of 0.57776 indicates a moderately strong connection between parameters despite the fact that many other parameters in the wing geometry were also being modified for the trials shown. Figure 8.4 is also interesting because it shows how there is a lower density of successful trials for thinner skin thicknesses. This is likely due to high shear stresses in thin skin sections, which leads to shear buckling. The result is that many of the lightest designs are difficult to attain because they require a careful placement of the wing geometry between the load carrying members in order to prevent failure.

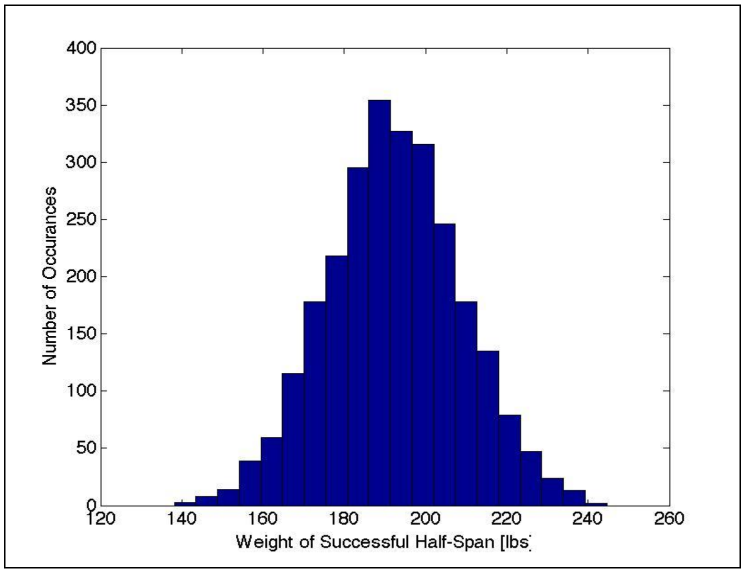


Figure 8.2: Histogram of successful wing designs output by the Monte Carlo optimization against their respective weight ranges. These results come from 30,000 trials of aluminum 7475-T6

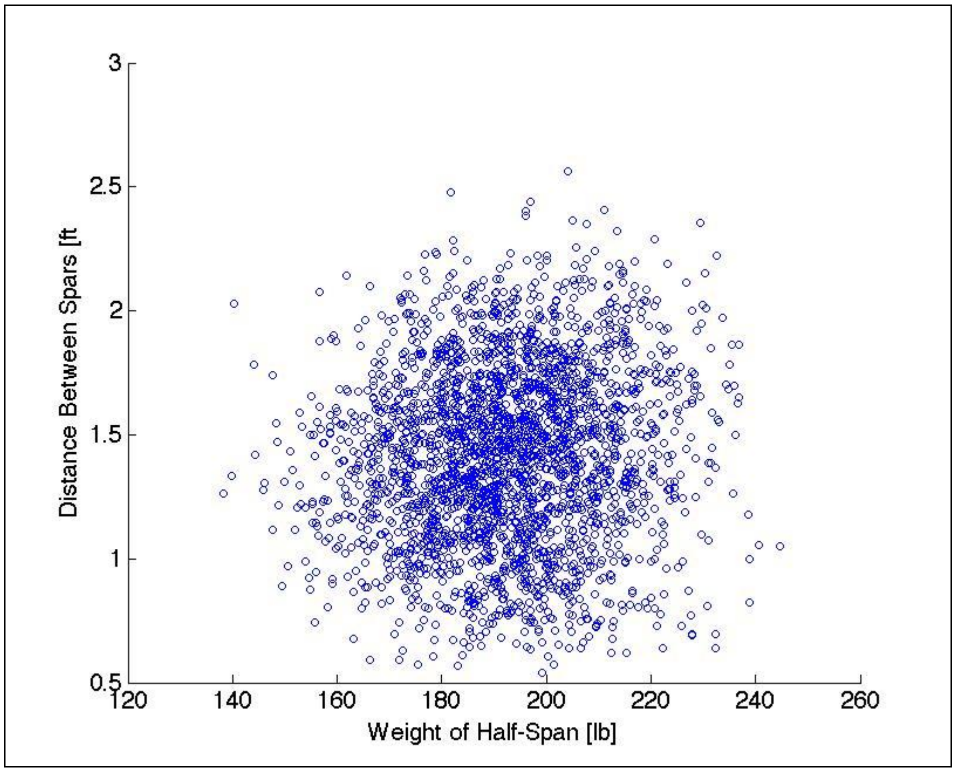


Figure 8.3: Distance between front and rear spar against the wing weight for successful half-span designs

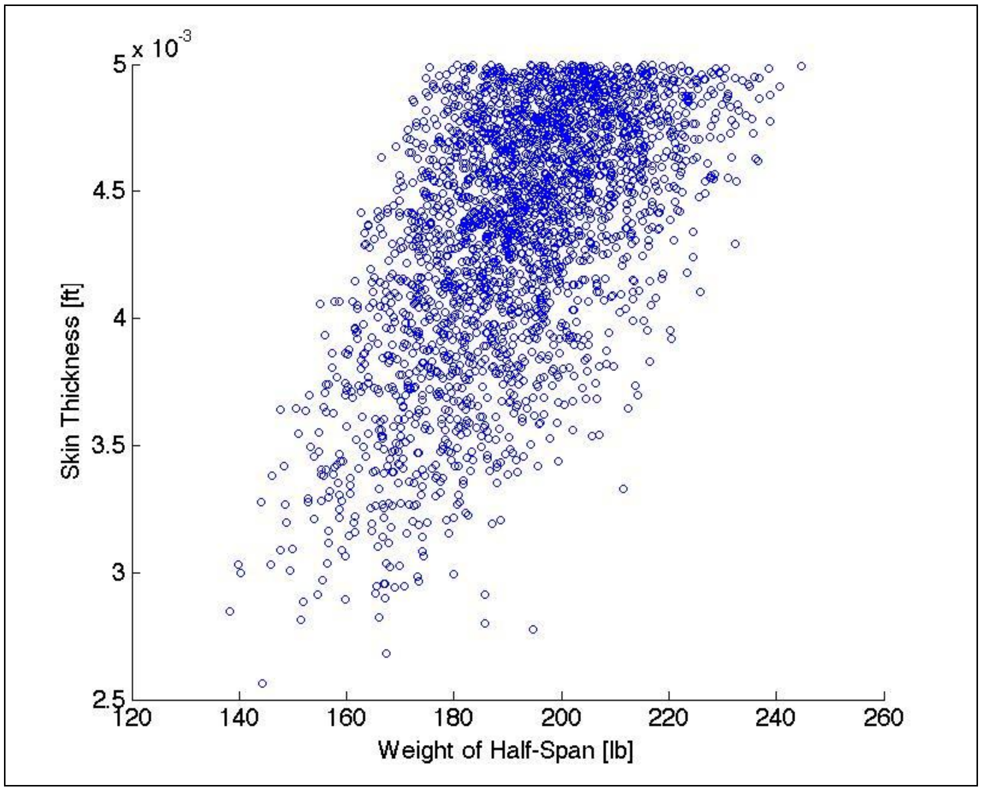


Figure 8.4: Skin thickness against successful half-span weights

8.4) A Study of Geometry, Inertia, and Stress Correlations

As part of the post processing package developed from the Monte Carlo optimization, the lowest weight geometry was used as a baseline wing design. One at a time, different geometrical parameters were altered from the baseline case and the resulting changes in moments of inertia and stresses were plotted. For each parameter, 100 trials were completed using randomly perturbed geometries. By altering one parameter at a time, it is possible to generate a much more precise understanding of how individual parameters affect the overall performance of the wing. For instance, Figure 8.5 shows the area moment of inertia about the z-axis as a function of front spar location while the rear spar stays fixed. Because all areas in the wing cross-section remain constant throughout the trials, it is evident that the minimum in this graph must be a result of front spar passing through a minimum distance to the wing centroid, which as a result minimizes the parallel axis theorem contribution to moment of inertia. It is desirable to avoid this in the design since it will result in a relative maximum stress, as evidenced by Equation 4.7 (stress is inversely proportional to the moment of inertia).

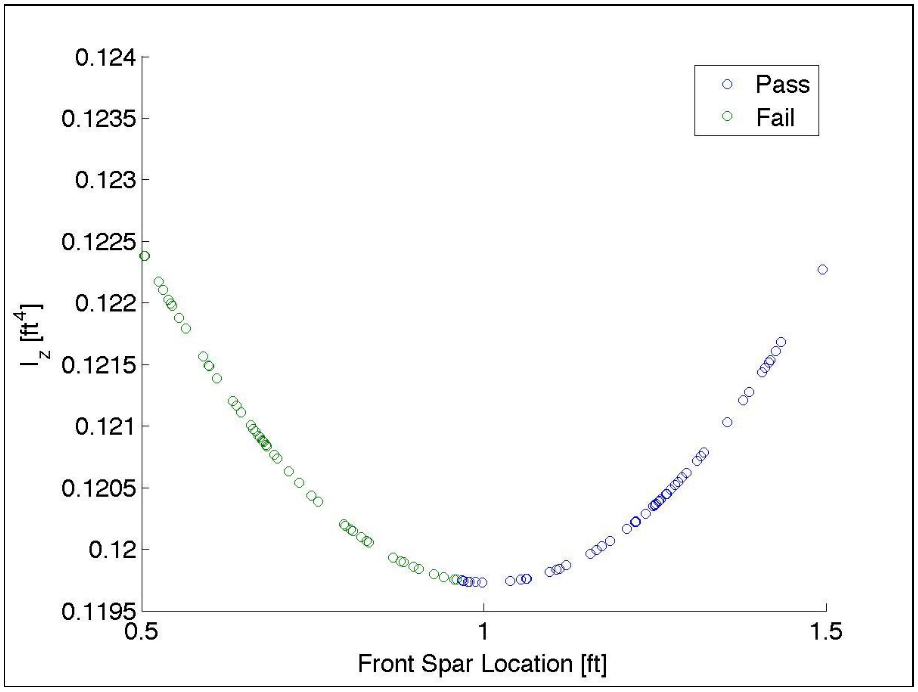


Figure 8.5: Area moment of inertia about the z-axis as a function of front spar location. Blue dots correspond to wing designs that pass failure criteria, while green dots fail.

Another interesting plot was generated from altering the stringer cross sectional area and examining the resulting shear stress, shown in Figure 8.6. In this figure the maximum shear stress appears to shift from one panel to another as the stringer area passes roughly *0.001 ft*^{2}. The trend of exponentially decreasing shear stress on the left of the graph makes sense, however, the minimum followed by ascending shear stress is somewhat puzzling. This behavior is somewhat counterintuitive because it is expected that as the stringer area increases, the moment of inertia will increase (this was in fact observed for this case, see Appendix C) and along with it so will the shear stresses.

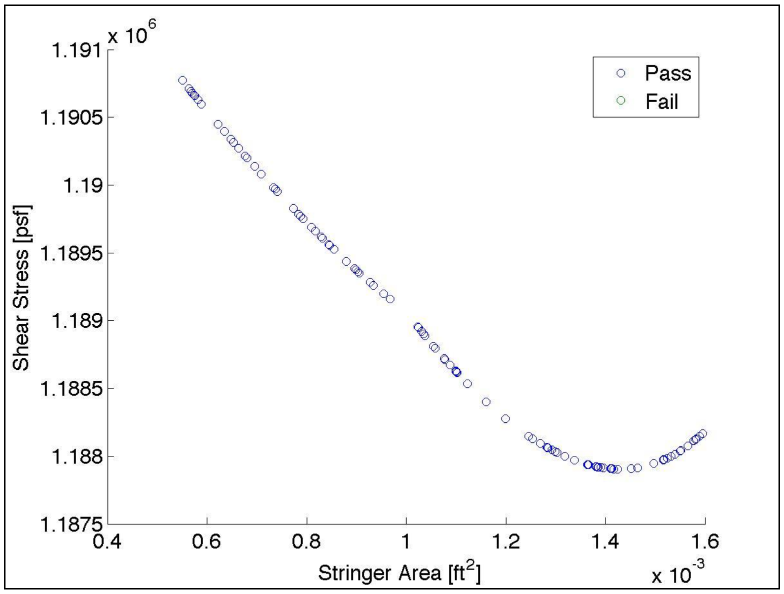


Figure 8.6: Maximum shear stress as a function of stringer cross-sectional area

A simpler plot is obtained from the bending stresses obtained while altering the skin thickness, shown in Figure 8.7. The correlation is nearly perfectly linear, showing how the maximum bending stress decreases as the skin is thickened and is able to carry a larger bending load. It is also important to note that there exists a threshold near the skin thickness of *0.00275 ft* (*0.033 in*) under which the wing does not pass due to structural failure (likely due to shear since the skin primarily handles shearing stresses) and above which it passes.

For brevity, only one more case will be examined here. Plots showing the moments of inertia and stresses for each of the seven parameters which were altered are shown in Appendix C. For this final case the maximum Von Mises stress as a function of stringer cross-sectional area will be examined. The plot of this is shown in Figure 8.8. This plot shows an interesting discontinuity in the derivative of Von Mises stress with respect to stringer area. This is due to the fact that as the stringer area decreases the stresses in each panel decrease at different rates depending on their local geometry. Therefore this discontinuity is a result of the Von Mises stress starting out very large in one panel and rapidly descending, while in some other panel the stress does not start out as high, but because its decay rate as the stringer area increases is considerably less than the first panel, eventually it contributes the largest bending stress and is therefore shown in the plot.

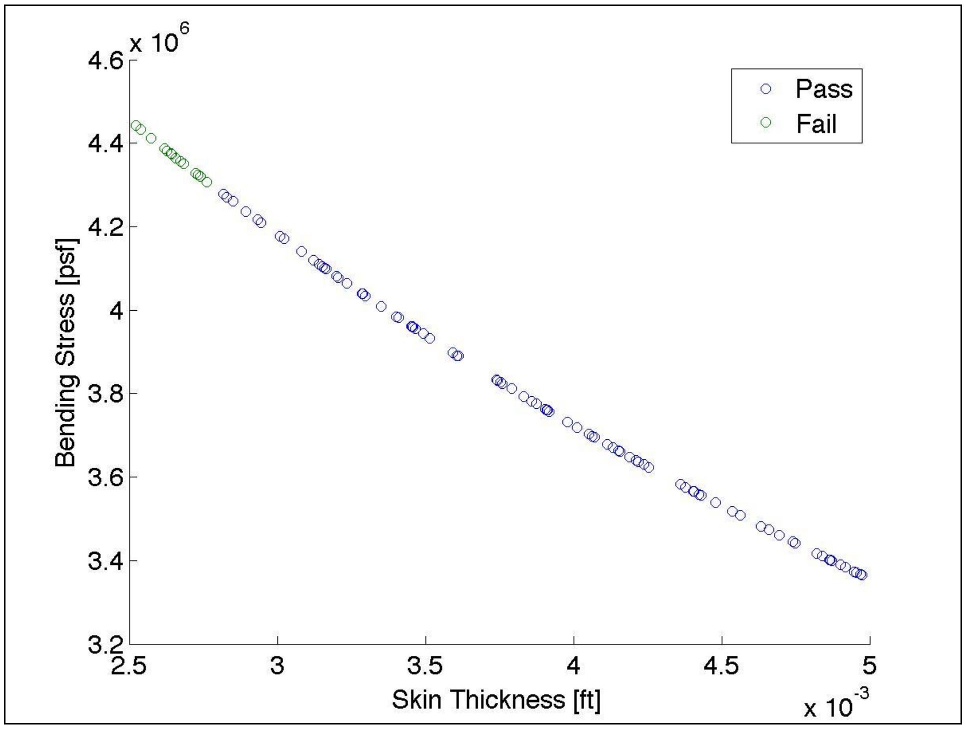


Figure 8.7: Bending stress against skin thickness (correlation *r =-0.9972*)

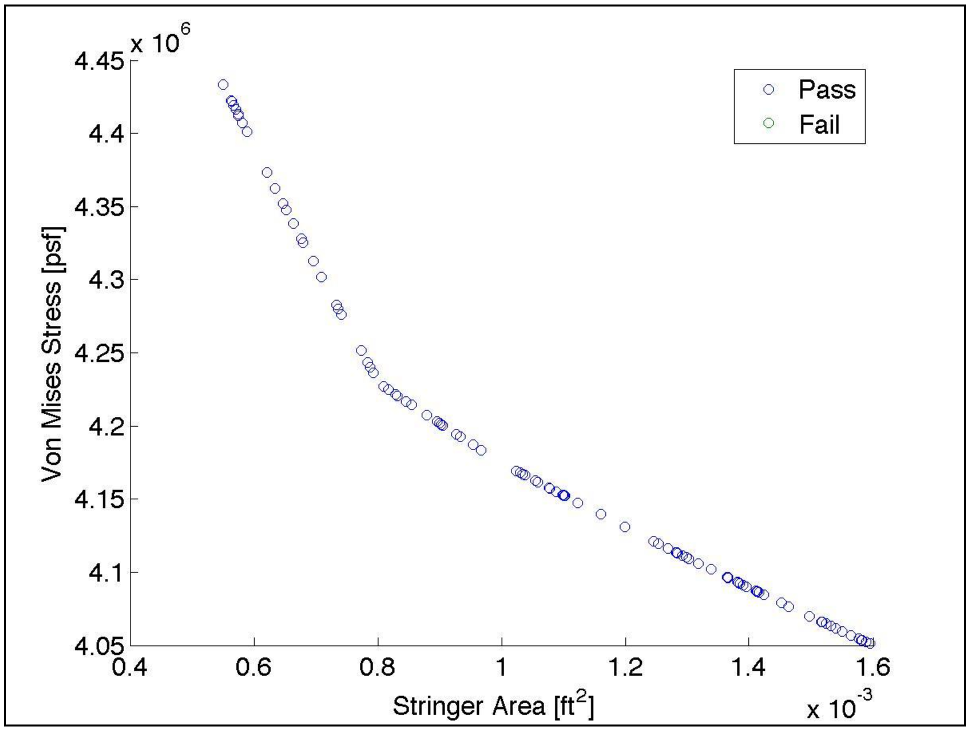


Figure 8.8: Maximum Von Mises Stress against stringer cross-sectional area

9) Final Design

After running the Monte Carlo optimization, several of the lowest weight geometries were converted to use standard parts from McMaster-Carr. Of these, the lowest weight trial resulted in a weight of 138.6 lbs. Figure 9.1 plots out the cross-section components for the final design. The three cells are spaced fairly evenly and moreover this design features a good distribution of stringers throughout each of the cells.

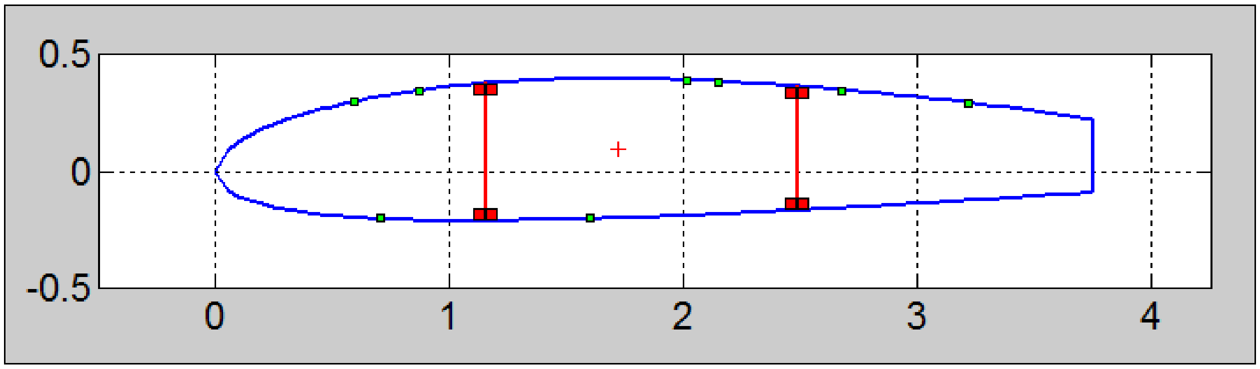


Figure 9.1: Final cross sectional geometry. Red plus sign is centroid, blue lines are skin, red lines are spars, red squares are spar caps, and small green squares are stringers.

Specifically, the geometry for this trial is shown in Table 9.1. This design uses well-spaced stringers to implement a fairly small skin thickness, which moderates the total weight substantially. There are also only a moderate number of stringers in the design. As a result a large number of ribs are required in order to prevent failure, however ribs are fairly light and thus this is a small weight penalty.

Table 9.1: Final design geometry and max stresses

Material	7475-T6	
Spar placement [ft]	1.15	2.49
Spar thickness [in]	0.032	
Skin thickness [in]	0.04	
Spar cap area [in^2]	0.24	
Stringer area [in^2]	0.094	
Number stringers	8	
Number ribs	14	
Total cross section [in^2]	8.8	
Max bending stress [psf]	4.30E+06	
Max shear stress [psf]	1.20E+06	
Weight [lbs]	138.63	

10) Calculation Run-Through for Final Design

This section demonstrates the calculations performed in obtaining stresses, verifying failure criteria, and computing weights as described in previous sections for the final design in order to verify the accuracy of the analysis presented in this report. Most intermediate steps are presented, and for conciseness equation numbers will be referenced to their respective sections rather than re-shown.

Using the geometry listed in Section 9, first the centroid and moments of inertia are computed. Using Equations 4.1 and 4.2, the x and z locations of the centroid relative to the leading edge of the airfoil are determined:

$$x_c = 1.7236 \text{ ft} \qquad z_c = 0.09511 \text{ ft}$$

The moments of inertia about this centroid location are determined using the simplified versions of Equations 4.3-4.5, which are shown in Equation 4.6. Adding the contributions from the skin, stringers, spar caps, and spars, the moments of inertia about the centroid are computed:

$$I_x = 0.003951 \text{ ft}^4 \qquad I_z = 0.1299 \text{ ft}^4 \qquad I_{xz} = 0.05496 \text{ ft}^4$$

For most of the stress computations and failure criteria verification, calculations for only one panel are shown for conciseness. The skin panel in the top right of the first cell was used, as shown in Figure 10.1. Note that all panels are actually subdivided into 10 subpanels (not shown in Figure 10.1) as described in Section 4.3.1 in order to maintain the validity of the approximation of constant shear flow through webs. Thus, the calculations shown below are not for the entire panel, but instead for only 1/10 of the panel (on the farthest right).

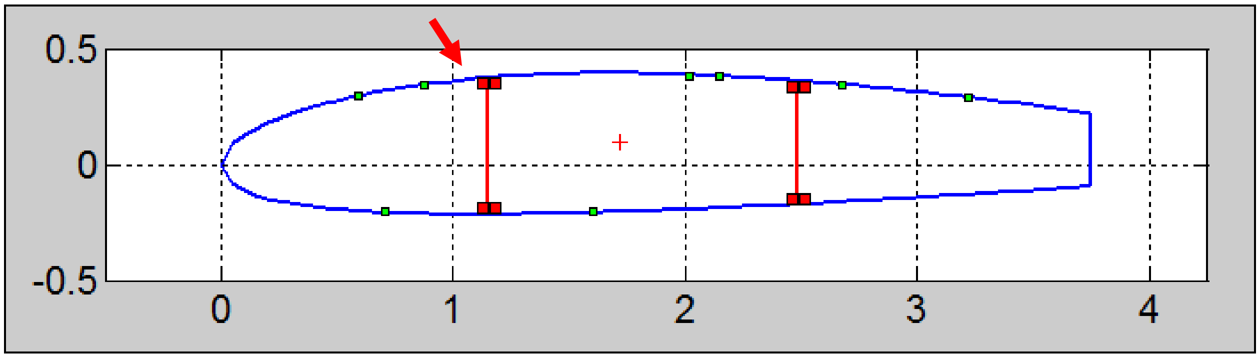


Figure 10.1: Panel used to demonstrate calculations for stresses and verification of failure criteria

First, the bending stress calculations are shown at this panel. Using Equation 4.7, the bending stress is computed:

$$\sigma_y = -3.9487 * 10^6 \text{ lb/ft}^2$$

The negative sign of the bending stress verifies that the panel is in compression.

Next, the shear stresses are computed. First, the flexural component of the shear flow due to aerodynamic loads is calculated. Since the top right skin panel of each cell is the location of the cut, the flexural shear flow is zero in this panel. For verification of the computations from Equation 4.10, the flexural shear flow in the 2nd panel (more specifically, the second 1/10 from the farthest right of the top right panel in the 1st cell) is also shown:

$$q_{b,1} = 0 \text{ lb/ft} \qquad q_{b,2} = 46.428 \text{ lb/ft}$$

The positive sign indicates that the flexural shear is in the counterclockwise direction. Using Equations 4.11 and 4.12, the torsional component of the shear flow due to aerodynamic loads is computed for all three cells:

$$q_{s,1} = -908.68 \text{ lb/ft} \quad q_{s,2} = -2366.5 \text{ lb/ft} \quad q_{s,3} = -2158.9 \text{ lb/ft}$$

The negative sign indicates that the torsional components of the shear flows are clockwise. The torsional component of the shear flow due to the moment about the aerodynamic center is also computed for all three cells using Equations 4.14-4.16:

$$q_{t,1} = -345.31 \text{ lb/ft} \quad q_{t,2} = -422.89 \text{ lb/ft} \quad q_{t,3} = -318.20 \text{ lb/ft}$$

The negative sign also indicates a clockwise orientation. Summing up the flexural and torsional components of the shear flow due to aerodynamic loads and the torsional shear flow due to the moment about the aerodynamic center, the total shear flow in the panel of interest is computed:

$$q_{tot,1} = q_{b,1} + q_{s,1} + q_{t,1} = -1254.0 \text{ lb/ft}$$

The shear stress is computed by dividing by the skin thickness:

$$\tau_{tot,1} = -3.762 * 10^5 \text{ lb/ft}^2$$

The shear center is computed using the procedure described in Section 4.3.4, as measured from the leading edge:

$$x_{sc} = 0.939 \text{ ft} \qquad z_{sc} = 0.504 \text{ ft}$$

The Von Mises Stress in this panel is computed using Equation 4.19:

$$\sigma_{VM,1} = 4.0021*10^6 \text{ lb/ft}^2$$

To verify the Von Mises Failure Criteria, the maximum allowable stress is computed using Equation 4.20:

$$\frac{\sigma_{yield}}{1.25} = 8.179*10^6 \text{ lb/ft}^2 \qquad \frac{\sigma_{ultimate}}{1.5} = 7.968*10^6 \text{ lb/ft}^2$$

Since the Von Mises stress of this panel does not exceed either of these conditions, the panel passes the Von-Mises failure check.

Next, the buckling failure criteria are evaluated. Referring to section 6.2, the critical bending and shear stresses are computed referring to Equations 6.2-6.9:

$$\sigma_{y,cr} = 7.26*10^7 \text{ lb/ft}^2 \qquad \tau_{tot,cr} = 4.10*10^6 \text{ lb/ft}^2$$

The bending and total shear stresses computed earlier are well below these values, and hence this panel passes the buckling check.

Next, the fracture and fatigue failure criteria are evaluated. Using Equation 6.10 and assuming an edge crack of *a*=10 microns, the stress intensity factor is computed:

$$K_I = 1.255 \text{ MPa}\sqrt{\text{m}}$$

which is below the critical stress intensity factor of KIC=29 MPa for Al 7075-T6. The material will not fail in fracture for this crack size. The critical crack size at which fracture will occur is computed using Equation 6.11:

$$a_{CR} = 0.124 \text{ m}$$

The number of cycles till fatigue failure occurs is computed using Equation 6.12:

$$N = 6.69*10^5 \text{ cycles}$$

To compute the wing divergence velocity, the torsional rigidity is computed using Equation 7.2:

$$GJ = 7.27 * 10^5 \text{ lb-ft}^2$$

The wing divergence velocity is then computed using Equation 7.3:

$$V_{div} = 5281.3 \text{ ft/s} = 3600.9 \text{ mph}$$

which is much higher than the dive velocity. This ensures that the wing will not fail through divergence.

Flutter natural frequency using the simplified cantilever beam approach is then computed. The effective spring constant of the wing is computed using Equation 7.4, assuming a wing tip deflection of δ_{max}=0.4892 ft:

$$k_{eff} = 14391 \text{ lb/ft}$$

The flutter natural frequency of the wing is then computed using Equation 7.5:

$$f_n = 8.64 \text{ Hz}$$

which is fairly low compared to most plane wings. This is most likely due to the fact that the approach used to obtain this value was highly simplified and made use of a single degree-of-freedom system (bending mode of vibration), which cannot go unstable (as mentioned by Megson) and hence is not a theoretically valid approach.

11) CAD and FEA

11.1) CAD Details

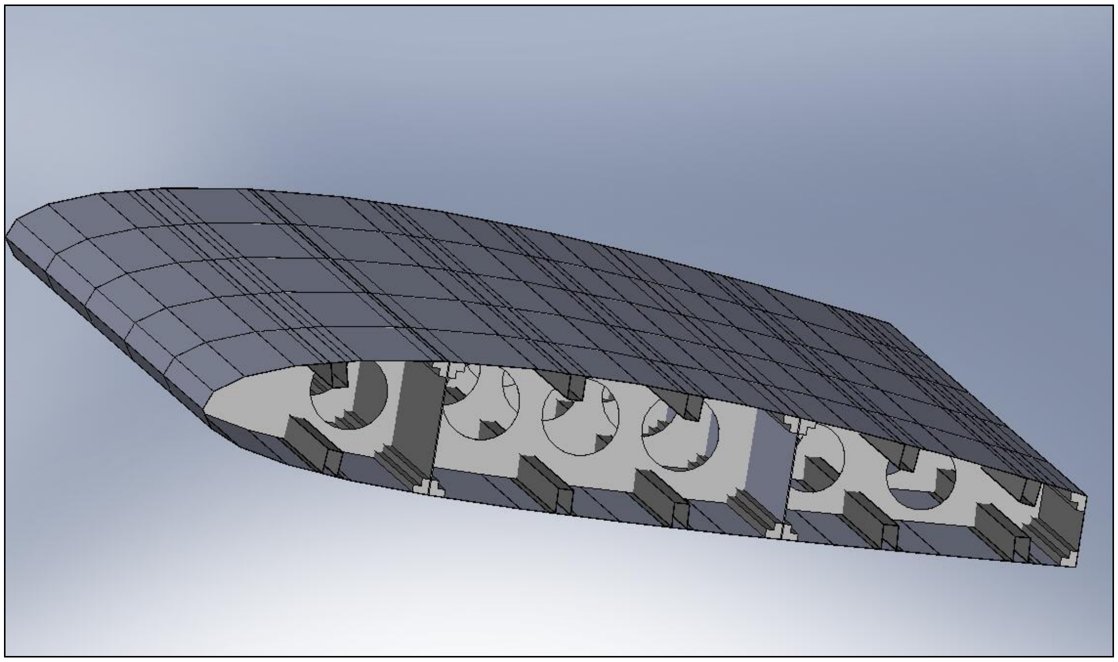


Figure 11.1: CAD of wing half-span for arbitrary wing cross section

A CAD model of the wing design has been developed using SolidWorks, as shown in Figure 11.1. This shows an example wing configuration which is not necessarily the most optimized, but gives an idea on how all the structural components are mounted together. The wing is a three-cell box beam and its structural portion extends from the airfoil's leading edge to 75% of the chord length.

In the drawings appear all the structural elements: the wing skin, the spars with spar caps, the stringers and the ribs. The skin itself is not actually drawn as a continuous surface and this is due to the fact that it needs to match the function of the NACA 2412 airfoil. The procedure to create a realistic airfoil starts with a Matlab code that writes the airfoil coordinates in a .dxf file. This file is then imported in SolidWorks which automatically connects the points provided with straight lines/panels and gives a discontinuous profile. The higher the number of points the more accurate the airfoil shape, with the disadvantage that this increases the number of entities in the CAD drawing, significantly slowing the graphic visualization and manipulation. For this reason only a relatively small number of panels is displayed, still giving a good idea of the actual wing shape.

The spars are connected to the airfoil skin through L-shaped spar caps that also hold a skin panel, which closes the third cell at the back end of the structure. The ribs are shaped to fit the internal cavity of the wing and some holes are cut through them in order to lighten the structure. Top-hat stringers run parallel to the spars and are connected uniquely to the skin.

11.2) Riveted Joints

All the components in the structure are attached to each other through riveted joints. Three different connections are designed: spar-skin, stringer-skin, and rib-skin. Every one of these joints is not only required to fix the skin to the other components, but also to fasten different skin panels between them. In order to give material continuity throughout the joint some aluminum web straps are designed to be inserted between the skin and the connecting element (i.e. stringer, spar w/ spar cap or rib). With this device the carried load is split between rivets and strap, leading to a safer connection with only a negligible weight increase. The joints between spar or rib and the skin require the presence of spar (or rib) caps. It is convenient to define a joint plane which contains the rivet axis and the direction along which the rivets' row is situated.

As far as the loads go, two different kinds of joints can be defined: those which connect skin panels in the spanwise direction (rib-skin joints) and those which connect them in the chordwise direction (spar-skin and stringer-skin joints). The first configuration carries both tensile stresses and shear stresses perpendicular to the joint plane ('x-z' plane). These shear stresses derive from the shear stresses in the airfoil cross sectional plane ('x-z' plane) through a simple equilibrium condition: in order to keep the skin panels from rotating about the 'z' axis, equal and opposite shear stresses have to lay on the 'y-z' plane, which is perpendicular to the joint plane. The second kind of joint only carries the shear stresses, in this case parallel to the 'x-z' plane. From this discussion it is easy to notice that the joints actually see shear stresses also parallel to the joint plane, but their contribution is neglected since they are split between a large number of rivets and, therefore, do not threaten the strength of the connection considerably. The rib-skin joint is designed to be equipped with a double row of rivets in order to carry the additional load.

Four different failure modes are identified for riveted joints. The first mode (1) is shear across the rivet's diameter at the interface between the connected plates. The second mode (2) is due to bearing pressure that can cause fracture both in the rivet and in the plate. The last two modes affect only the plate that can fail either in tension (3), since the rivet holes reduce the skin's cross sectional area, or in shear (4), perpendicularly to the joint plane. Figure 11.2 shows the key parameter for the joint design process; the actual joints are different from that in the picture.

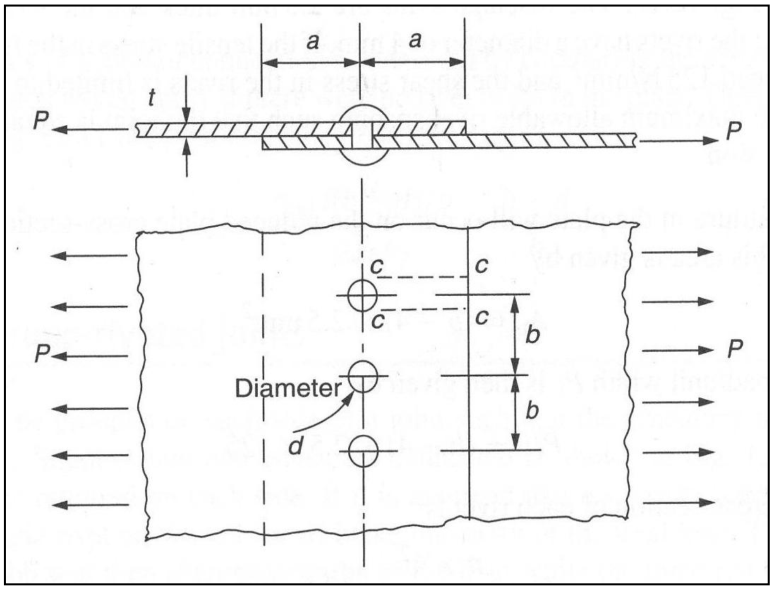


Figure 11.2: joint key parameters and geometry

The actual sizing procedure is based on verification. The joint is characterized by two critical parameters: the rivet diameter, *d*, and the rivet spacing, *b*. These parameters were chosen a priori (*d*= 0.315 in and *b*=0.75 in) and then failure was checked by computing the minimum stress to cause fracture in all the four modes. The rivets come from the German DIN standard parts and they are made of aluminum 7475-T651 with $s_{ult,7475}$=85000 psi, while the plates are made of 2024-T3 with $s_{ult,2024}$=70000 psi. Note that while the optimized design used the Al 7475-T6 material for the plates, for purposes of this rough preliminary design the behavior is assumed to be similar. Equation 11.1 gives expressions for the critical load per unit length *P* [3].

$$\begin{array}{ll} (1)\ \ P = \dfrac{\pi d^2 \tau_1}{4b} & (2)\ \ P = \dfrac{p_b t d}{b} \\ (3)\ \ P = \dfrac{\sigma_{all} t(b-d)}{b} & (4)\ \ P = \dfrac{2at\tau_2}{b} \end{array} \qquad (11.1)$$

where τ_1 is the rivet's maximum shear stress (evaluated with Von Mises yield criterion), p_b is the bearing pressure, *t* is the plate thickness, *a* is the distance between the hole center and the plate edge, and s_{all} and τ_2 are the maximum allowable tensile and shear stresses in the plate respectively. Equation 11.2 shows the values assumed by some parameters.

$$a = 0.5 \text{ in} \qquad t = 0.049 \text{ in}$$

$$\begin{aligned} \tau_1 &= \frac{\sigma_{ult,7475}}{FOS_{riv}\sqrt{3}} = 16360 \text{ psi} \\ \tau_2 &= \frac{\sigma_{ult,2024}}{FOS_{pl}\sqrt{3}} = 26940 \text{ psi} \\ \sigma_{all} = p_b &= \frac{\sigma_{ult,2024}}{FOS_{pl}} = 46670 \text{ psi} \end{aligned} \tag{11.2}$$

where FOS_{riv}=3 and FOS_{pl}=1.5 are the factors of safety on the rivet and plate respectively. From these relations Equation 11.3 was derived to express the maximum allowable tensile stresses in the joint.

$$\begin{aligned} &(1)\ \sigma = \frac{\pi d^2 \tau_1}{4bt} = 34690 \text{ psi} \qquad &&(2)\ \sigma = \frac{\sigma_{all}(b-d)}{b} = 27070 \text{ psi} \\ &(3)\ \sigma = \frac{2a\tau_2}{b} = 35920 \text{ psi} \qquad &&(4)\ \sigma = \frac{d}{b}\sigma_{all} = 19600 \text{ psi} \end{aligned} \tag{11.3}$$

These values are computed for the single row joint, while those in Equation 11.4 are for the skin-rib joint and are double the values in Equation 11.3.

$$\begin{aligned} &(1)\ \sigma = 69380 \text{ psi} \qquad &&(2)\ \sigma = 54140 \text{ psi} \\ &(3)\ \sigma = 71840 \text{ psi} \qquad &&(4)\ \sigma = 39200 \text{ psi} \end{aligned} \tag{11.4}$$

These values are compared to the actual stresses present in the structure which are obtained from average maximum values given by the optimization code which are τ_{xz}=11110 psi and s_y=24310 psi. From these the Von Mises equivalent tensile stresses in Equation 11.5 are computed.

$$\begin{aligned} \sigma_{VM} &= \sqrt{3}\tau_{xz} = 19240 \text{ psi} \qquad && \text{single joint} \\ \sigma_{VM} &= \sqrt{\frac{\sigma_y^2 + 6\tau_{xz}^2}{2}} = 25800 \text{ psi} \qquad && \text{double joint} \end{aligned} \tag{11.5}$$

All the values computed for the maximum allowable stresses in the joints are greater than the Von Mises stresses computed for the worst case loading scenario. Figure 11.3 shows a complete set of joints in the wing structure. More detailed pictures are shown in Appendix A.

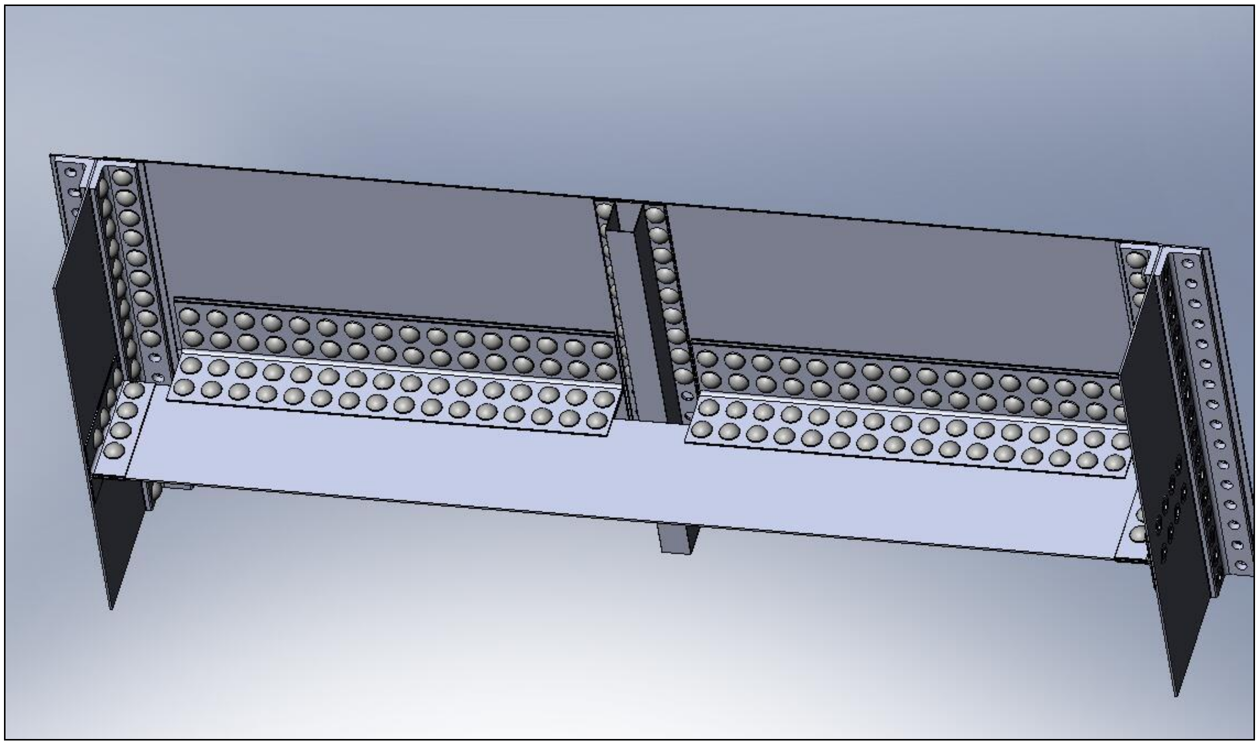


Figure 11.3: CAD of a complete set of riveted joints

11.3) FEA Structural Verification

A Finite Element Analysis has been performed with COSMOSworks in order to double check the stress and deflection analytical results so far obtained. The complexity of the complete CAD model is too high to execute an accurate FEA on it. Particularly, the very small skin thickness would require an excessively fine mesh strongly increasing the computational time. For this reason a simpler model was designed so that reasonably accurate results can be obtained at the cost of an acceptable level of approximation. The wing was reduced to a rectangular-shaped box beam whose moments of inertia about the ‘x’ and ‘y’ axes match those of the actual structure. The beam’s vertical and horizontal elements have two different skin thicknesses for this purpose. The box height and width were set to be equal to the maximum airfoil thickness and to 75% of the chord length respectively in order to predict the bending stresses correctly. This left only the two skin thicknesses as unknown variables in the moment of inertia equations. A Matlab code was written to compute these two values through an iterative process leading to $t_{horizontal}$= 0.065 in and $t_{vertical}$= 0.36.

The first step in the analysis is to assign the material properties to the structure and the material chosen was aluminum 2024-T3. Again, while the material chosen in the optimized design was Al 7475-T6, for purposes of this rough finite element analysis the behaviors of the materials are assumed to be similar. A pretty coarse solid mesh was chosen to reduce the computational work required and still obtain good results. Next, the root section of the beam was constrained properly so that it does not move and can sustain the loads. All the webs of the root section were fixed in the y (spanwise) direction. In addition to that, one vertical element was constrained in the ‘x’ (chordwise) direction and one horizontal element was fixed in the ‘z’ (vertical) direction. This set of constraints was thought to simulate the wing attachment accurately. The external loads were finally applied to the structure. Non-uniform pressure loads were spread over the bottom and the front surfaces of the beam in order to simulate both the lift and drag forces at the same time. Since the actual load distributions are characterized by a sharp step at 80% of the span, two different quadratic distributions were applied on each surface, the first going from wing tip to 80% of the span, the second one spreading from that point to the wing root.

The most interesting results are the stresses and the deflection of the wing. Figure 11.4 shows the stresses on the upper surface, while Figure 11.5 displays the stresses on the bottom surface of the box beam. It is easy to see that the stresses’ absolute value increases toward the wing root and that the upper surface in characterized by negative (compressive) stresses while the lower has mainly positive (tensile) stresses. Also the values are the same order of magnitude of those computed for the actual wing. A couple of weird patterns can be seen on these plots. The wing root section presents very low stresses; this is probably due to the fact that the constraints do not allow the material to deform causing positive stresses on the compressed side and negative on the other. This apparently results in a balanced set of stresses with very low values.

All along the bottom surface a wide stripe of compressive stresses appears. This is probably due to the fact that the thin skin deforms under the pressure so that the center of the web is actually in compression on the lower surface. The skin panels deflect toward the inside of the structure and the fibers along the bottom surface are in compression when crossing this stripe. This is not a very realistic result and it is produced by the not completely accurate model used.

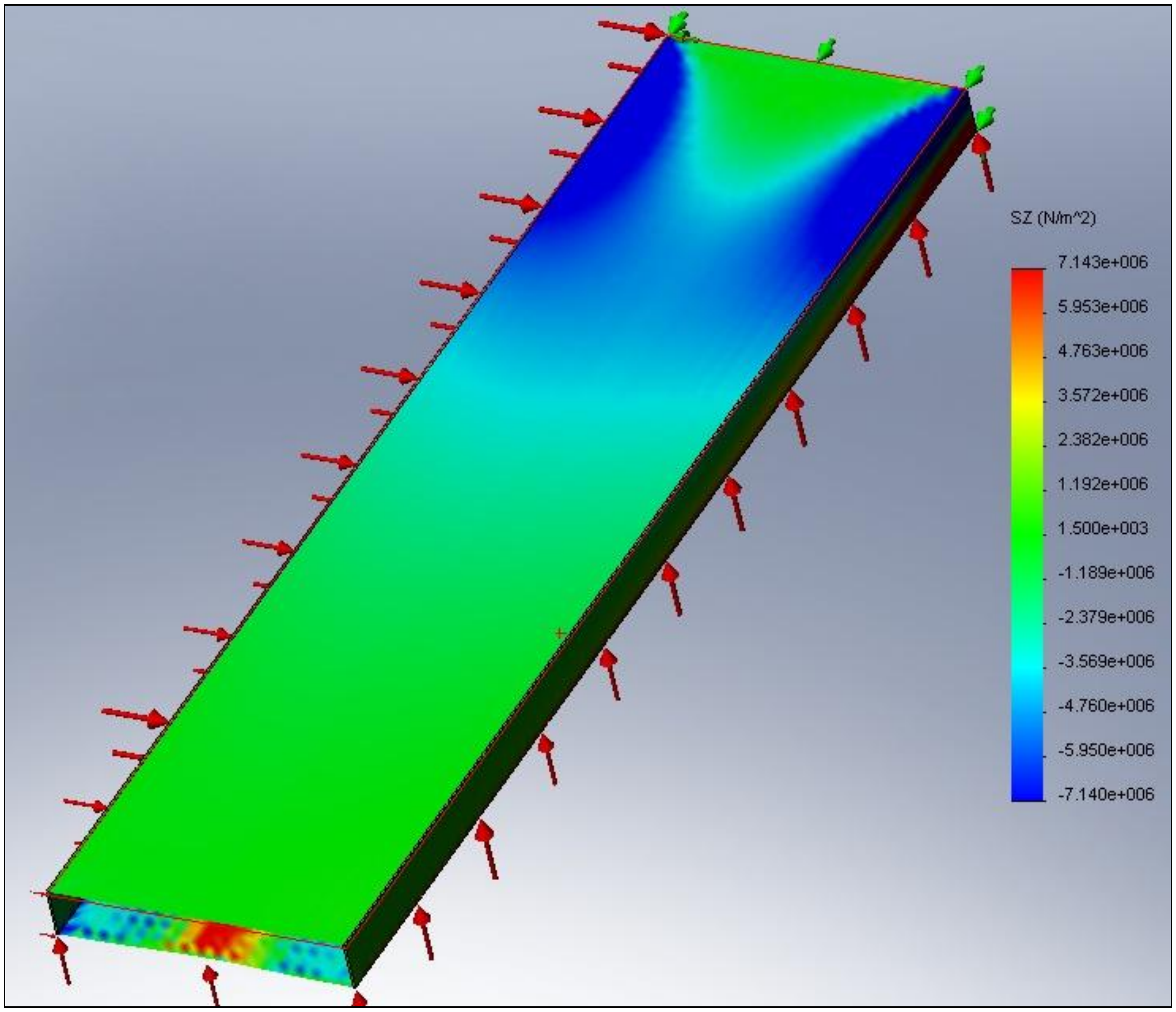


Figure 11.4: Stress Distribution on Upper Surface of Box-Beam

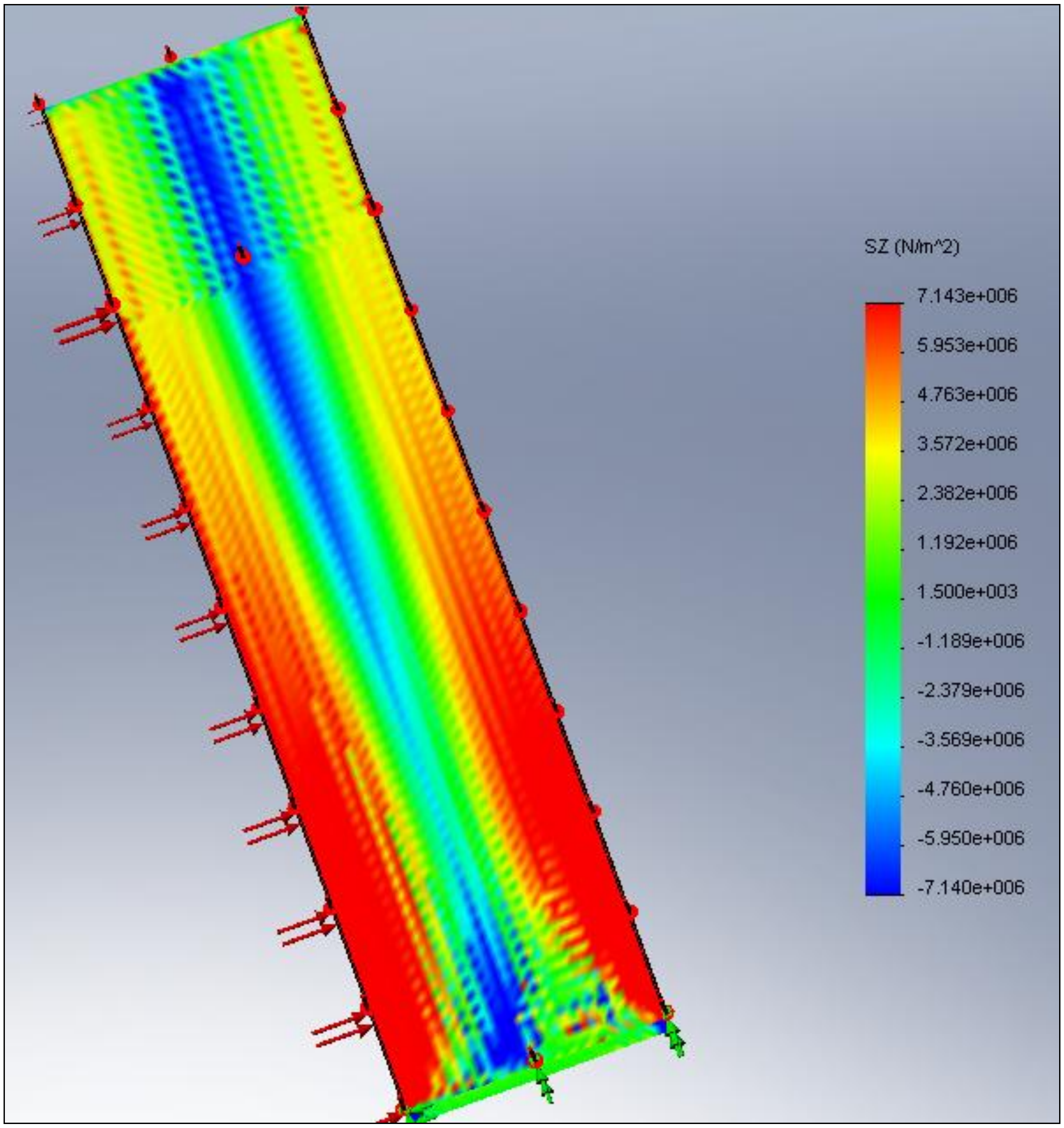


Figure 11.5: Stress Distribution on Lower Surface of Box-Beam

Figures 11.6 and 11.7 show the wing deflection on the upper and bottom surfaces respectively. Consistent with the stress plots the maximum deflection appears in the central portion of the lower surface. The deflection increases pretty uniformly toward the wing tip and the values shown are only one order of magnitude off from those computed for the actual wing.

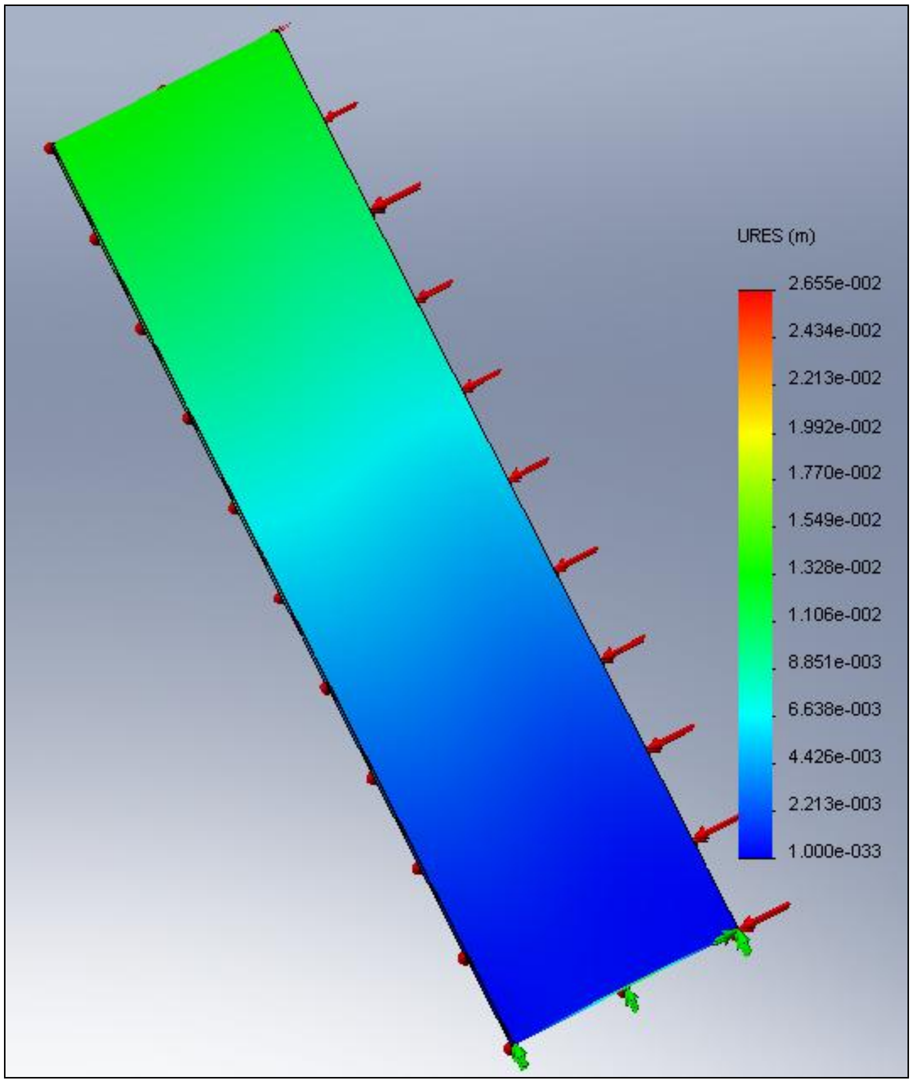


Figure 11.6: Deflection Distribution on Upper Surface of Box-Beam

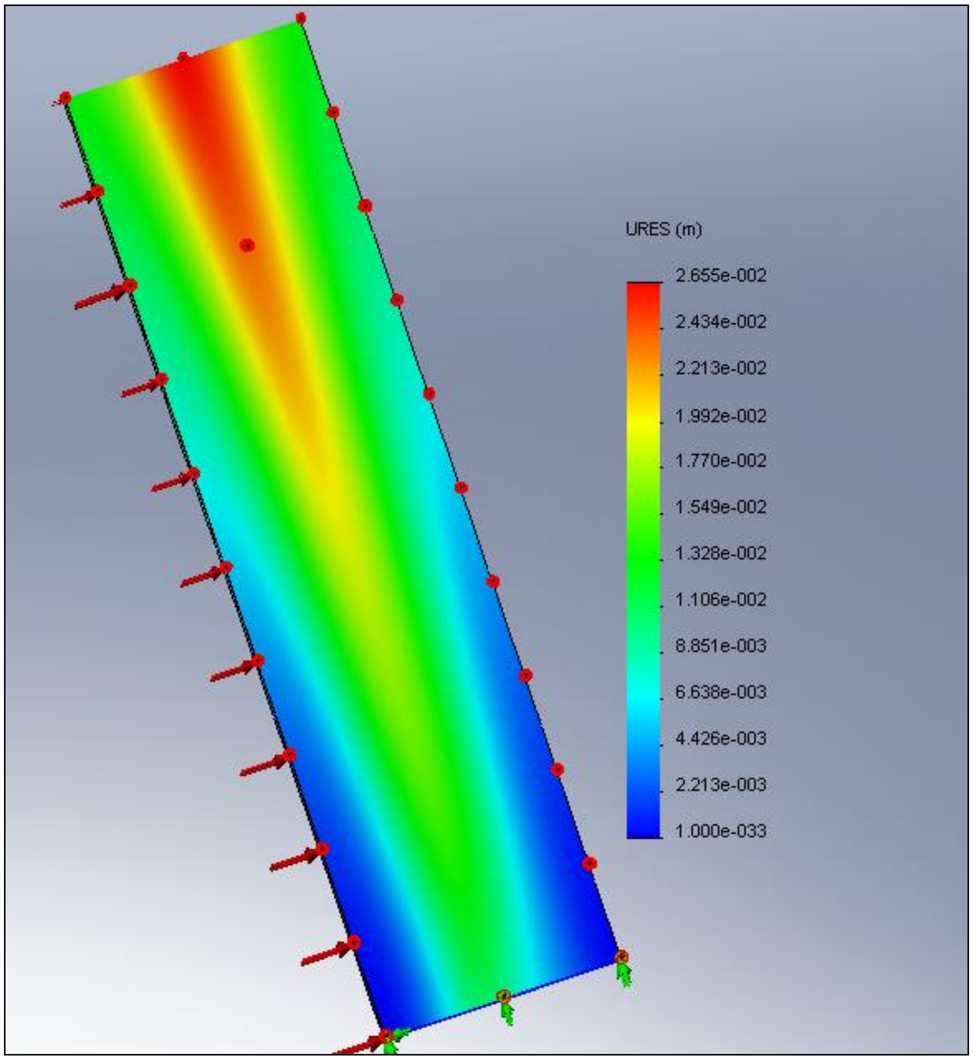


Figure 11.7: Deflection Distribution on Lower Surface of Box-Beam

12) Conclusion

While many aspects of the wing design have been considered in this report, there is still much left to be done before releasing for production. Throughout the analysis presented, many approximations and assumptions were used that may contribute some lack of accuracy that must be addressed. For example, the stress and buckling analyses required some simplifying assumptions (panel idealization, constant shear flow, etc.) that probably affected the obtained results; a more accurate theoretical model could provide more reliable outcomes but might be better handled with simulation software. The fracture and fatigue analysis would need to be more systematic and check every critical location in the structure, particular where stress intensity factors are present like at rivet holes and sharp edges. Only preliminary aeroelasticity analysis was performed and hence more rigorous flutter and aileron reversal analysis must be considered. The finite element verification has to be extended to the complete CAD model, including riveted joints and bolted attachments to the fuselage, since it presents the most accurate prediction of the wing's behavior aside from direct experimental testing. The control surface design was so far neglected and must be addressed in the future. Lastly, due to time constraints, the optimization code was only run for a sufficient number of trials to produce a desirable final weight; additional trials or a more directed optimization routine (such as evolutionary optimization) may lead to a lighter and more effective wing design.

13) References

[1] White, Frank M., *Fluid Mechanics, Sixth Edition*, McGraw-Hill Companies, Inc., New York, NY, 2008, pp. 818

[2] Federal Aviation Administration (FAA), "Part 23: "Airworthiness Standards: Normal, Utility, Acrobatic, and Commuter Category Airplanes", Federal Aviation Regulations (FARS),

[3] Megson, T.H.G., *Aircraft Structures for Engineering Students, Fourth Edition,* Elsevier Ltd., Oxford, UK, 2007

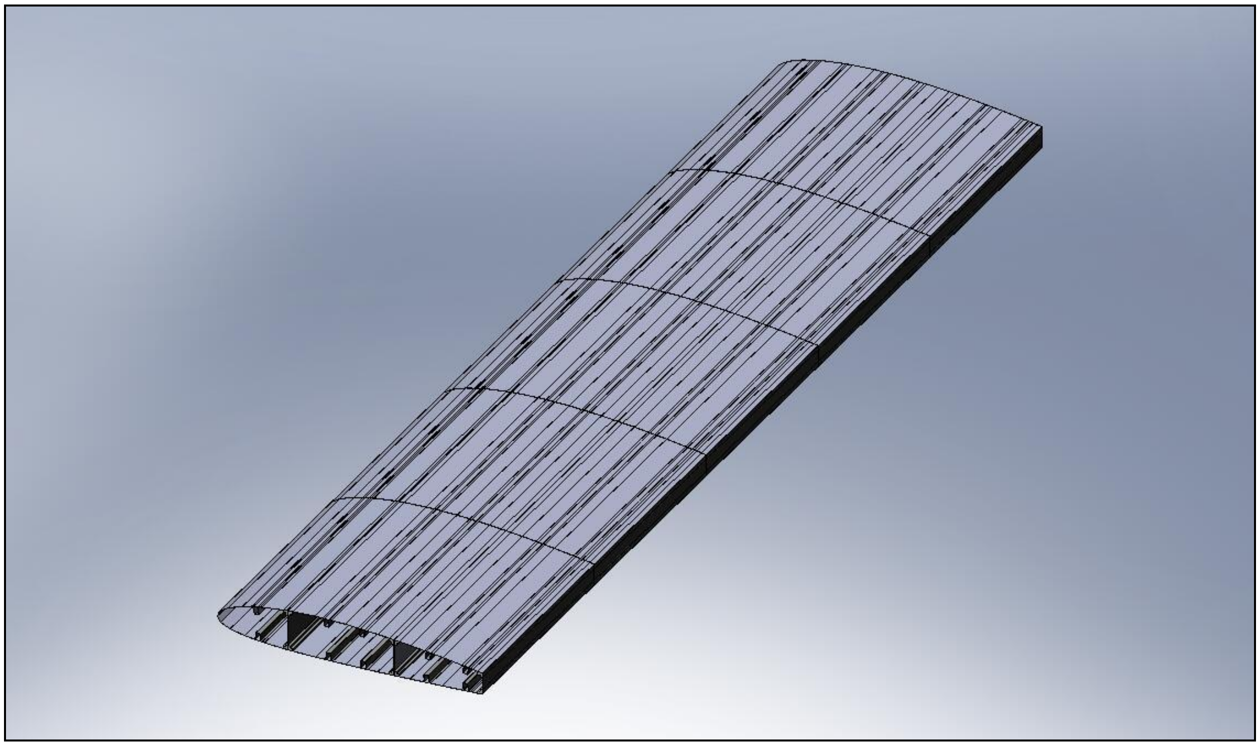
[4] Sun, C. T., *Mechanics of Aircraft Structures*, John Wiley & Sons, Inc., Hoboken, NJ, 2006

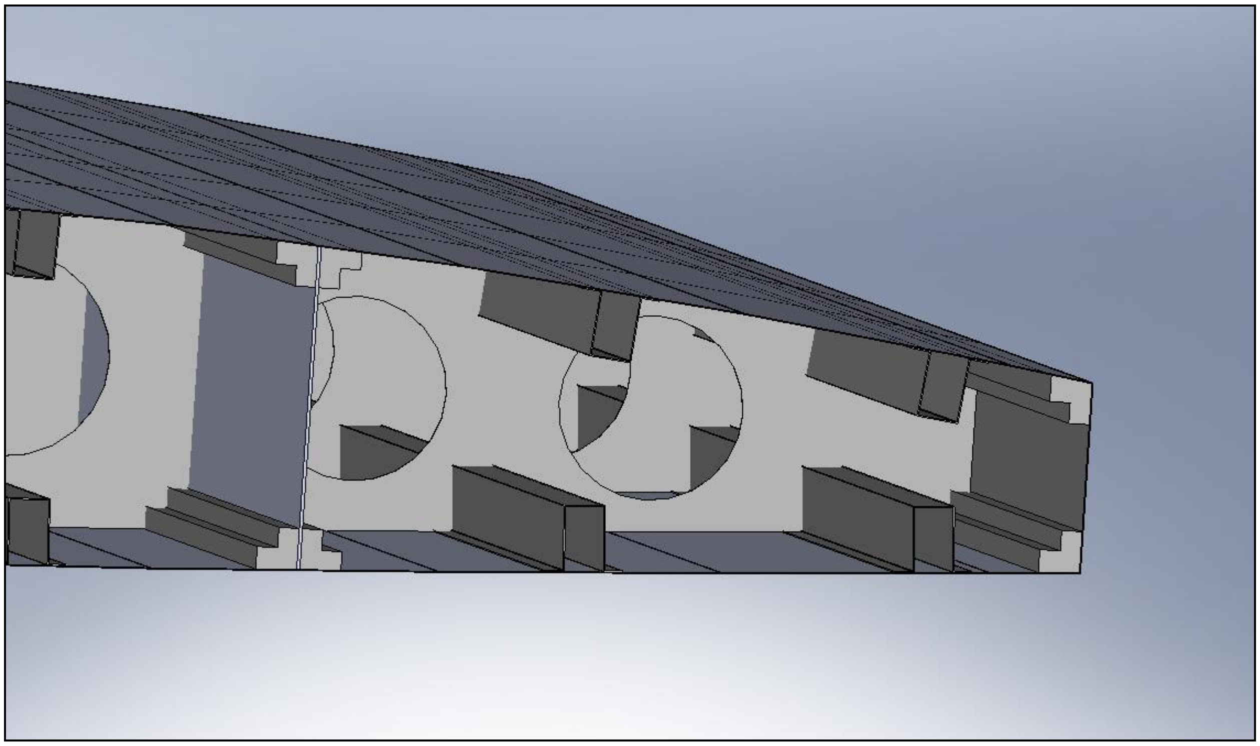
[5] Griffin, K. H., "The influence of the method of stringer attachment on the buckling and failure of skin panels with square top-hat stringers", *Ministry of Supply Aeronautical Research Council*, Thesis, College of Aeronautics, Cranfield, 1952

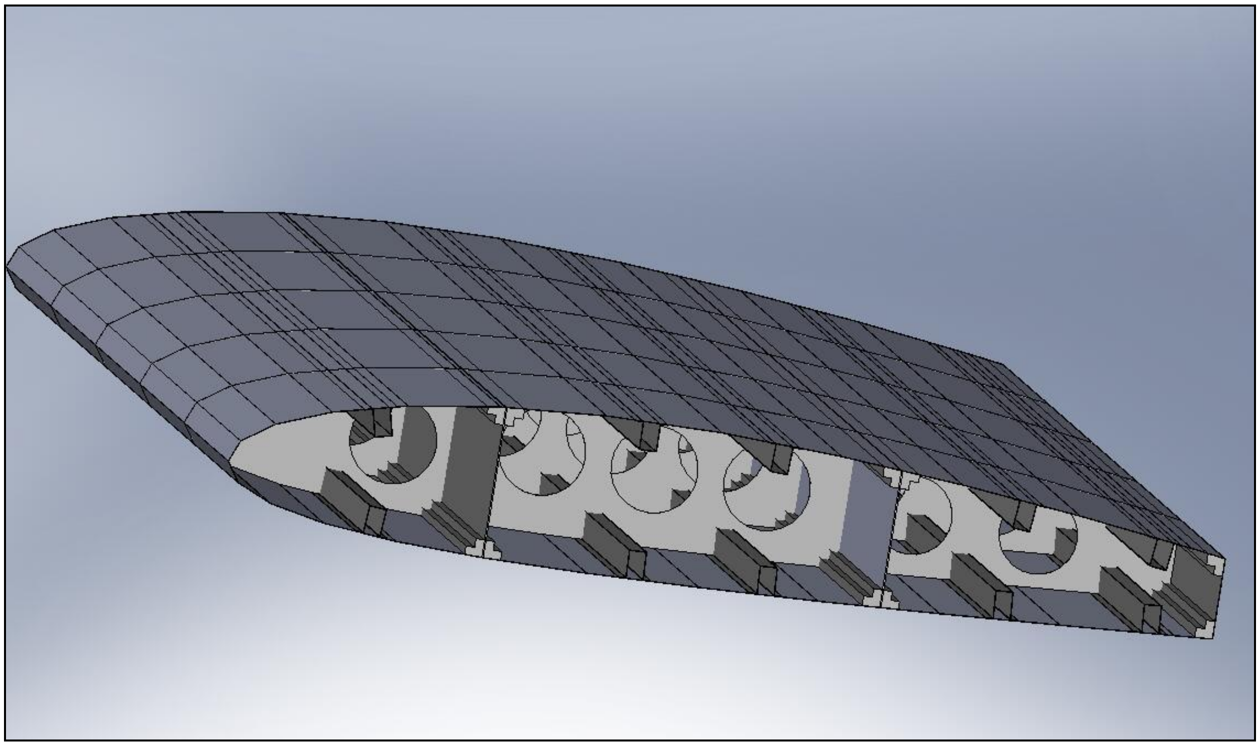
[6] Second moment of area, Wikipedia, http://en.wikipedia.org/wiki/Second_moment_of_area, Accessed May 8, 2009

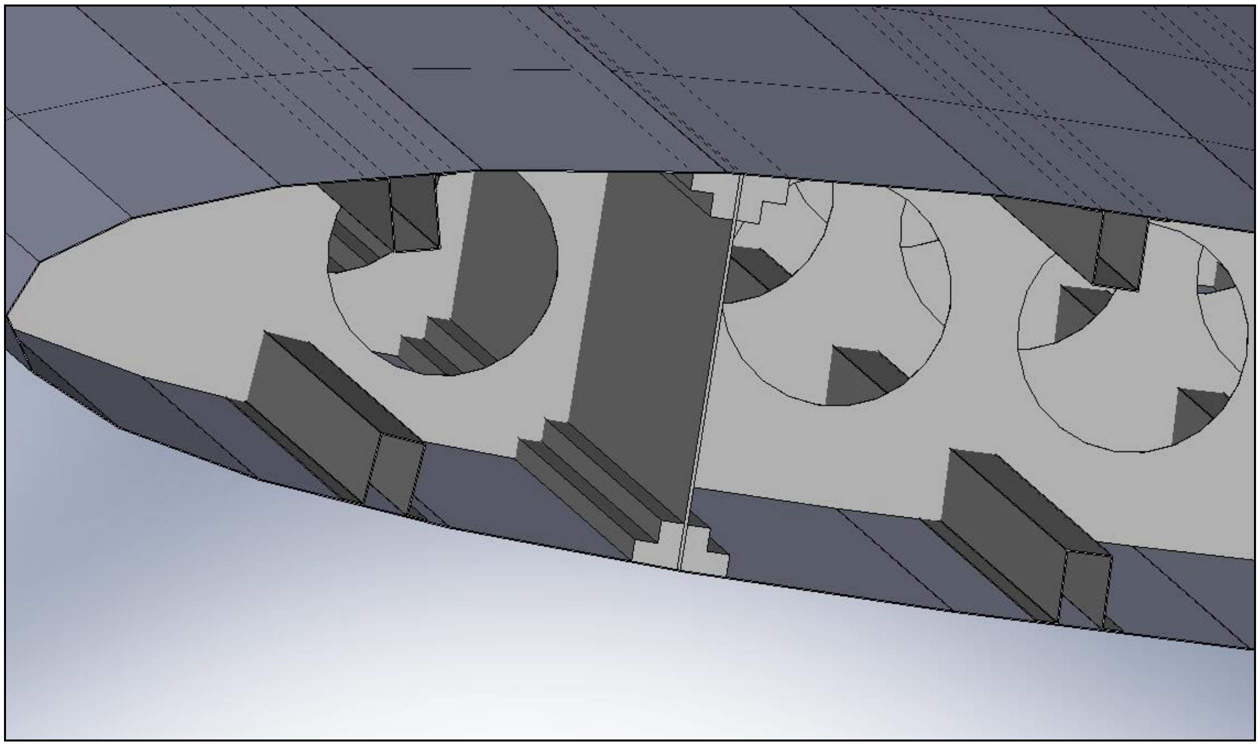
Appendix A: CAD Pictures

Wing Design

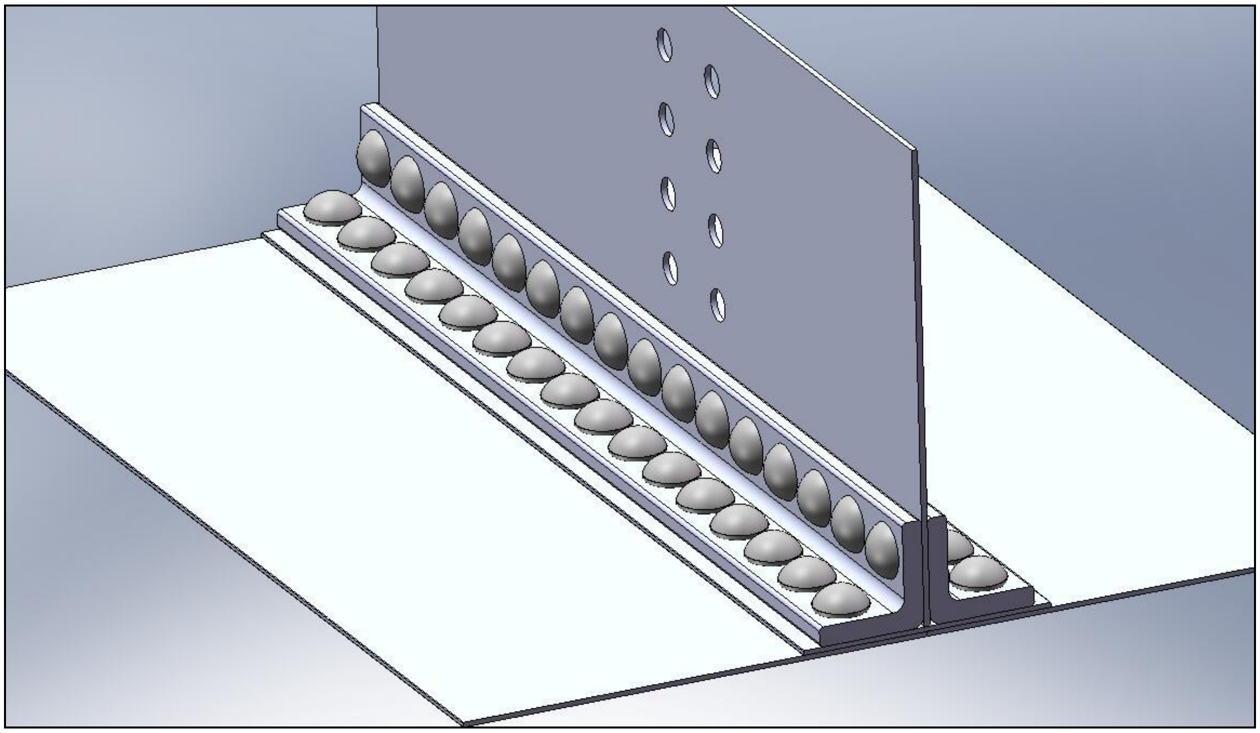


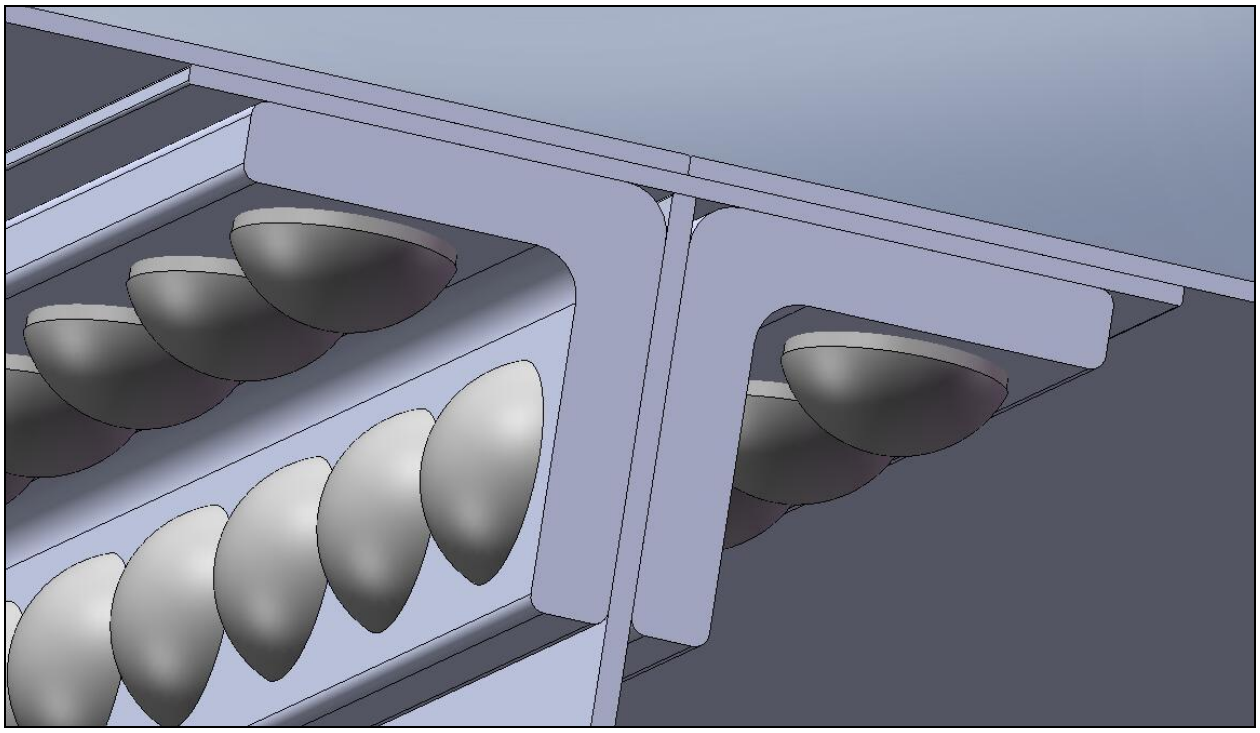




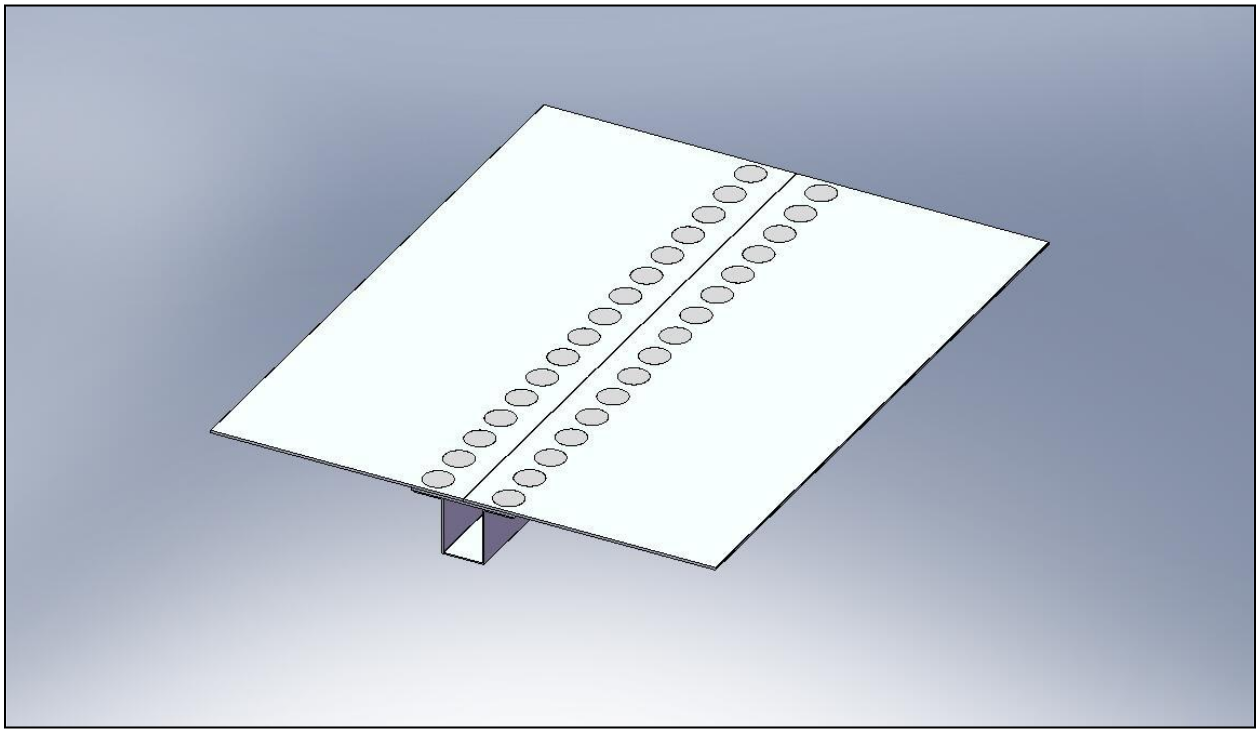


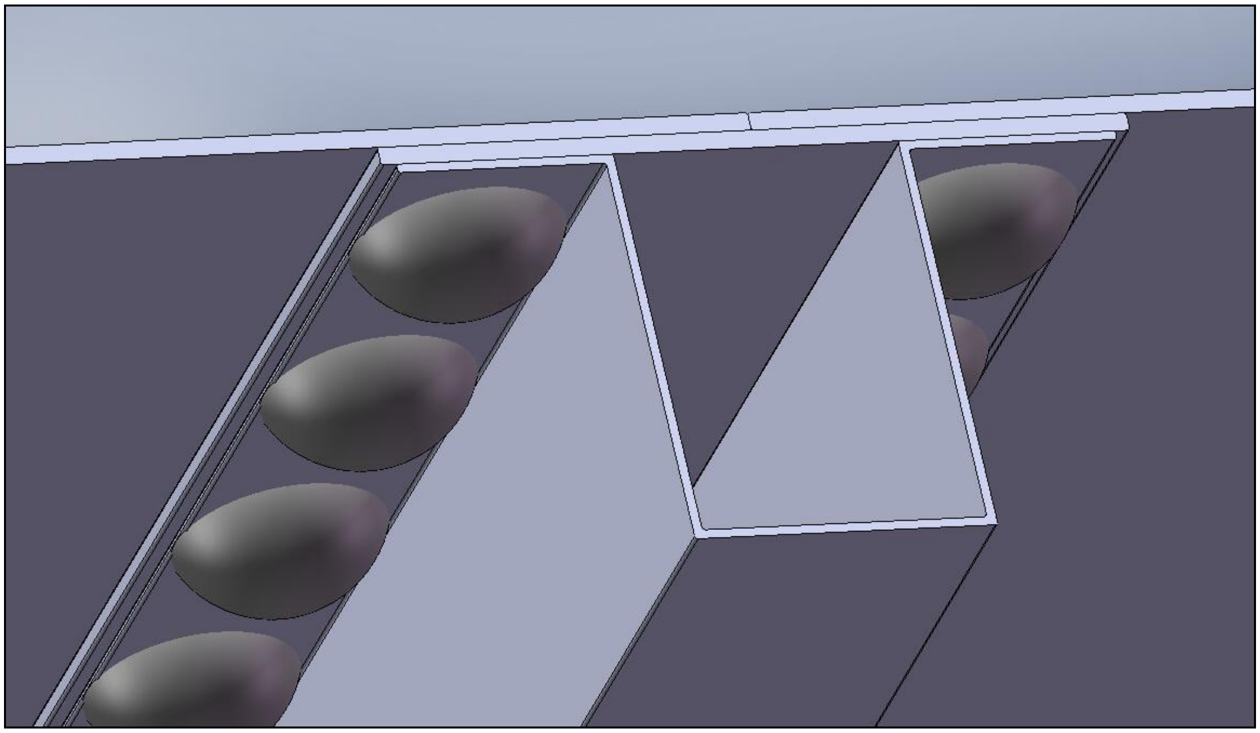
Rivet Design- Skin to Spar



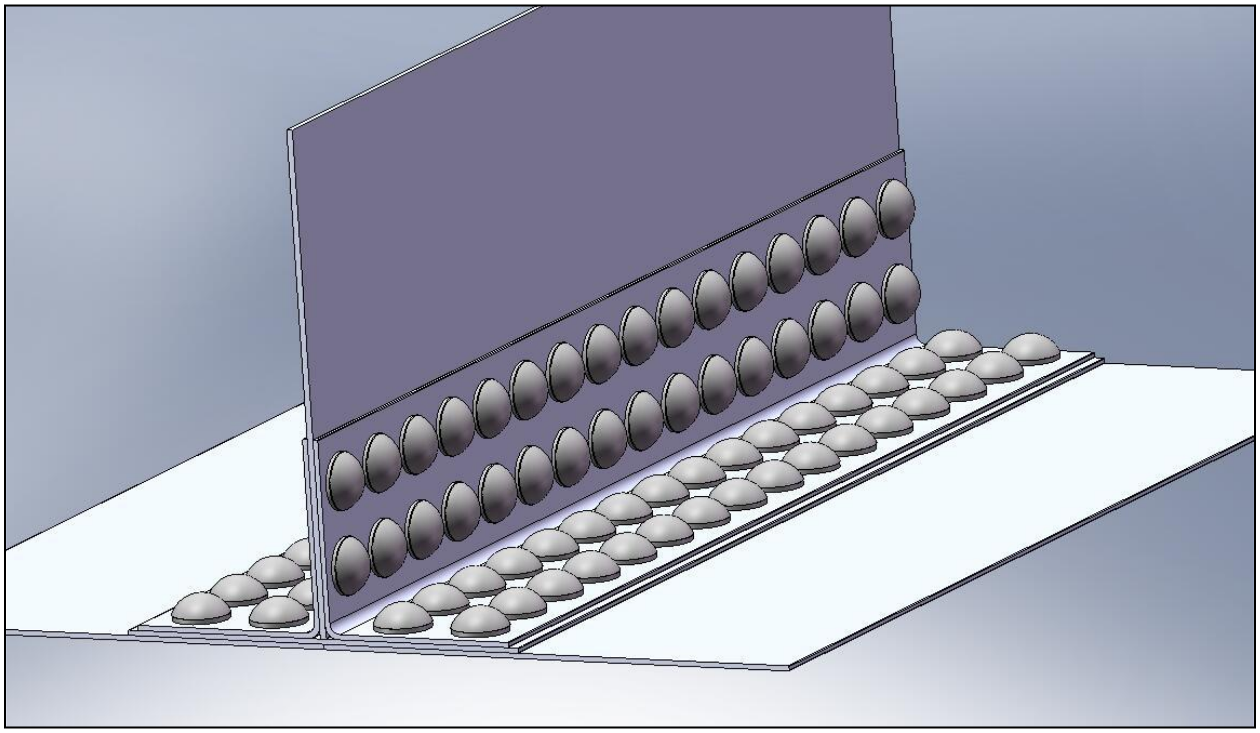


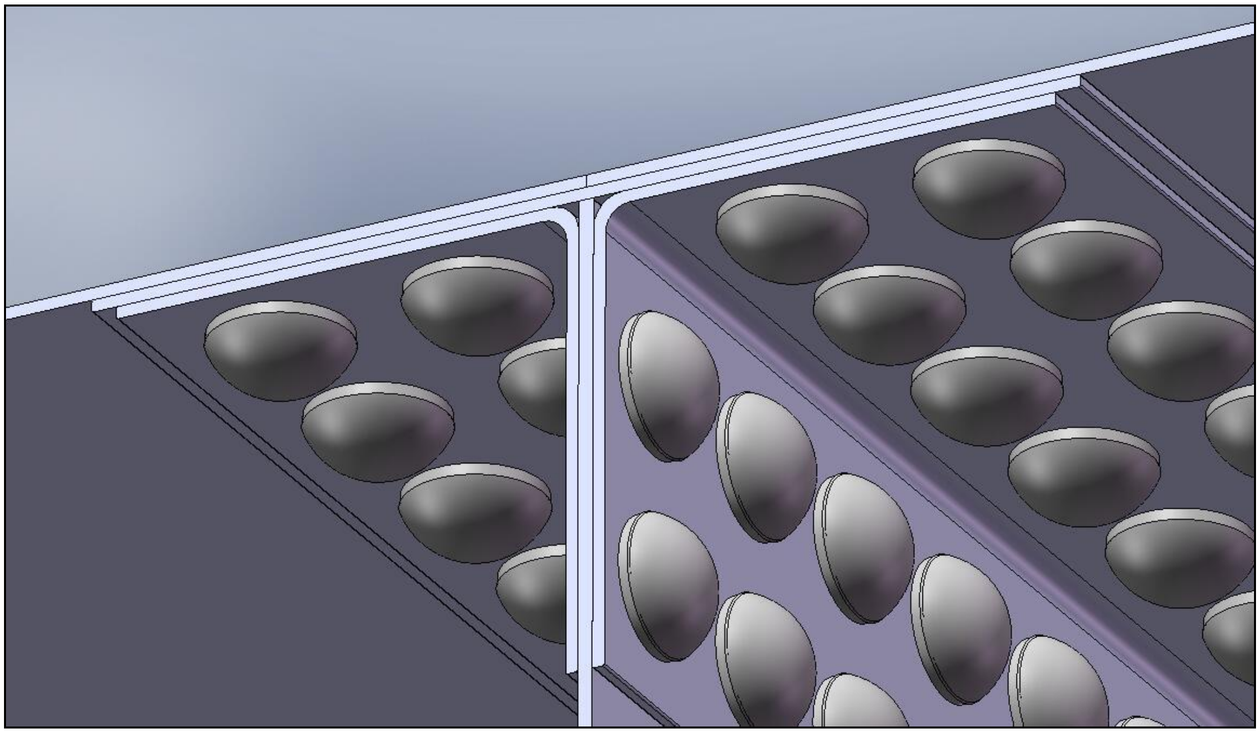
Rivet Design- Stringer to Skin



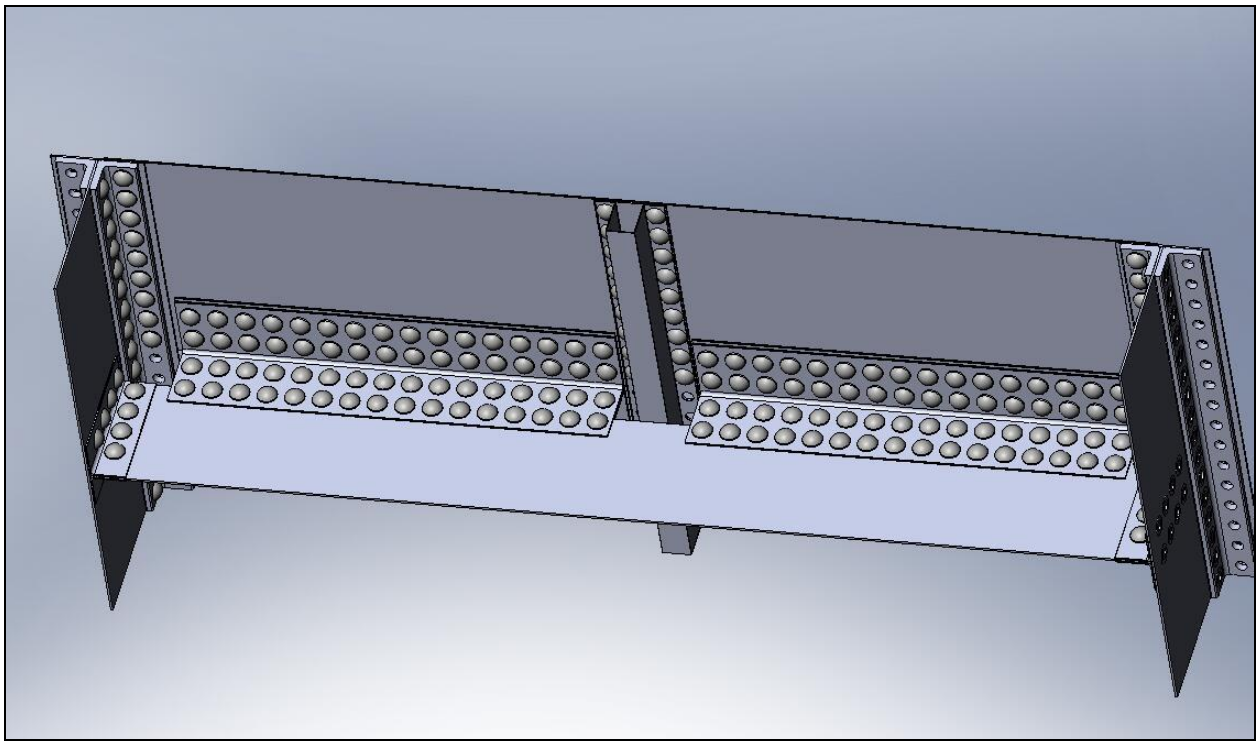


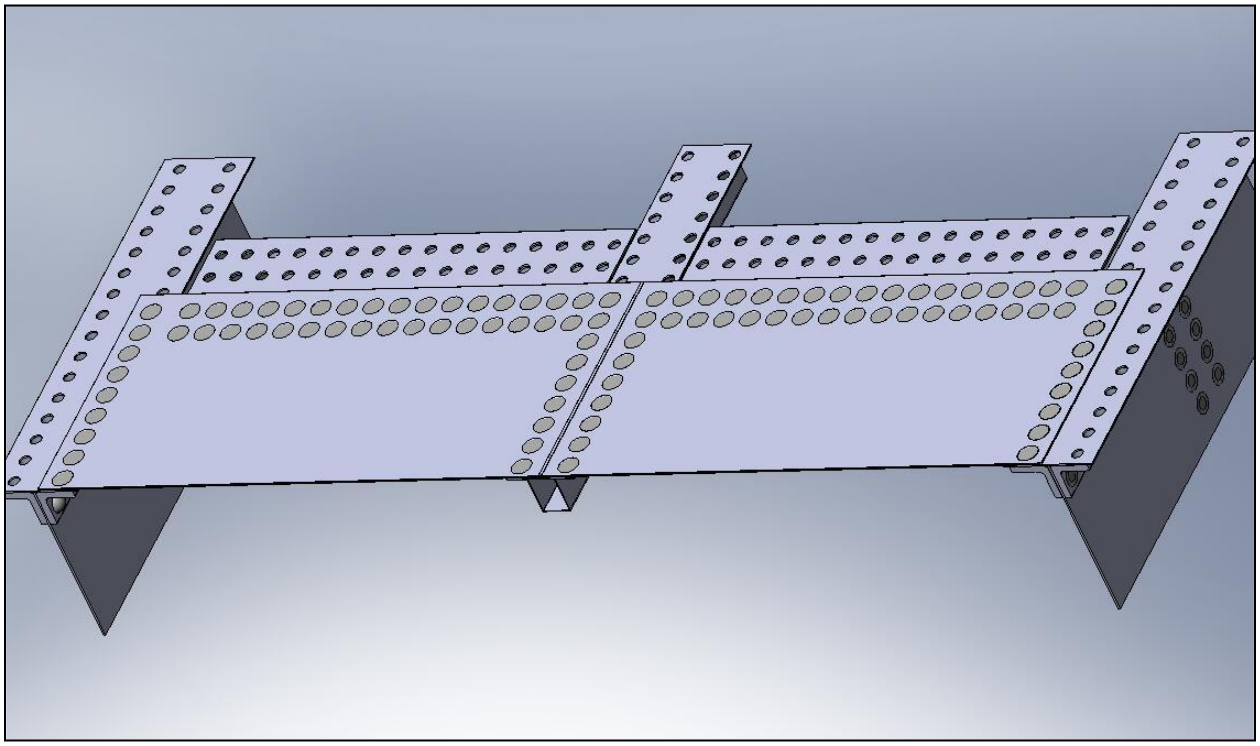
Rivet Design- Skin to Rib

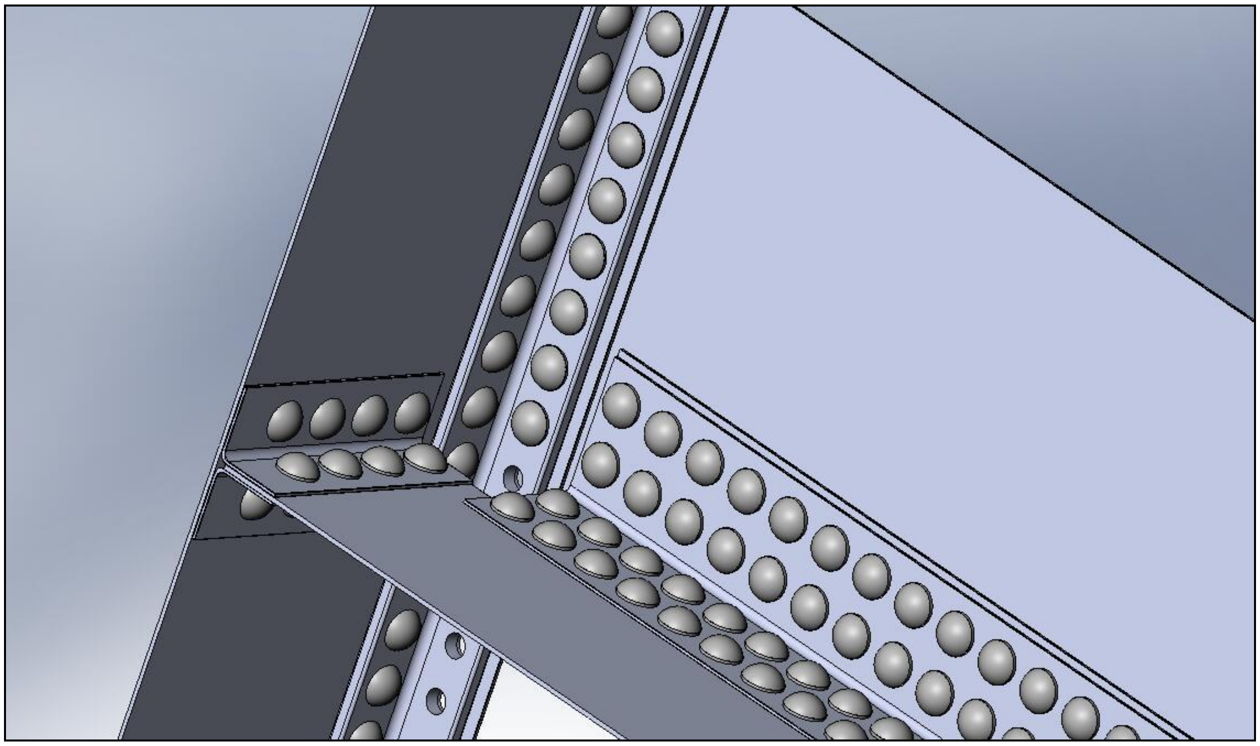


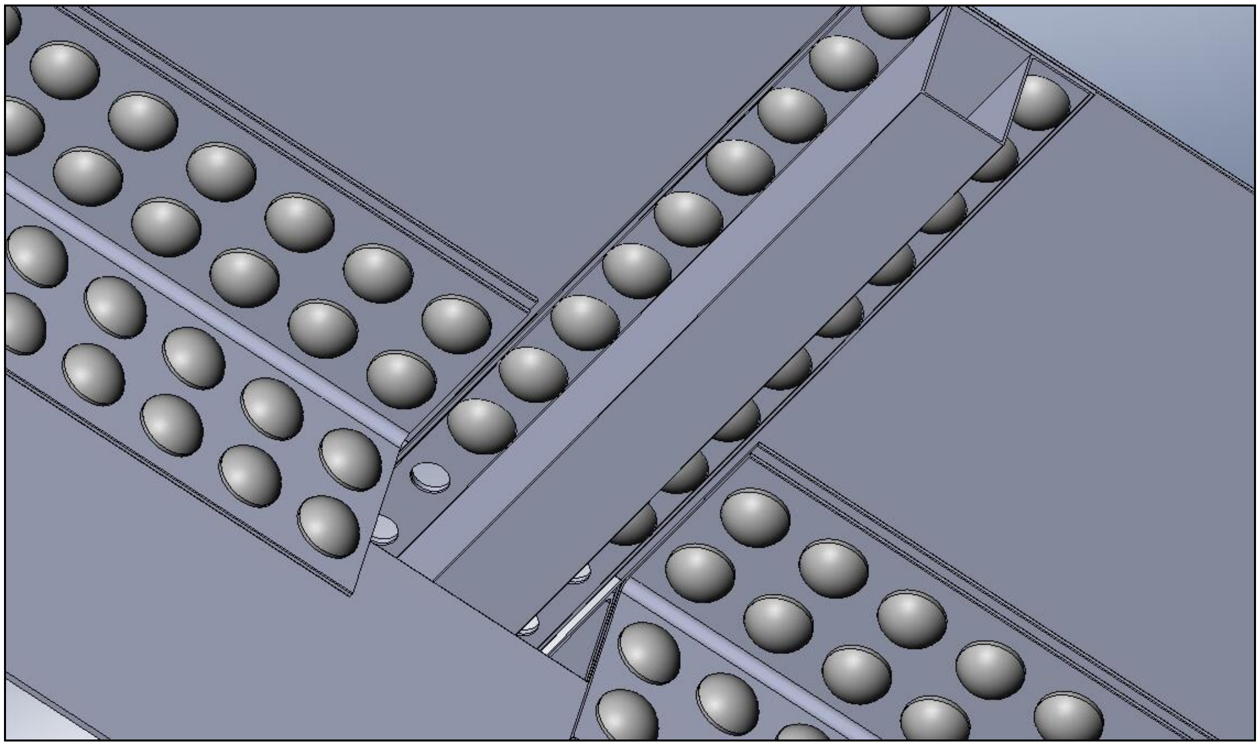


Rivet Design- Complete Set of Riveted Joints









Appendix B: Optimization Scatter Plots:

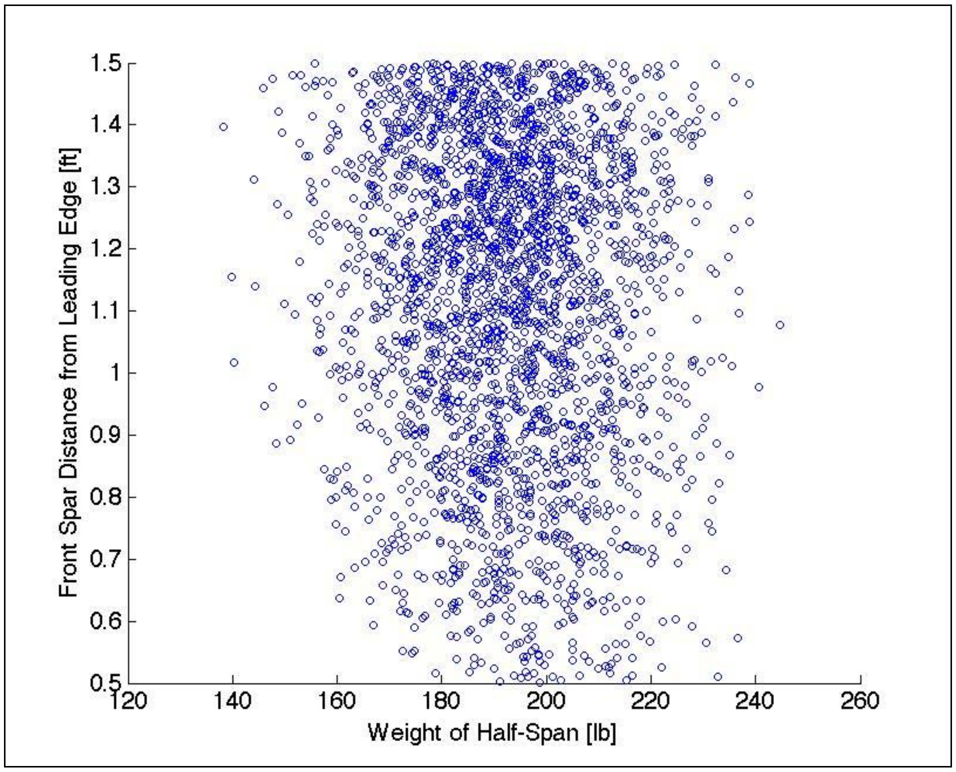


Figure B.1: Front spar distance to leading edge against half-span weight

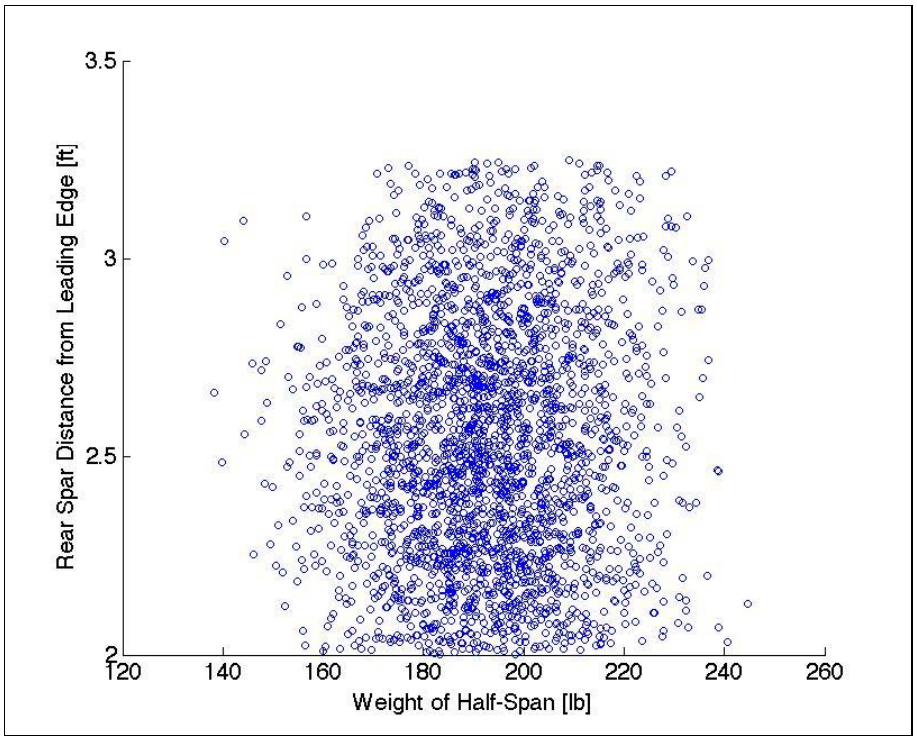


Figure B.2: Rear spar distance from leading edge vs. half-span weight

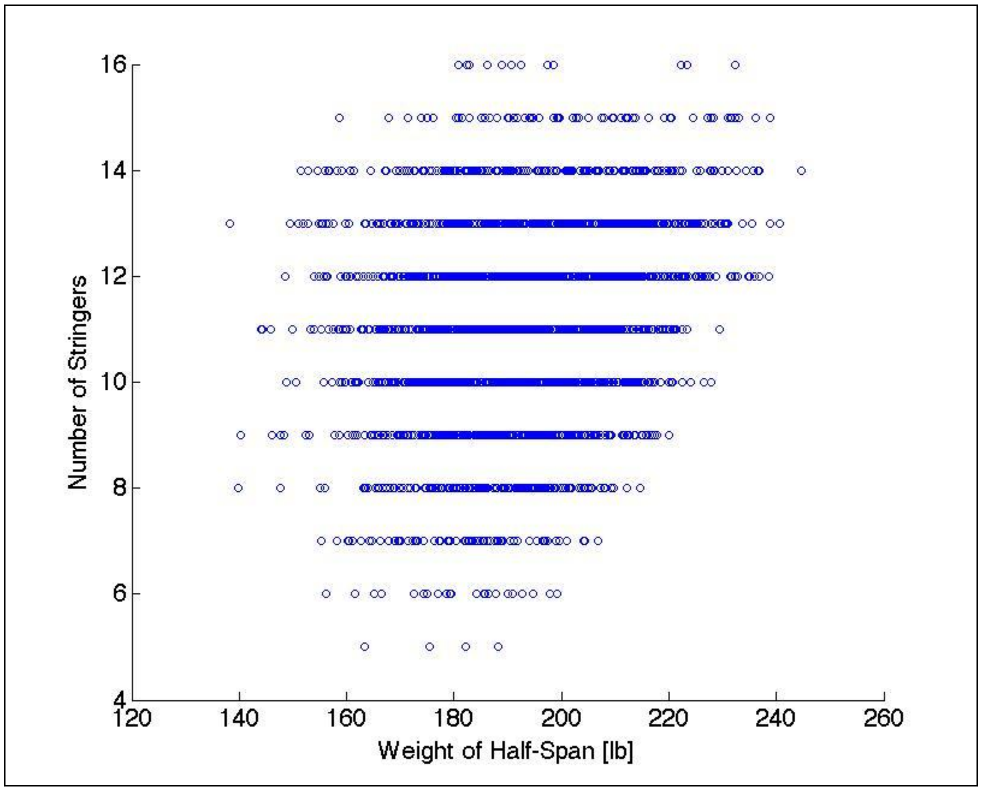


Figure B.3: Number of stringers vs. half-span weight

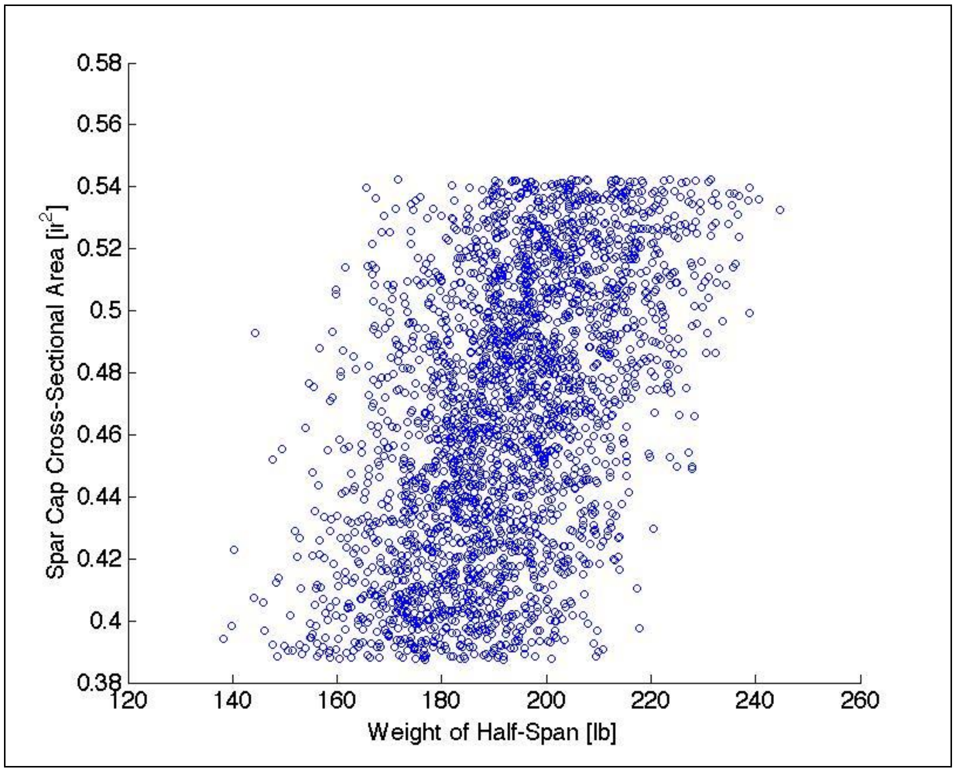


Figure B.4: Spar cap cross-sectional area vs. half-span weight

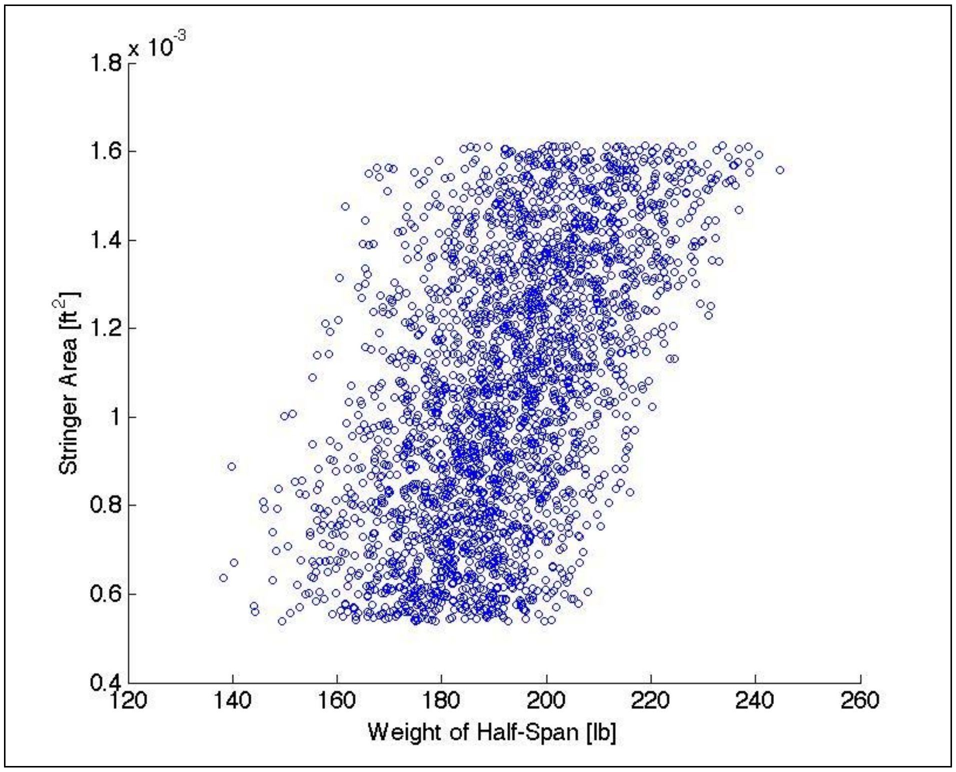


Figure B.5: Stringer cross-sectional area vs. half-span weight

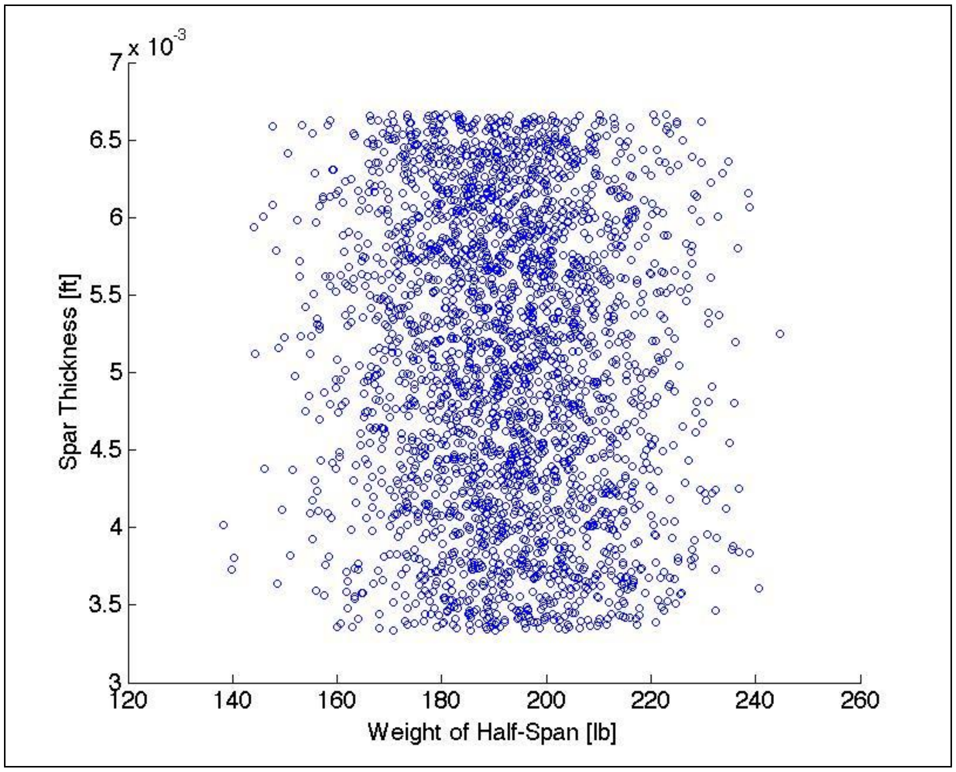


Figure B.6: Spar thickness vs. half-span weight

Appendix C: Geometry, Inertia, and Stress Correlations

Appendix C.1: Varying Number of Ribs

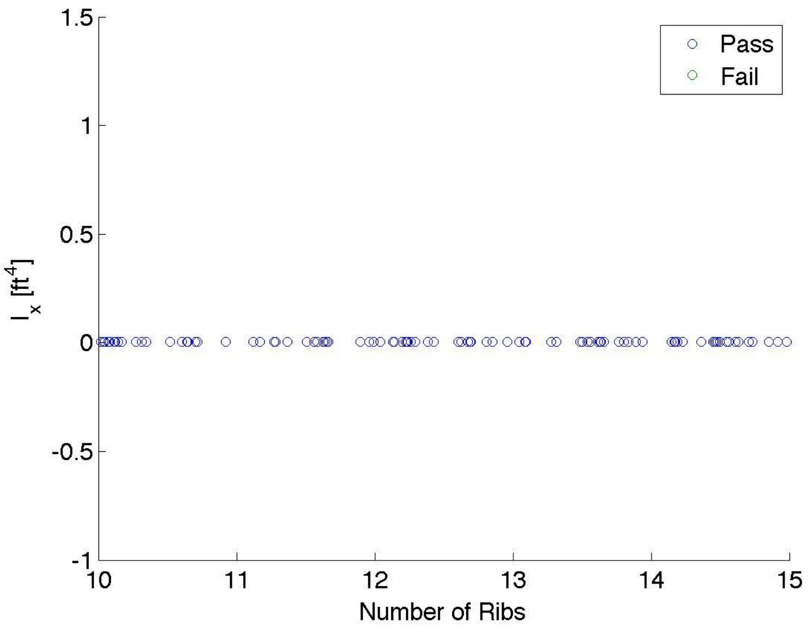


Figure C.1

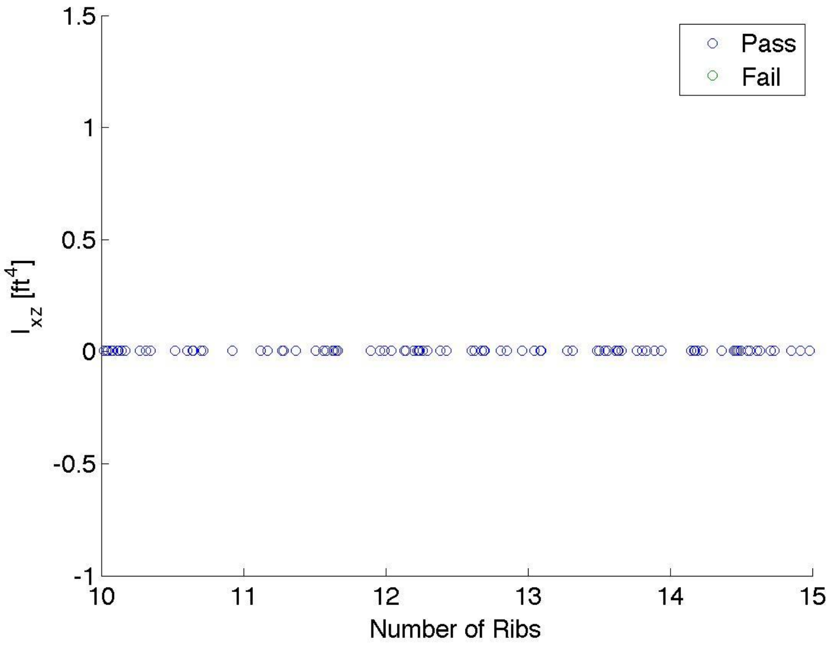


Figure C.2

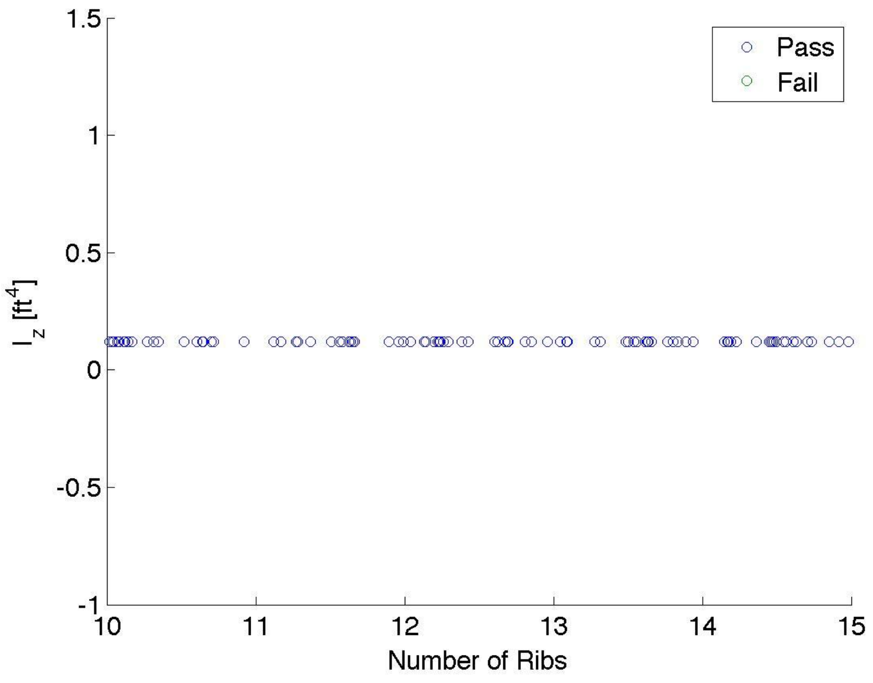


Figure C.3

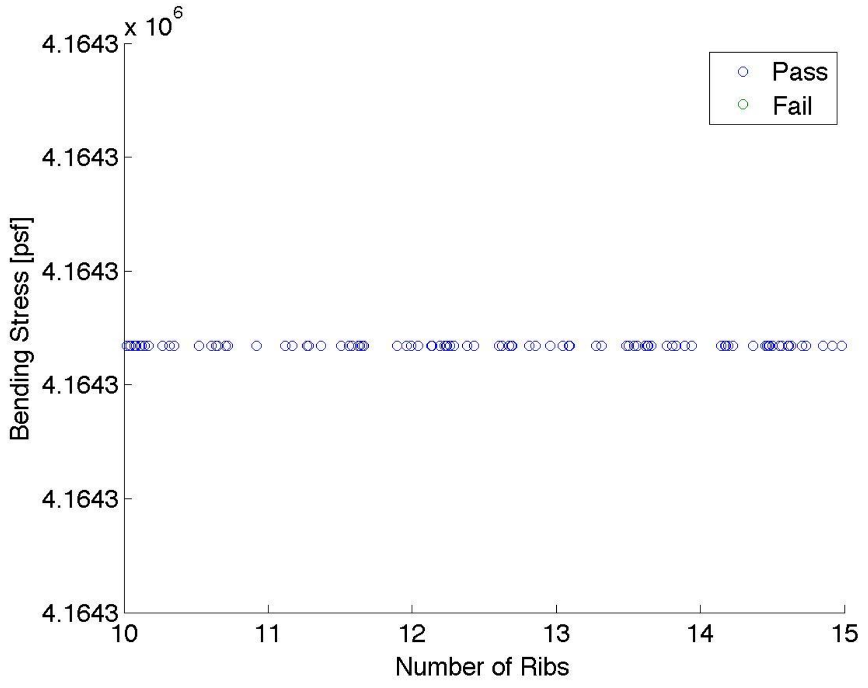


Figure C.4

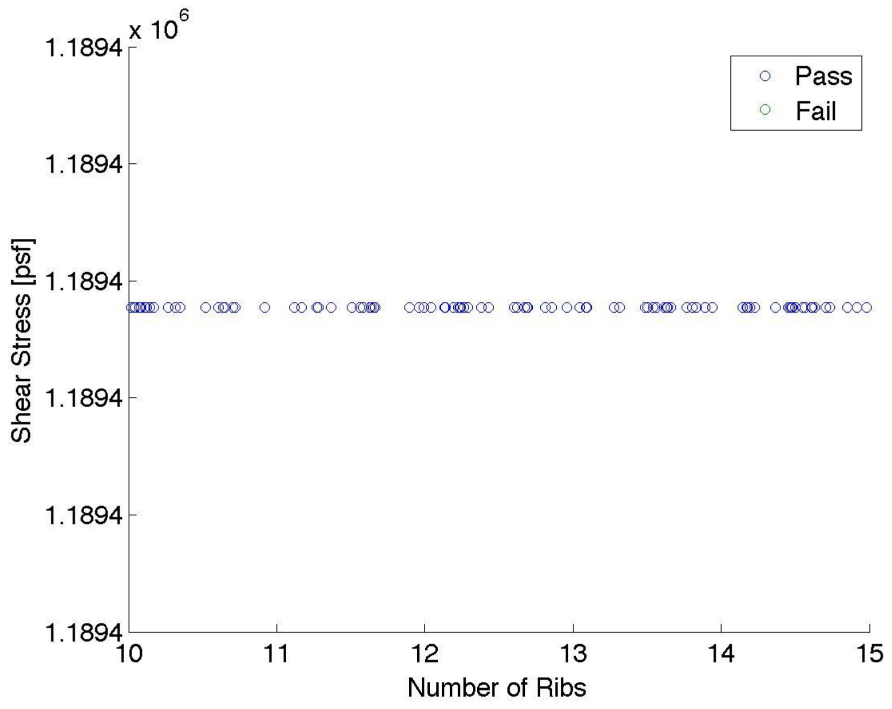


Figure C.5

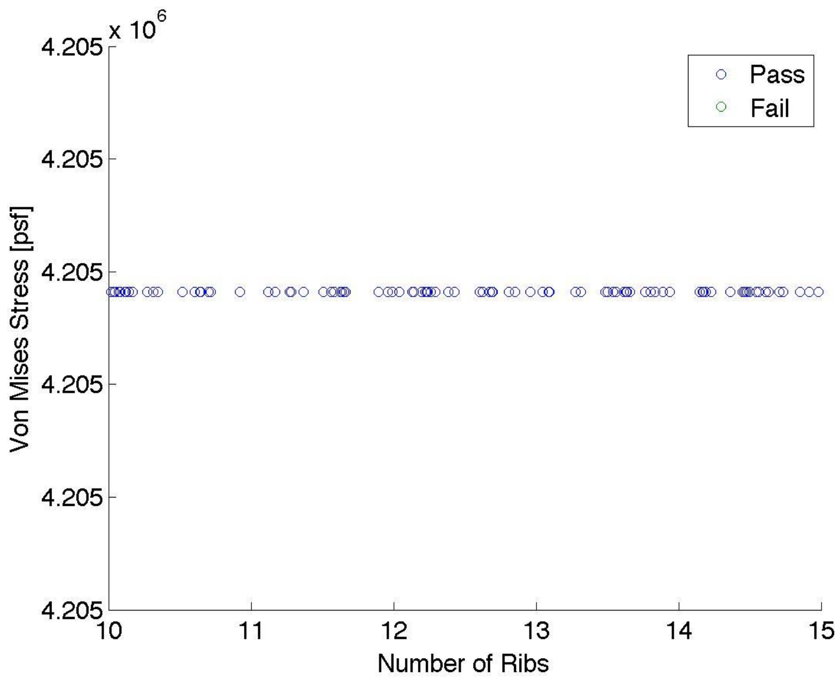


Figure C.6

Appendix C.2: Varying Spar Cap Area

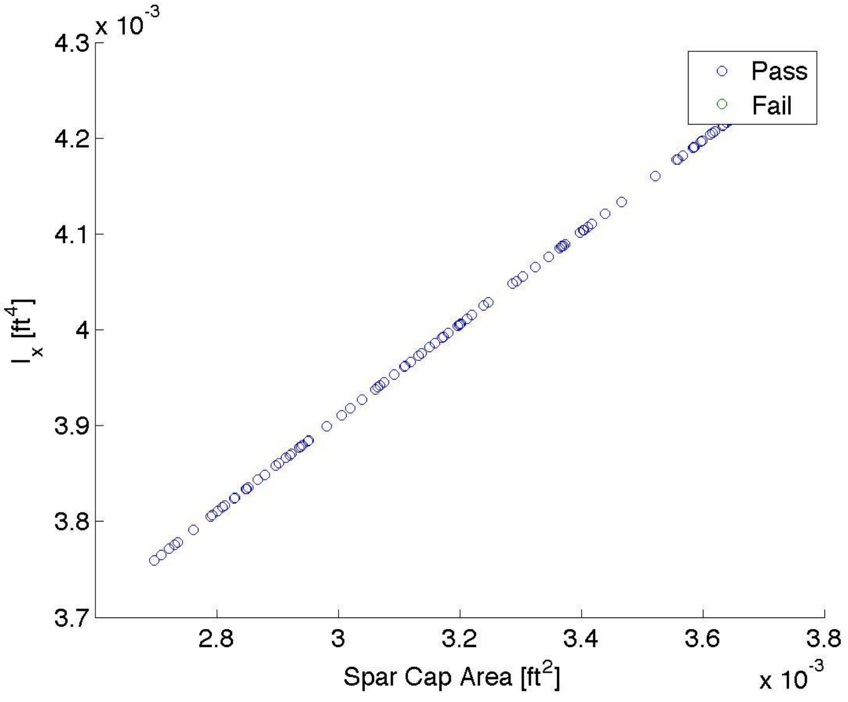


Figure C.7

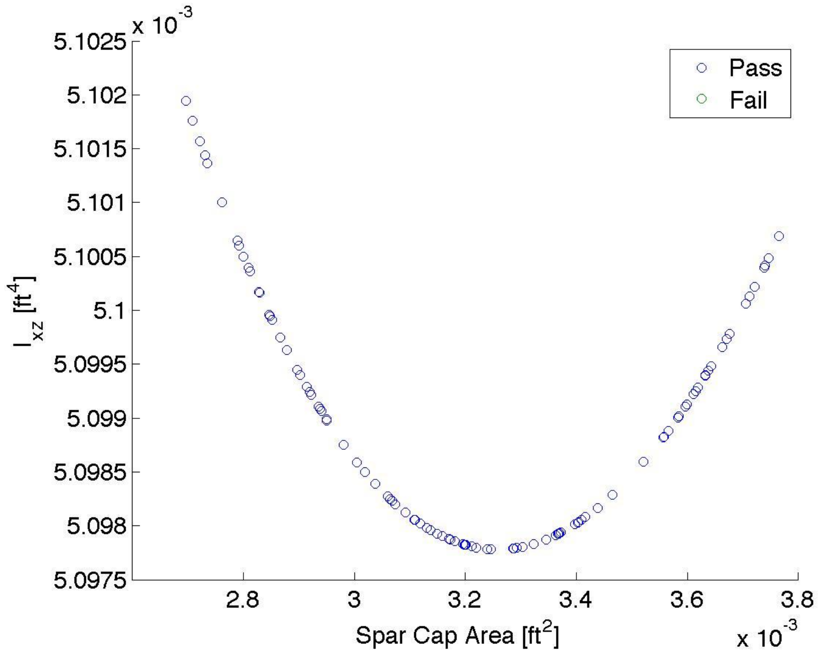


Figure C.8

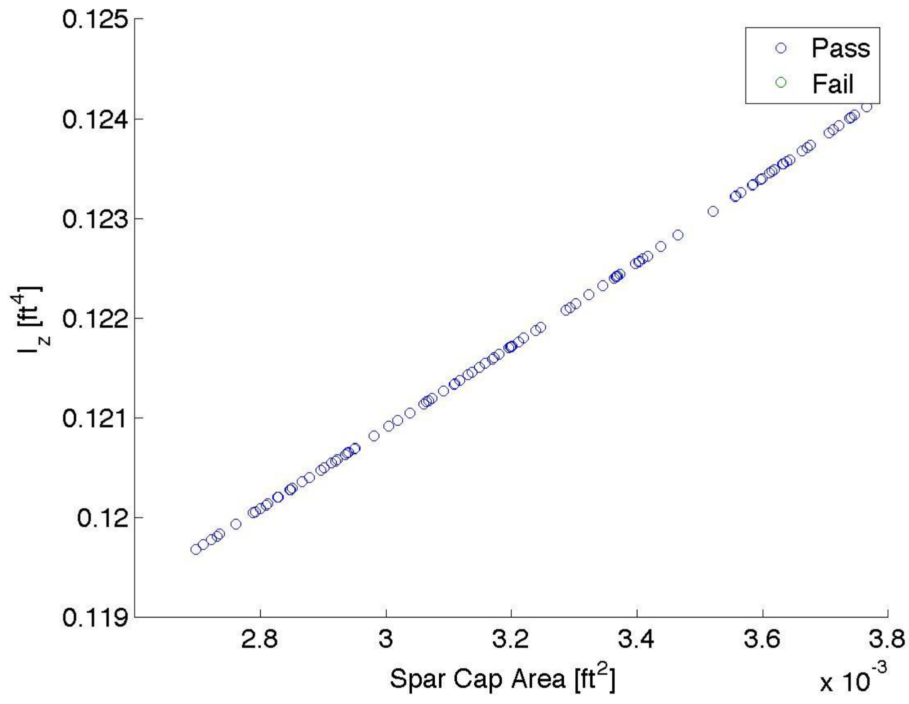


Figure C.9

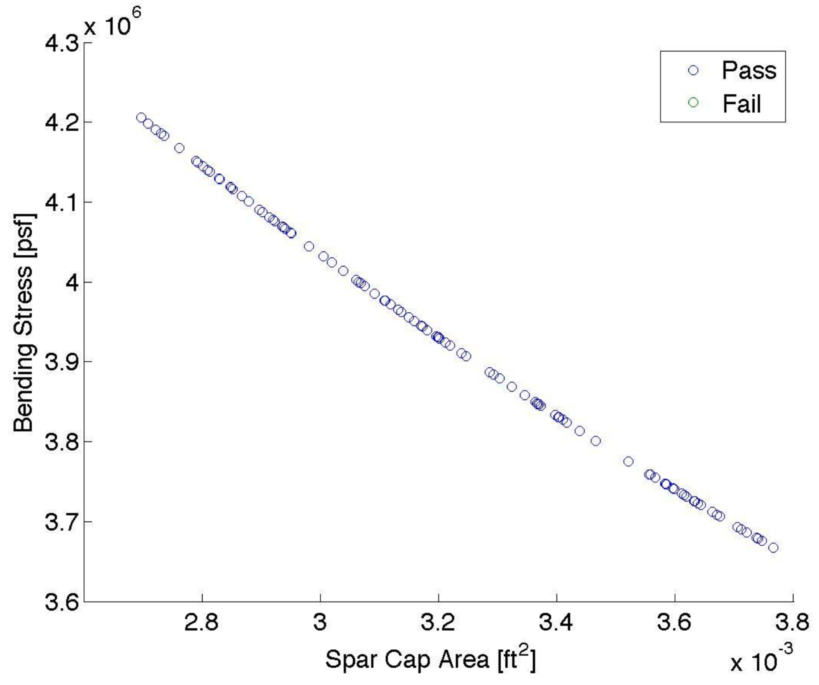


Figure C.10

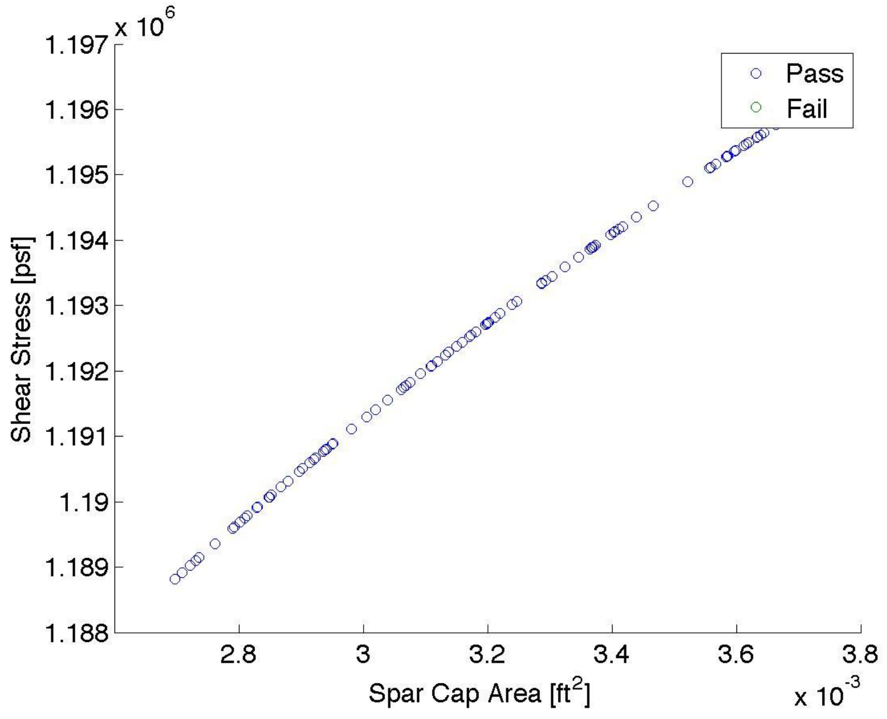


Figure C.11

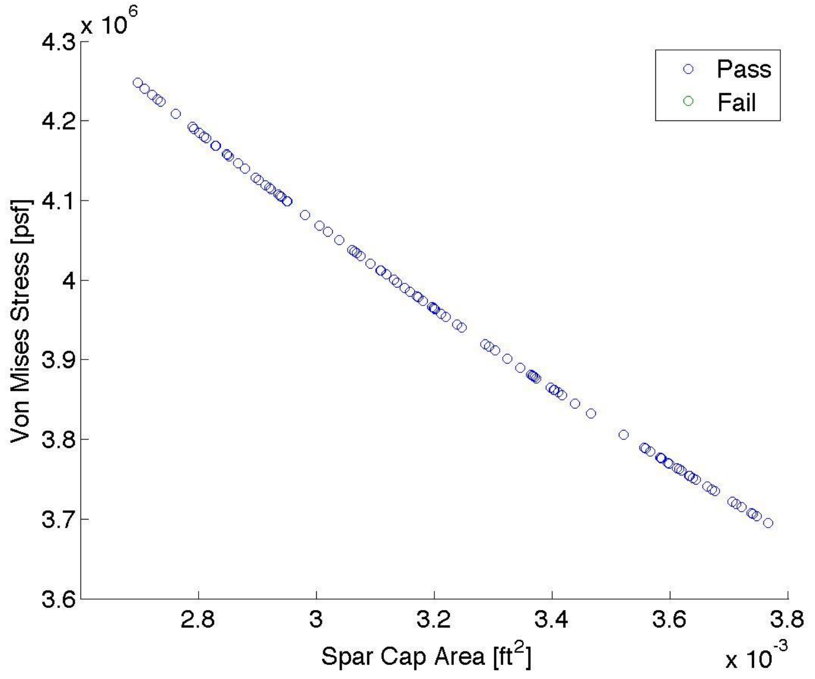


Figure C.12

Appendix C.3: Varying Stringer Area

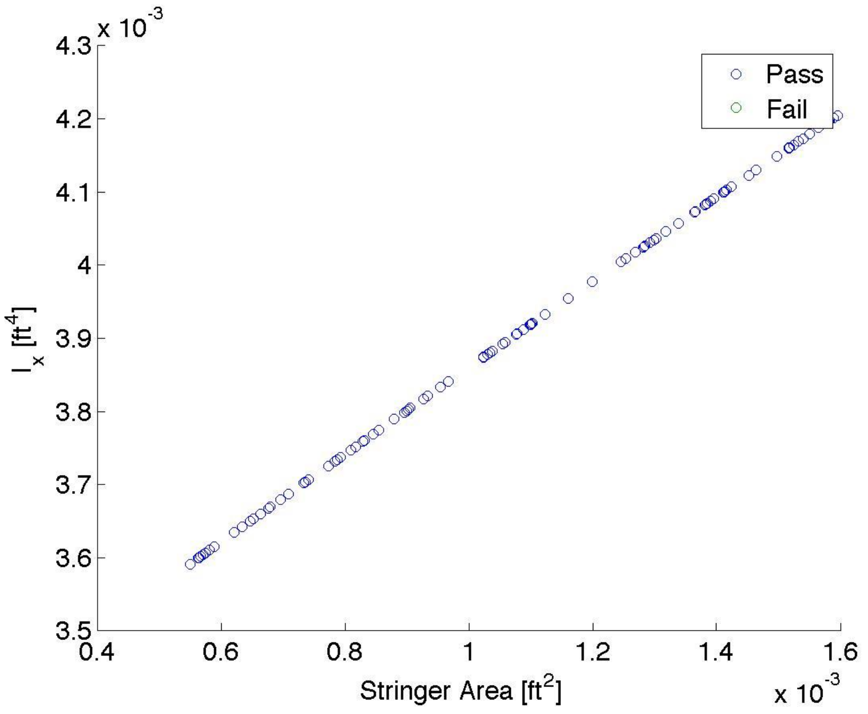


Figure C.13

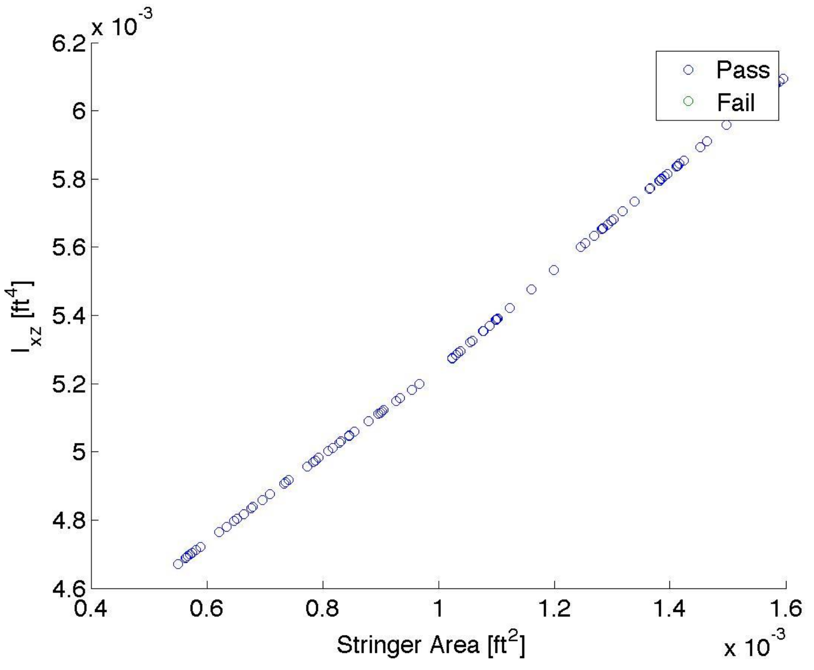


Figure C.14

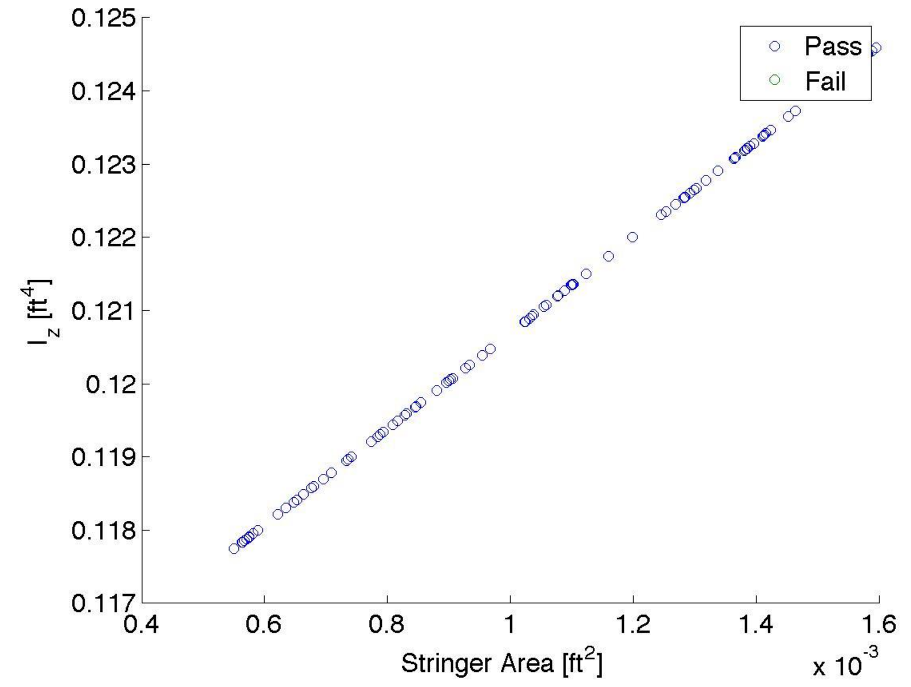


Figure C.15

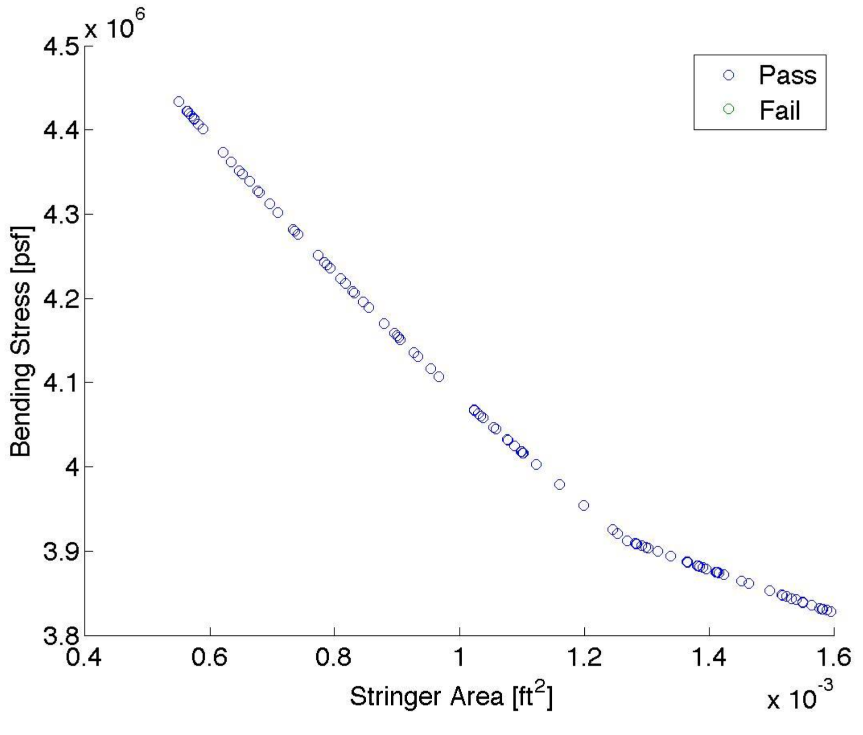


Figure C.16

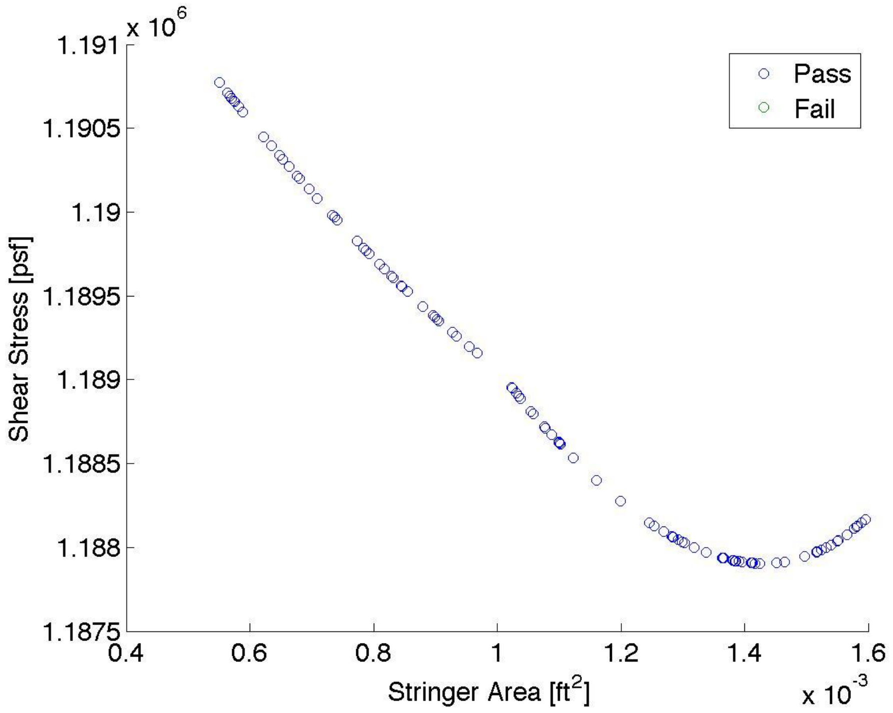


Figure C.17

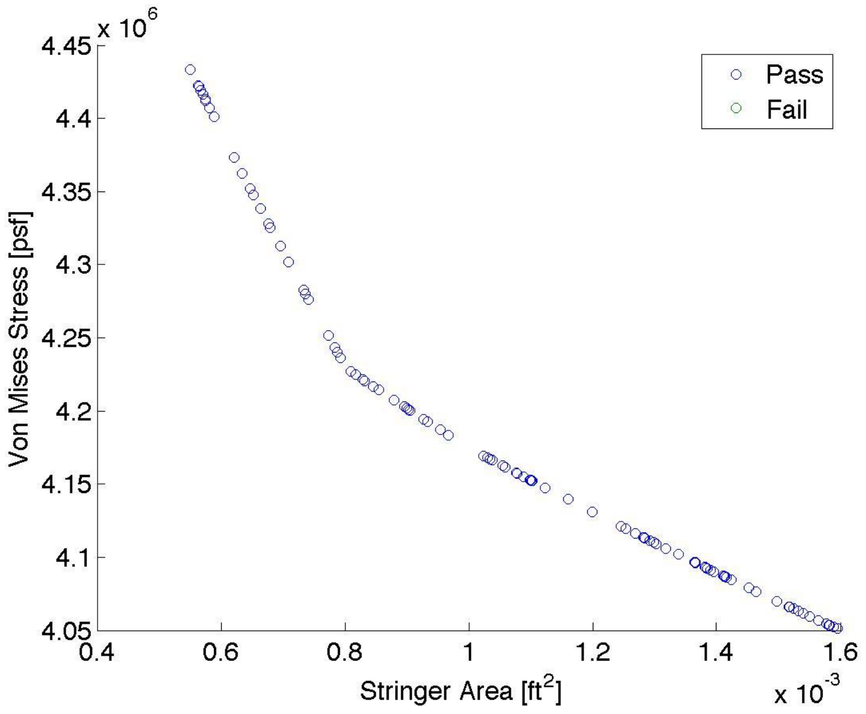


Figure C.18

Appendix C.4: Varying Skin Thickness

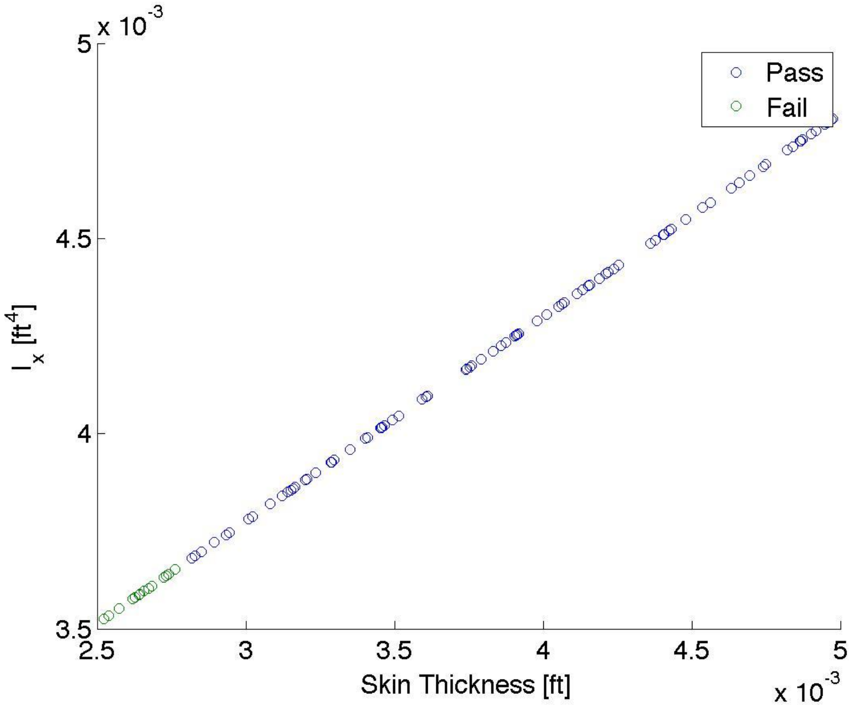


Figure C.19

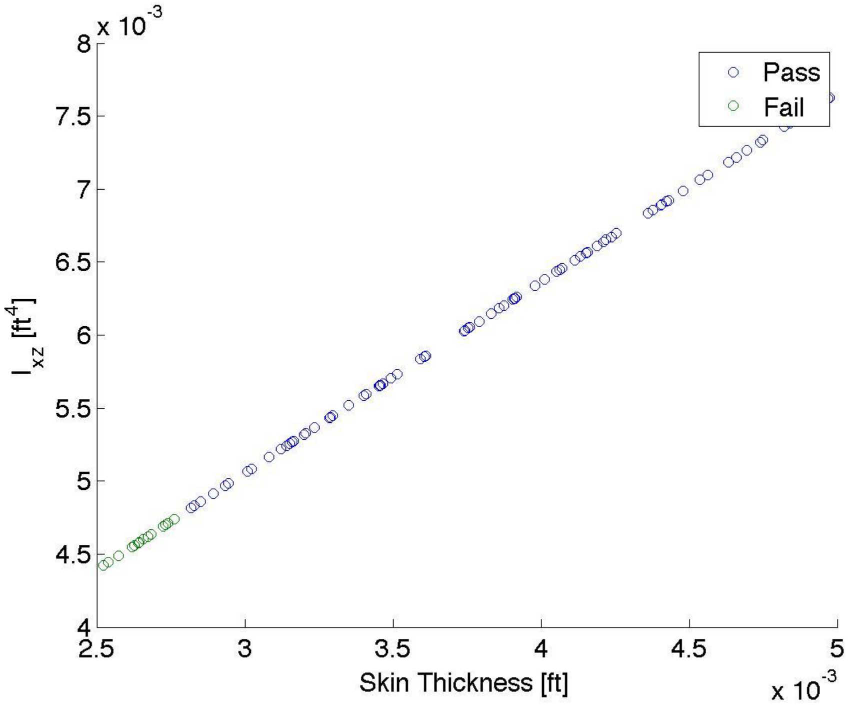


Figure C.20

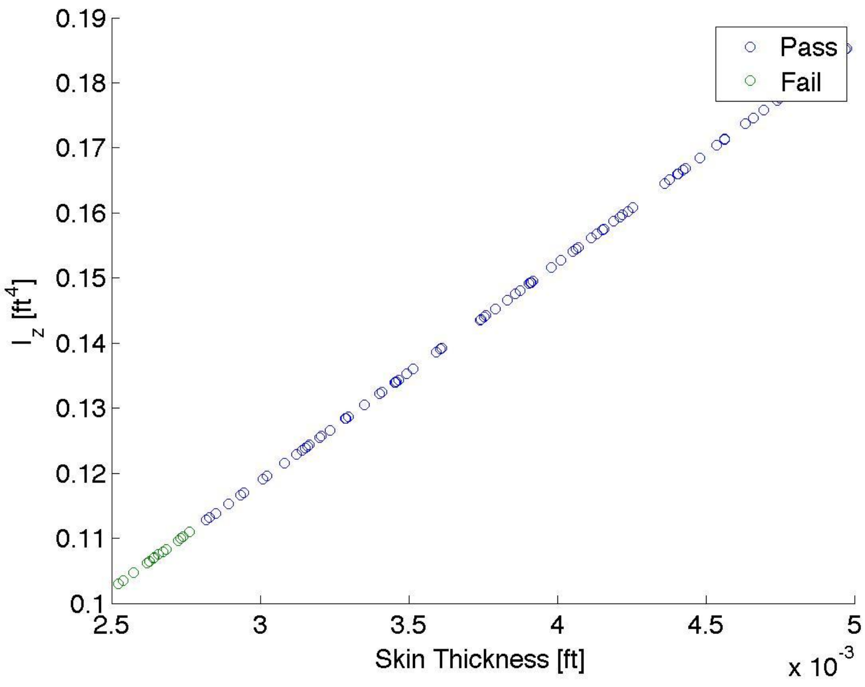


Figure C.21

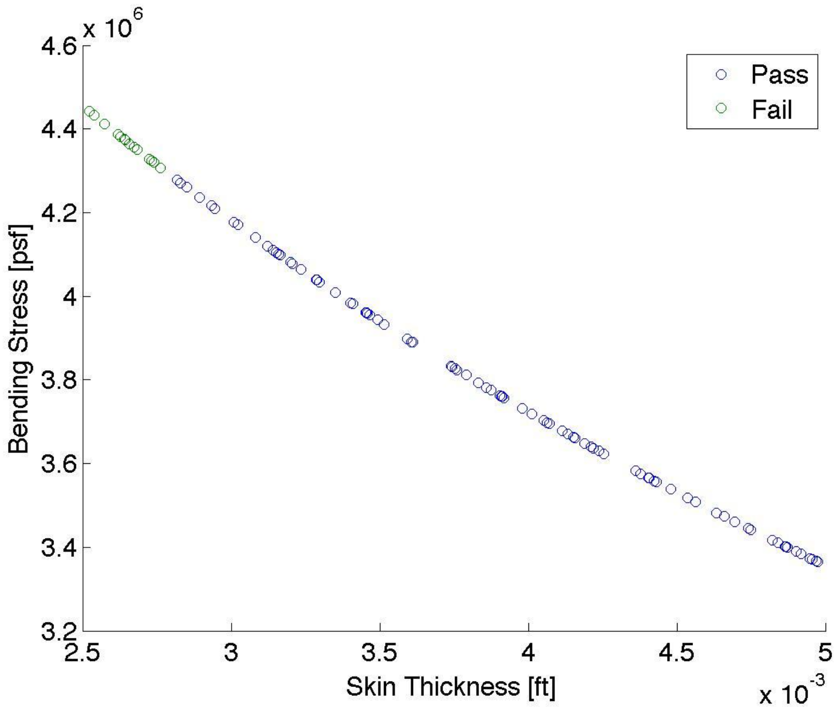


Figure C.22

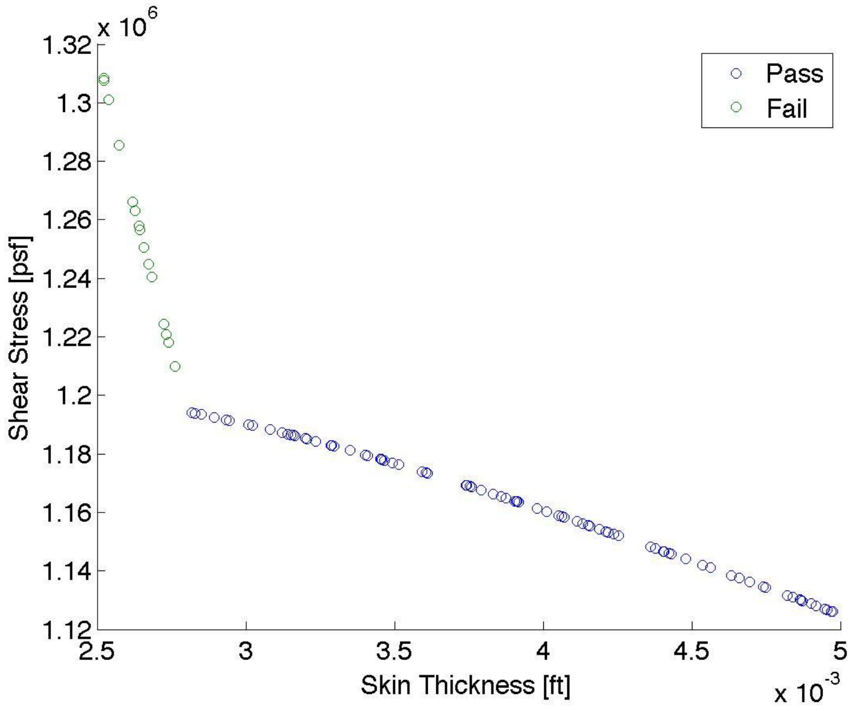


Figure C.23

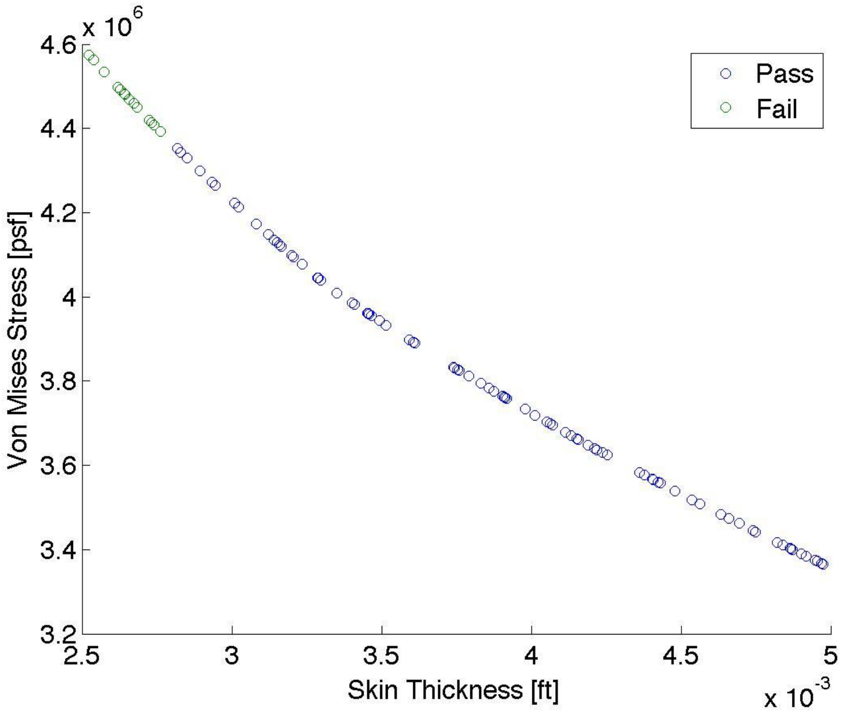


Figure C.24

Appendix C.5: Varying Spar Thickness

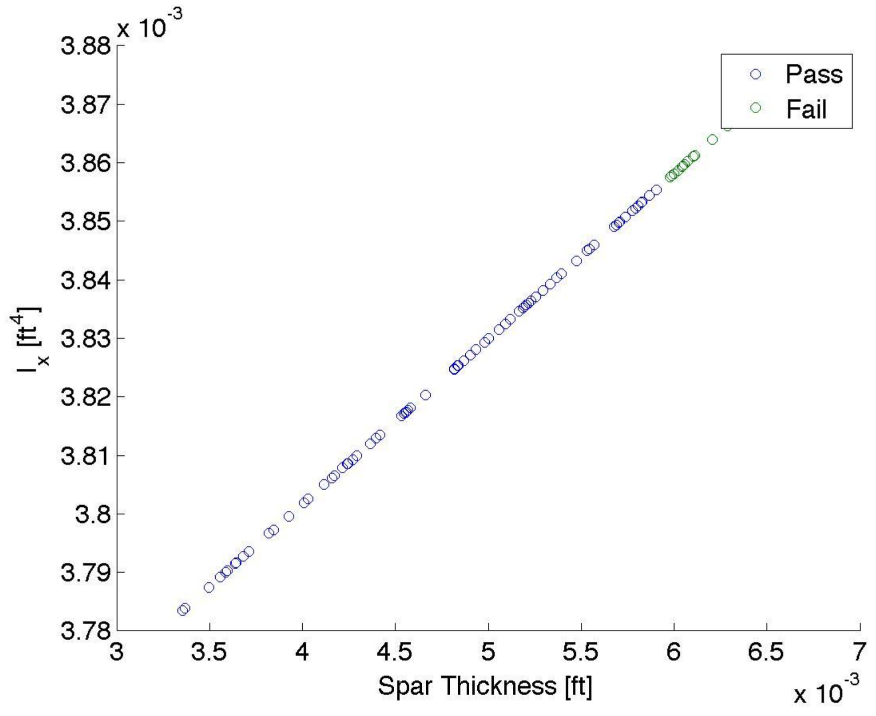


Figure C.25

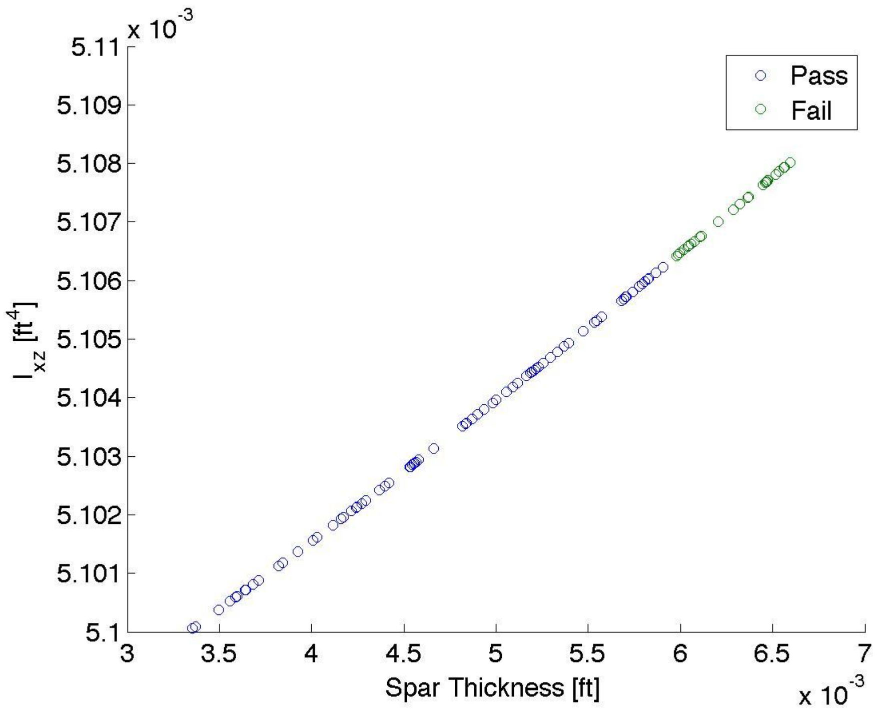


Figure C.26

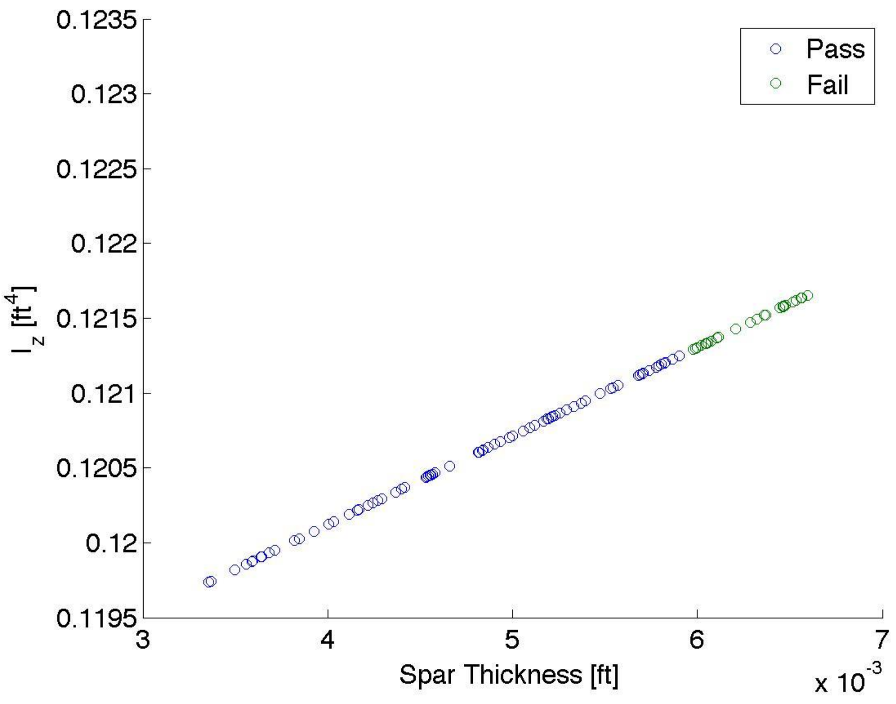


Figure C.27

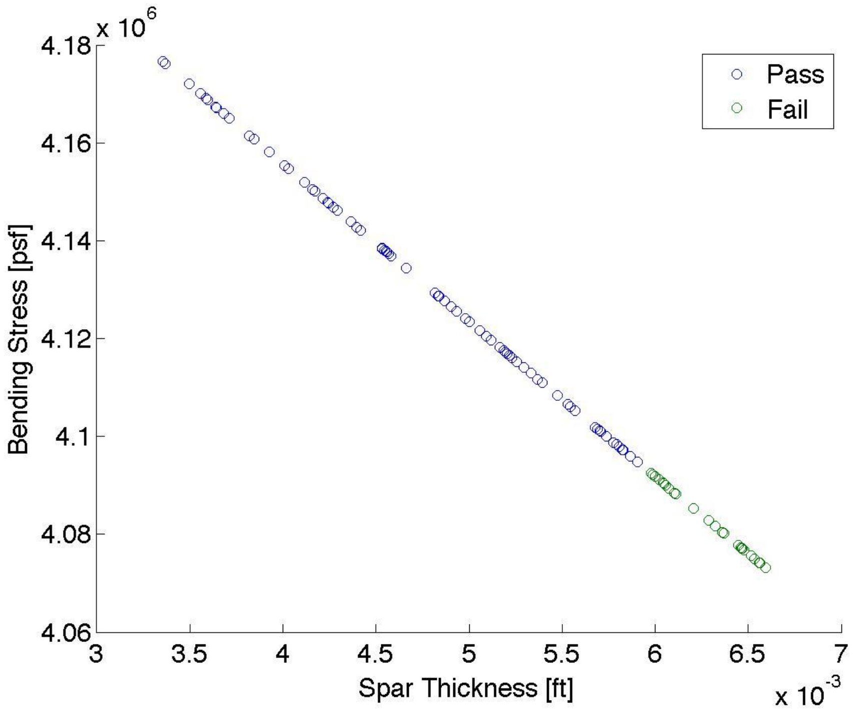


Figure C.28

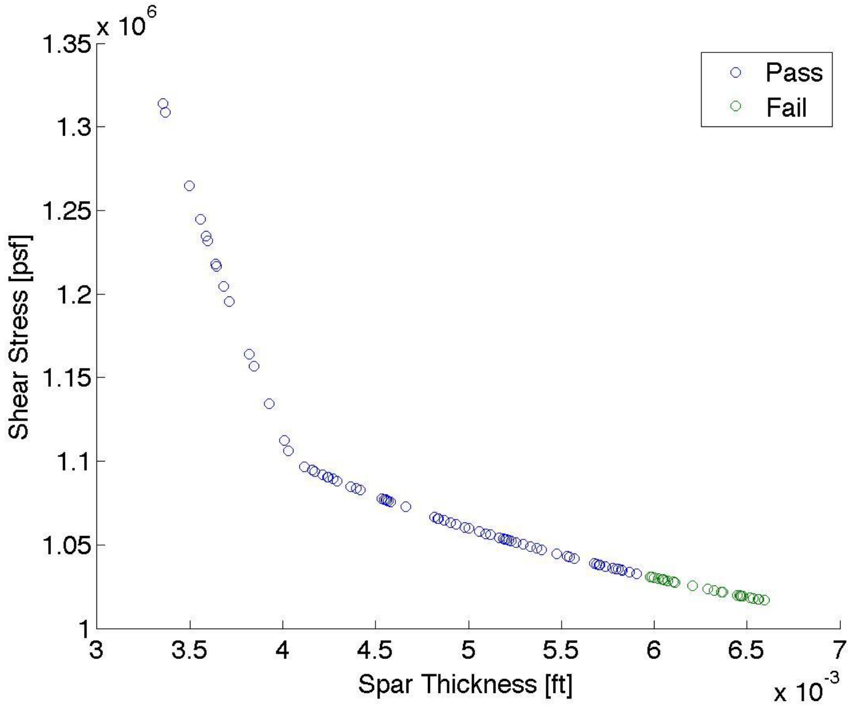


Figure C.29

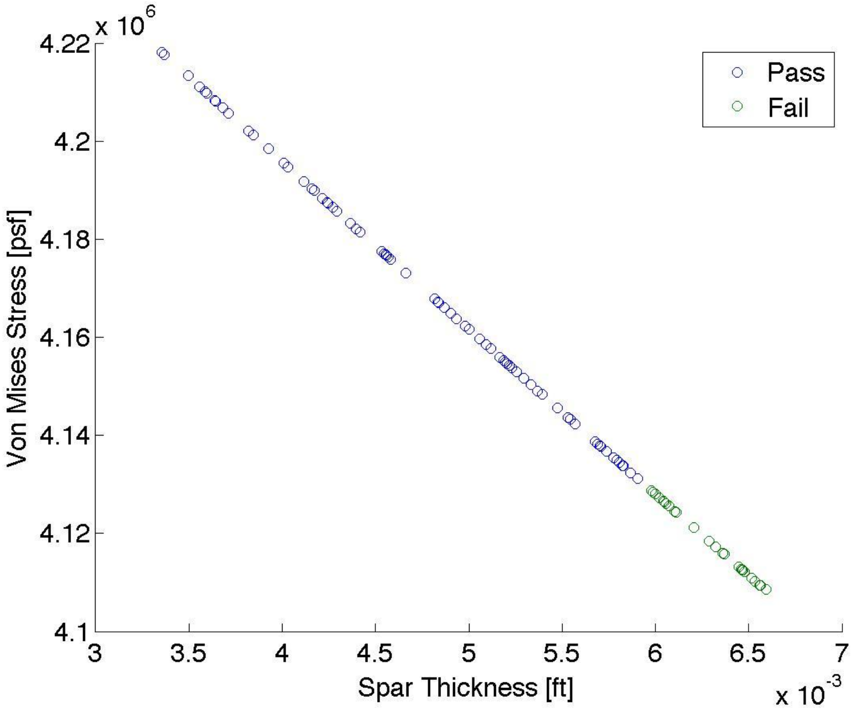


Figure C.30

Appendix C.6: Varying Front Spar Location

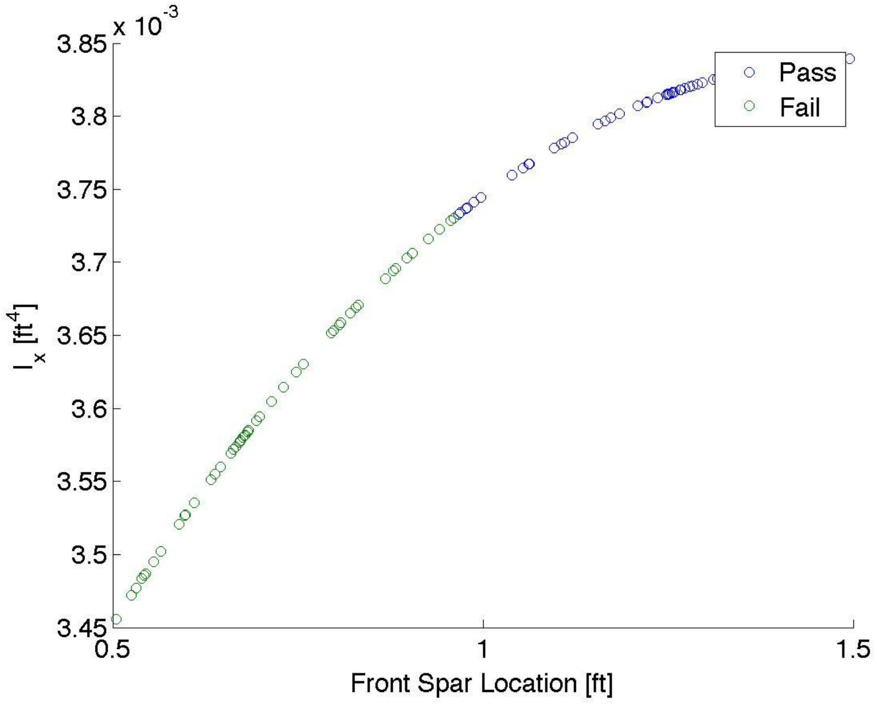


Figure C.31

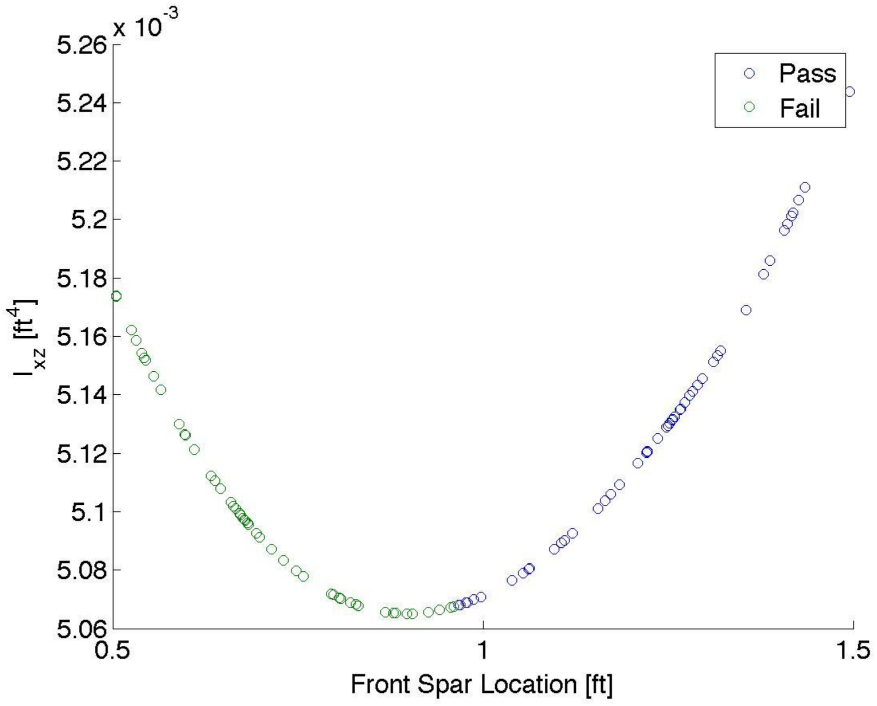


Figure C.32

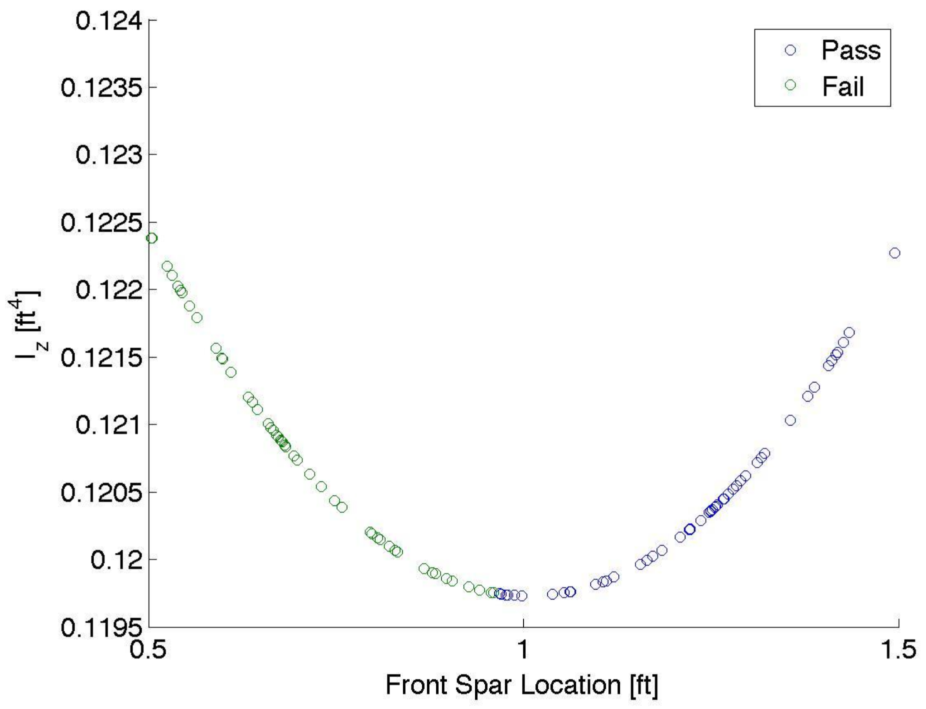


Figure C.33

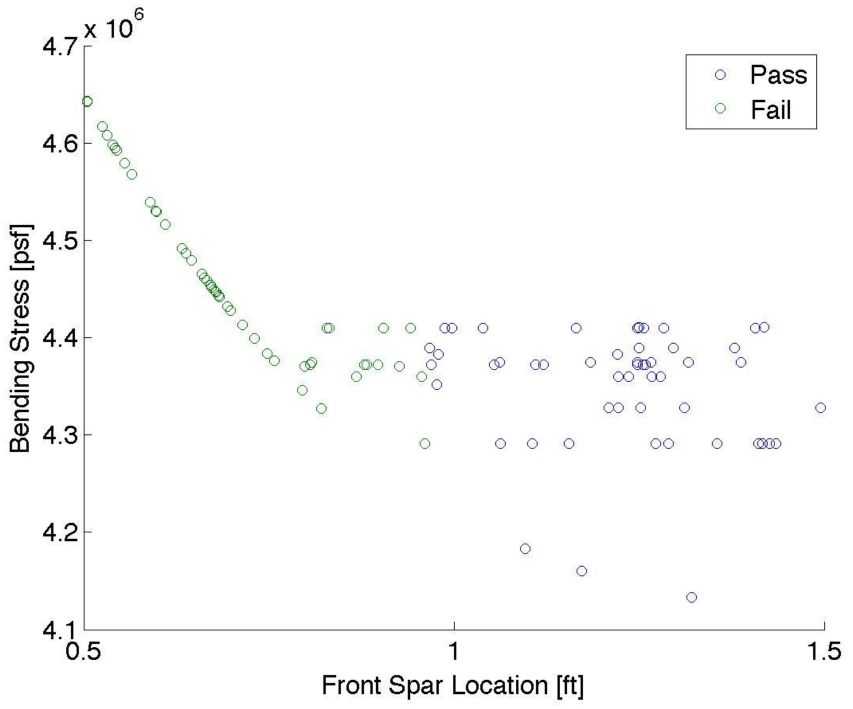


Figure C.34

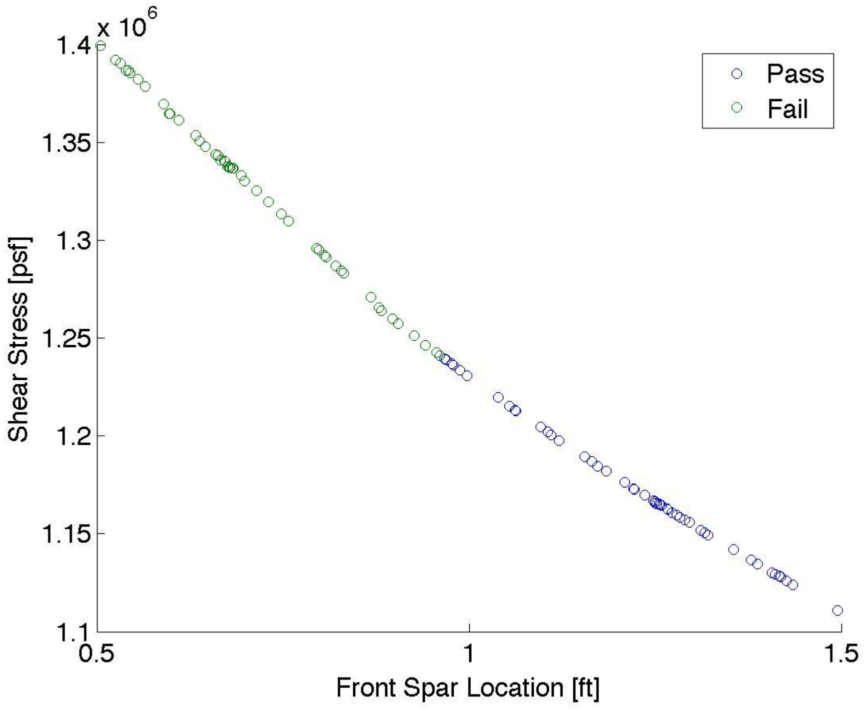


Figure C.35

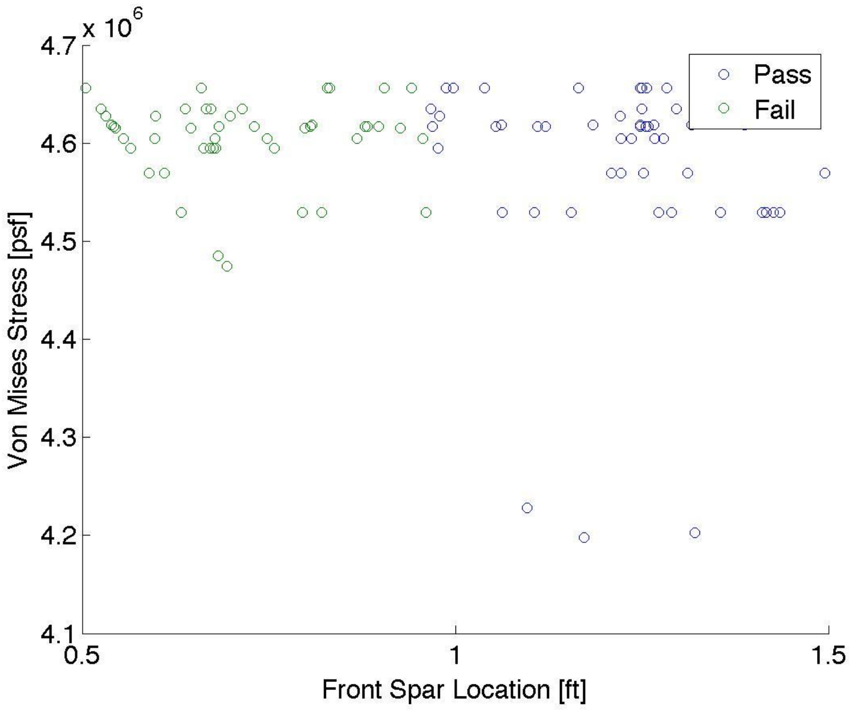


Figure C.36

Appendix C.7: Varying Number and Placement of Stringers

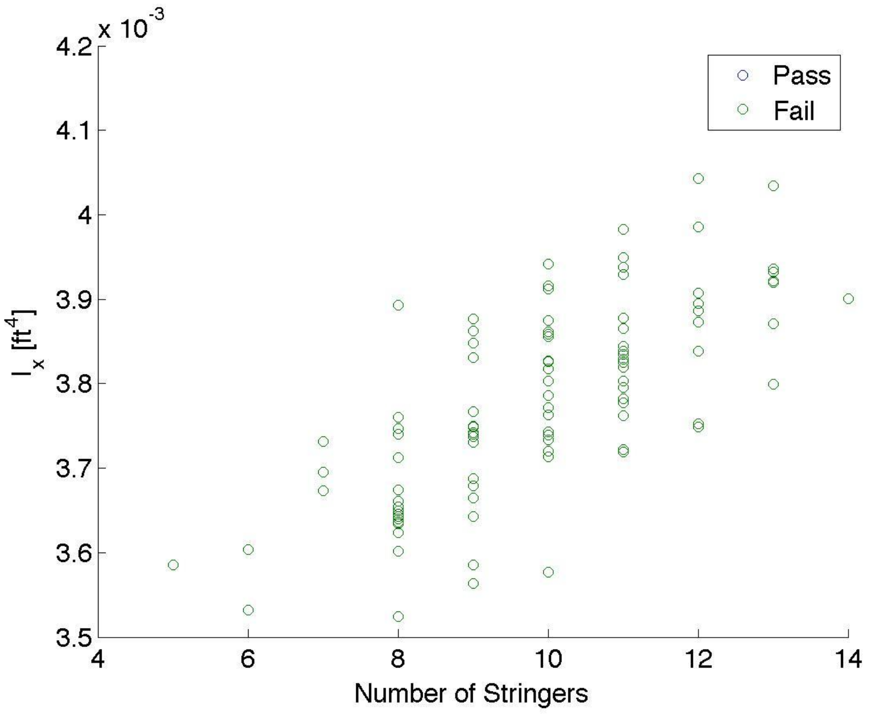


Figure C.37

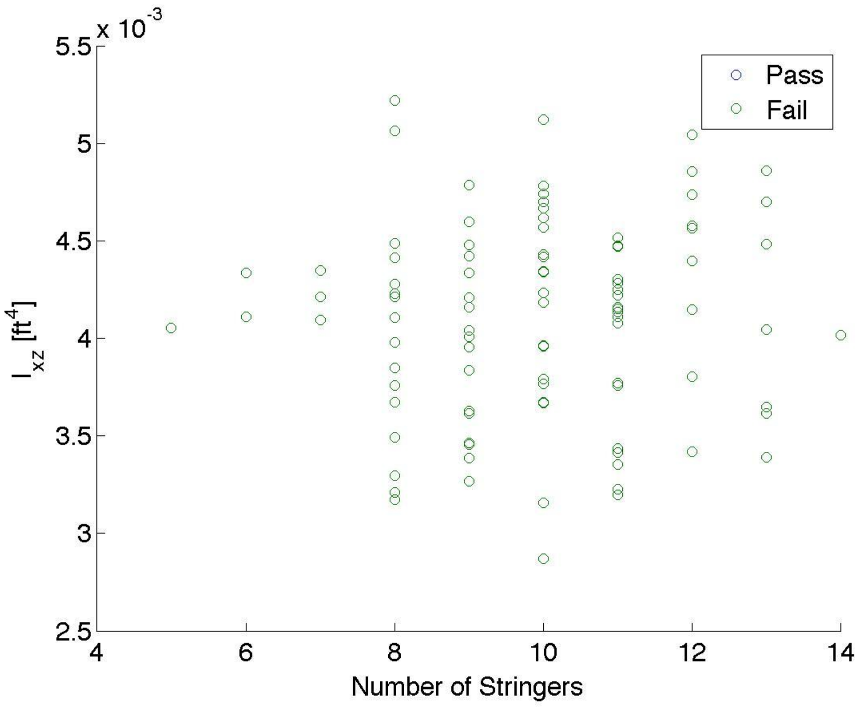


Figure C.38

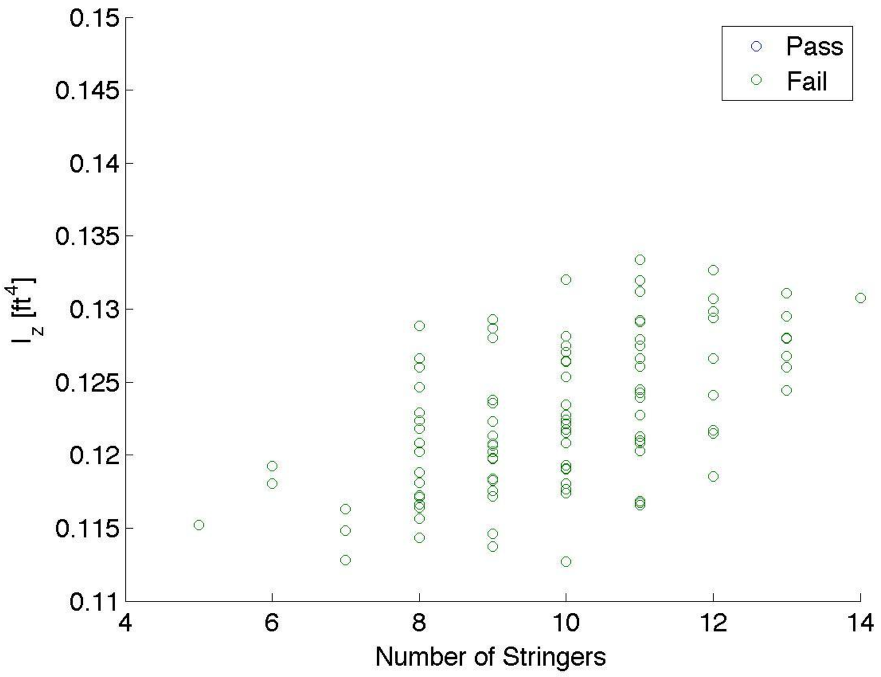


Figure C.39

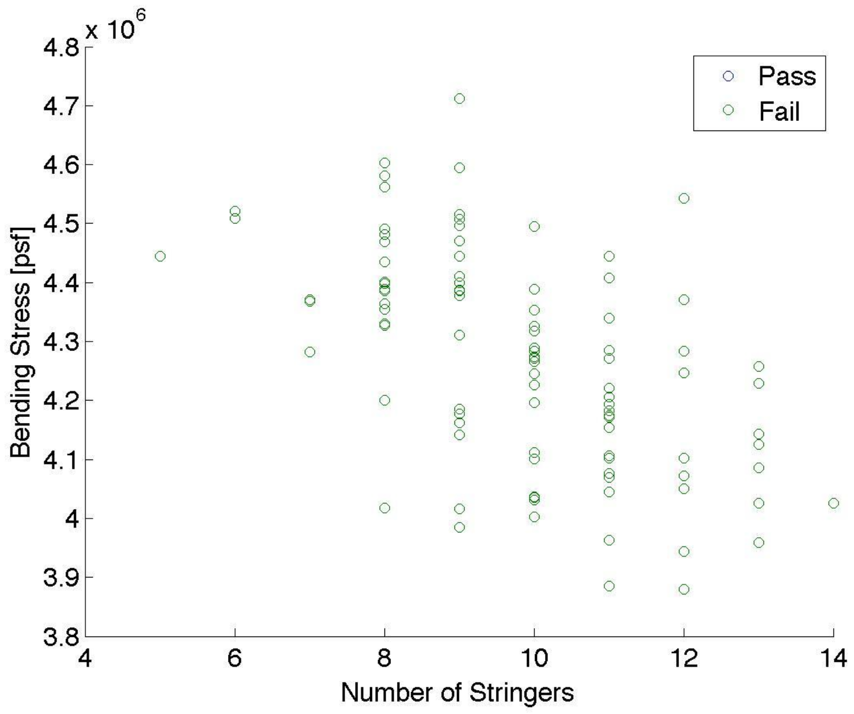


Figure C.40

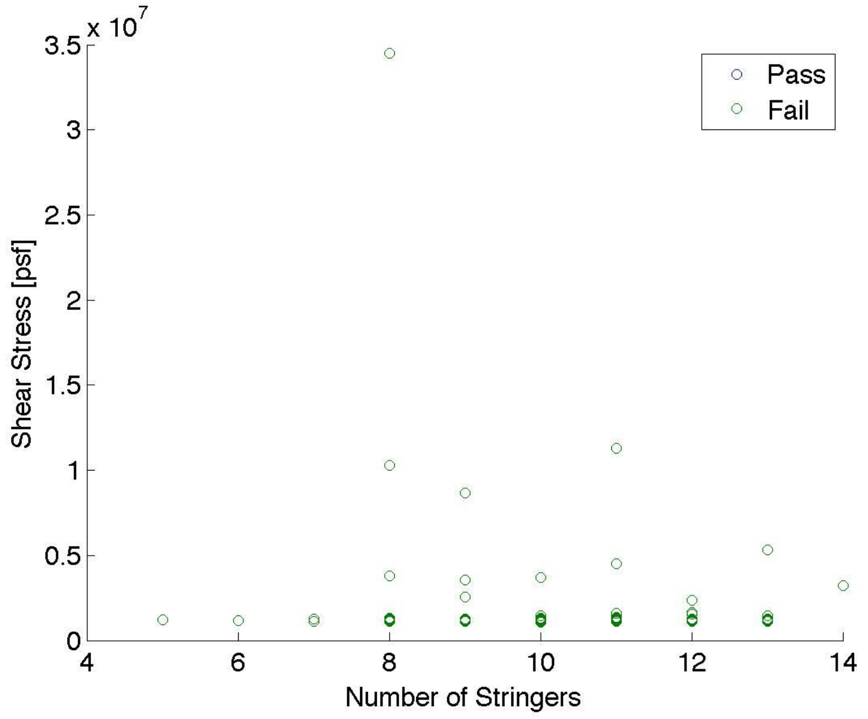


Figure C.41

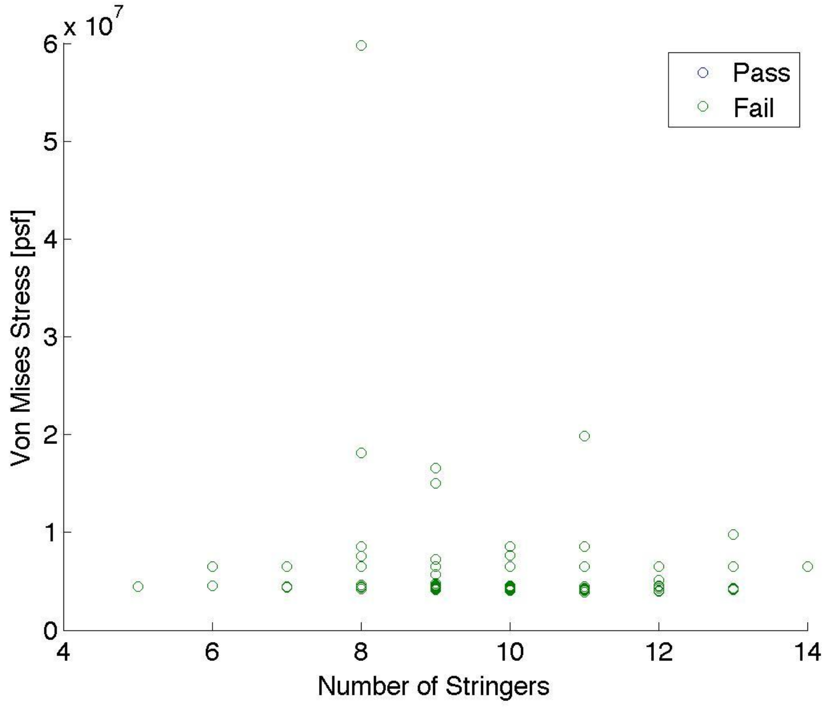


Figure C.42

Appendix D: Matlab Code

Wingsizing.m---

```
%% Monte Carlo Optimization Code

clc
clear all
close all
format short g

tic

N = 1000;

% airplane, wing and flight parameters
W = 3200;   % [lb]
b = 30;     % [ft]
c = 5;      % [ft]
Lambda = 0; % sweep back angle [rad]
h = [0 10000 25000];    % [ft]


%% material



KIc=34; %[MPa*m^0.5] (Al 2023-T3)
% KIc=29; %[MPa*m^0.5] (Al 7075-T6) (C.T.Sun, Pg 212)

C_fat=1.60e-11; %(Al 2023-T3)
m_fat=3.59;     %(Al 2023-T3)

a_crack=10e-6; %[m]

% Al 2024-T3
% E = 10600 * 1000 * 144;          % Young's modulus [psf]
% G = 4060 * 1000 * 144;           % shear modulus [psf]
% yield = 50000 * 144;             % yield stess [psf]
% FOS = 1.5;                       % factor of safety
% allow = yield / FOS;             % allowable stess [psf]

% Al 7475-T61
E = 10200 * 1000 * 144;          % Young's modulus [psf]
G = 3920 * 1000 * 144;           % shear modulus [psf]
yield = 71000 * 144;             % yield stess [psf]
FOS = 1.5;                       % factor of safety
allow = yield / FOS;             % allowable stess [psf]

%% V-n diagram
v = [247.3 396 396 187.2 287.7 396 396 217.7 369.1 396 396 279.4];  % [ft/s]
n = [4.4 4.4 -1.76 -1.76 4.4 4.4 -1.76 -1.76 4.4 4.4 -1.76 -1.76];

% NACA 2412
```

```
m = .02;
p = .4;
t = .12;
x = [0 : .01 : .75];
for i = 1 : length(x)
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;
Cla = 6.7036;%5.731;% [1/rad] from javafoil

% coordinate definition
y = [0 : .1 : b / 2];
ymin = min(y);
ymax = max(y);

% javafoil calculations
load javafoil1.mat
load javafoil2.mat
load javafoil3.mat

% preliminary calculations
S = c * b;                      % [ft^2] rectangular wing
AR = b ^ 2 / S;                 % [dim]
e = .8; % 4.61 * (1 - .045 * AR ^ .68) * cos(Lambda) ^ .15  - 3.1;
for i = 1 : 3
    RHO(i) = atm(h(i));         % [sl/ft^3]
end
aw = Cla / (1 + Cla / (pi * AR * e));   % [1/rad] wing lift curve slope
aw = aw * pi / 180;             % [1/deg]

% 12 conditions

for j = 1 : length(n)
    if j <= 4
        rho = RHO(1);
        % javafoil data
        ALPHA = javafoil1(:, 1);
        CDfoil = javafoil1(:, 3);
        CMfoil = javafoil1(:, 4);
    elseif j > 4 && j <= 8
        rho = RHO(2);
        % javafoil data
        ALPHA = javafoil2(:, 1);
        CDfoil = javafoil2(:, 3);
        CMfoil = javafoil1(:, 4);
    else
        rho = RHO(3);
        % javafoil data
        ALPHA = javafoil3(:, 1);
        CDfoil = javafoil3(:, 3);
        CMfoil = javafoil1(:, 4);
    end
    L = n(j) * W;
    CL = 2 * L / (rho * v(j) ^ 2 * S);
    alpha = CL / aw - 2.1;

    % LIFT DISTRIBUTION
    % elliptical distribution
    Gamma0 = 4 * L / (pi * rho * v(j) * b);
    LiftEll(j, :) = rho * v(j) * Gamma0 * sqrt(1 - (2 .* y ./ b) .^ 2);   %
[lb/ft]
    % rectangular distribution
    for i = 1 : length(y)
        LiftRec(j, i) = L / b;                  % [lb/ft]
    end
    % total lift distribution
    for i = 1 : length(y)
        LiftLoad(j, i) = (LiftEll(j, i) + LiftRec(j, i)) / 2;
    end

    % DRAG DISTRIBUTION
    % parasite drag coefficient from xfoil
    CDpar = interp1(ALPHA, CDfoil, alpha);
    alpha = alpha * pi / 180;                % [rad]
    % induced drag coefficient
    CDi = CL ^ 2 / (pi * AR * e);
    % total drag coefficient
    CD = CDpar + CDi;
    Dp = .5 * rho * v(j) ^ 2 * S * CDpar;    % [lb]
    D = .5 * rho * v(j) ^ 2 * S * CD;        % [lb]
    for i = 1 : length(y)
        if y(i) < (.8 * ymin) || y(i) > (.8 * ymax)
            DragLoad(j, i) = 1.154 * D / b;      % [lb/ft]
        else
            DragLoad(j, i) = .9615 * D / b;      % [lb/ft]
        end
        % WING COORDINATE DISTRIBUTION
        ZLoad(j, i) = LiftLoad(j, i) * cos(alpha) + DragLoad(j, i) *
sin(alpha);
        XLoad(j, i) = -LiftLoad(j, i) * sin(alpha) + DragLoad(j, i) *
cos(alpha);
    end

    % MOMENT ABOUT Y AXIS
    CM = interp1(ALPHA, CMfoil, alpha);
    My = .5 * rho * v(j) ^ 2 * c * S / 2 * CM;   % moment on a semispan
[lb*ft]
    if j==1
        L1=L(j);
        D1=D;
        Fz1=L1*cos(alpha)+D1*sin(alpha);
        Fx1=-L1*sin(alpha)+D1*cos(alpha);
        My1=My;
    end
end
LiftLoad = fliplr(LiftLoad);
LiftEll = fliplr(LiftEll);
LiftRec = fliplr(LiftRec);
DragLoad = fliplr(DragLoad);
ZLoad = fliplr(ZLoad);
```

```
XLoad = fliplr(XLoad);

figure
hold on
grid on
box on
plot(y, LiftRec(1, :), '-b', 'LineWidth', 2)
plot(y, LiftEll(1, :), '-r', 'LineWidth', 2)
plot(y, LiftLoad(1, :), '-g', 'LineWidth', 2)
set(gca,'fontsize',16)
xlabel('Lift Distribution [lb/ft]', 'fontsize', 17)
ylabel('Span [ft]', 'fontsize', 17)
legend('Rectangular Distribution', 'Elliptical Distribution', 'Average
Distribution', 'Location', 'SouthEast')
set(gcf,'position',[150 150 800 600])

figure
subplot(2, 1, 1)
hold on
grid on
box on
plot(y, LiftLoad(1, :), '-b', 'LineWidth', 2)
plot(y, LiftLoad(2, :), '-b', 'LineWidth', 2)
plot(y, LiftLoad(3, :), '-r', 'LineWidth', 2)
plot(y, LiftLoad(4, :), '-r', 'LineWidth', 2)
set(gca,'fontsize',16)
legend('PHAA', 'PLAA', 'NHAA', 'NLAA', 'Location', 'East')
xlabel('Span [ft]', 'fontsize', 17)
ylabel('Lift Load [lb/ft]', 'fontsize', 17)
subplot(2, 1, 2)
hold on
grid on
box on
plot(y, DragLoad(1, :), '-b', 'LineWidth', 2)
plot(y, DragLoad(2, :), '-g', 'LineWidth', 2)
plot(y, DragLoad(3, :), '-r', 'LineWidth', 2)
plot(y, DragLoad(4, :), '-k', 'LineWidth', 2)
set(gca,'fontsize',16)
legend('PHAA', 'PLAA', 'NHAA', 'NLAA', 'Location', 'East')
xlabel('Span [ft]', 'fontsize', 17)
ylabel('Drag Load [lb/ft]', 'fontsize', 17)
set(gcf,'position',[150 150 800 600])

% SHEAR DISTRIBUTION
[Vx, Vz] = numInt(-XLoad, -ZLoad);      % [lb]
[Vx2, Vz2] = spanPolyInt(-XLoad, -ZLoad, 3);

% MOMENT DISTRIBUTION
[Mz, Mx] = numInt(Vx, Vz);              % [lb*ft]
[Mz2, Mx2] = spanPolyInt(Vx, Vz, 3);
% Mz = -Mz;
% Mz2 = -Mz2;

figure
subplot(3, 1, 1)
hold on
```

```
grid on
box on
plot(y, ZLoad(1, :), '-b', 'LineWidth', 1)
plot(y, ZLoad(2, :), '-g', 'LineWidth', 1)
plot(y, ZLoad(3, :), '-r', 'LineWidth', 1)
plot(y, ZLoad(4, :), '-k', 'LineWidth', 1)
set(gca,'fontsize',16)
xlabel('Span [ft]', 'fontsize', 16)
ylabel('Z-Load [lb/ft]', 'fontsize', 16)
legend('PHAA', 'PLAA', 'NHAA', 'NLAA')
subplot(3, 1, 2)
hold on
grid on
box on
plot(y, Vz(1, :), '-b', 'LineWidth', 1)
plot(y, Vz(2, :), '-g', 'LineWidth', 1)
plot(y, Vz(3, :), '-r', 'LineWidth', 1)
plot(y, Vz(4, :), '-k', 'LineWidth', 1)
set(gca,'fontsize',16)
xlabel('Span [ft]', 'fontsize', 16)
ylabel({'Shear Force [lb]'; 'z-dir'}, 'fontsize', 16)
subplot(3, 1, 3)
hold on
grid on
box on
plot(y, Mx(1, :), '-b', 'LineWidth', 1)
plot(y, Mx(2, :), '-g', 'LineWidth', 1)
plot(y, Mx(3, :), '-r', 'LineWidth', 1)
plot(y, Mx(4, :), '-k', 'LineWidth', 1)
set(gca,'fontsize',16)
xlabel('Span [ft]', 'fontsize', 16)
ylabel({'Bending Moment [lb*ft]'; 'about x'}, 'fontsize', 16)
set(gcf,'position',[150 150 800 600])

figure
subplot(3, 1, 1)
hold on
grid on
box on
plot(y, XLoad(1, :), '-b', 'LineWidth', 1)
plot(y, XLoad(2, :), '-g', 'LineWidth', 1)
plot(y, XLoad(3, :), '-r', 'LineWidth', 1)
plot(y, XLoad(4, :), '-k', 'LineWidth', 1)
xlabel('Span [ft]', 'fontsize', 16)
ylabel('X-Load [lb/ft]', 'fontsize', 16)
legend('PHAA', 'PLAA', 'NHAA', 'NLAA')
set(gca,'fontsize',16)
subplot(3, 1, 2)
hold on
grid on
box on
plot(y, Vx(1, :), '-b', 'LineWidth', 1)
plot(y, Vx(2, :), '-g', 'LineWidth', 1)
plot(y, Vx(3, :), '-r', 'LineWidth', 1)
plot(y, Vx(4, :), '-k', 'LineWidth', 1)
xlabel('Span [ft]', 'fontsize', 16)
ylabel({'Shear Force [lb]'; 'x-dir'}, 'fontsize', 16)
```

```
set(gca,'fontsize',16)
subplot(3, 1, 3)
hold on
grid on
box on
plot(y, Mz(1, :), '-b', 'LineWidth', 1)
plot(y, Mz(2, :), '-g', 'LineWidth', 1)
plot(y, Mz(3, :), '-r', 'LineWidth', 1)
plot(y, Mz(4, :), '-k', 'LineWidth', 1)
set(gca,'fontsize',16)
xlabel('Span [ft]', 'fontsize', 16)
ylabel({'Bending Moment [lb*ft]'; 'about z'}, 'fontsize', 16)
set(gcf,'position',[150 150 800 600])

% maximum combined bending moment at root
for i = 1 : 12
    Mroot(i) = sqrt(Mz(i, end) ^ 2 + Mx(i, end) ^ 2);
end
[row, column] = find(Mroot == max(Mroot));
column1 = column;
fprintf('First guess sizing for condition # %d\n', column)
MX = abs(Mx(column, end));      % Mx in the worst case [lb*ft]
MZ = abs(Mz(column, end));      % Mz in the worst case [lb*ft]

% geometry
xs(1) = .2;   % 1st spar position [dim]
xs(2) = .5;   % 2nd spar position [dim]
Astring = .001;    % [ft^2]
tskin = .006 / 12; % skin thickness [ft]
tair1 = NACAthick(t, xs(1)) * c;  % airfoil thickness at spar 1 [ft]
tair2 = NACAthick(t, xs(2)) * c;  % airfoil thickness at spar 2 [ft]
hs(1) = tair1 - 2 * tskin;      % 1st spar height [ft]
hs(2) = tair2 - 2 * tskin;      % 2nd spar height [ft]
[row, short] = find(hs == min(hs));
[row, long] = find(hs == max(hs));
xs = xs * c;
ds = (xs(2) - xs(1)) / 2;

% 1st approximation spar thickness
tS(1) = (3 * MX + sqrt(9 * MX ^ 2 + 12 * hs(short) ^ 3 * MZ * allow)) ...
    / (2 * hs(short) ^ 2 * allow);
tS(2) = (3 * MX - sqrt(9 * MX ^ 2 + 12 * hs(short) ^ 3 * MZ * allow)) ...
    / (2 * hs(short) ^ 2 * allow);
ts = max(tS);   % 1st approximation spar thickness [ft]

UNSAFE = 1000;

Asc = .0033;          % [ft^2]
tsc = sqrt(Asc);            % [ft]

while 1
    % 1st approximation moment of inertia
    IxSpar = hs .^ 3 * ts / 12 + 4 * tsc ^ 2 * (.5 .* hs - .5 * tsc) .^ 2;
% moment of inertia single spar [ft^4]
    IzSpar = hs .* ts ^ 3 / 12 + 4 * tsc ^ 2 * (.5 * ts + .5 * tsc) ^ 2;
% moment of inertia single spar [ft^4]
```

```
    IxS = sum(IxSpar);                    % moment of inertia combined spars
[ft^4]
    IzS = sum(IzSpar) + sum(hs) * ts * ds ^ 2;  % moment of inertia combined
spars [ft^4]
    sigma = MX * max(hs) / 2 / IxS + MZ * (ds + ts / 2 + tsc) / IzS;  % [psi]

    if allow > sigma
        tS(1) = ts;
        tS(2) = .5 * ts;
        tS(3) = (tS(1) + tS(2)) / 2;
    else
        tS(1) = 2 * ts;
        tS(2) = ts;
        tS(3) = (tS(1) + tS(2)) / 2;
    end

    while abs(allow - sigma) > 100
        % moment of inertia
        IxSpar = hs .^ 3 * tS(1) / 12 + 4 * tsc ^ 2 * (.5 .* hs - .5 * tsc)
.^ 2;        % moment of inertia single spar [ft^4]
        IzSpar = hs .* tS(1) ^ 3 / 12 + 4 * tsc ^ 2 * (.5 * tS(1) + .5 * tsc)
^ 2;         % moment of inertia single spar [ft^4]
        IxS = sum(IxSpar);                    % moment of inertia combined
spars [ft^4]
        IzS = sum(IzSpar) + (sum(hs) * tS(1) + 8 * tsc ^ 2) * ds ^ 2;  %
moment of inertia combined spars [ft^4]
        SIG(1) = MX * max(hs) / 2 / IxS + MZ * (ds + tS(1) / 2 + tsc) / IzS;
% [lb/ft^2]

        % moment of inertia
        IxSpar = hs .^ 3 * tS(2) / 12 + 4 * tsc ^ 2 * (.5 .* hs - .5 * tsc)
.^ 2;         % moment of inertia single spar [ft^4]
        IzSpar = hs .* tS(2) ^ 3 / 12 + 4 * tsc ^ 2 * (.5 * tS(2) + .5 * tsc)
^ 2;         % moment of inertia single spar [ft^4]
        IxS = sum(IxSpar);                    % moment of inertia combined
spars [ft^4]
        IzS = sum(IzSpar) + (sum(hs) * tS(2) + 8 * tsc ^ 2) * ds ^ 2;  %
moment of inertia combined spars [ft^4]
        SIG(2) = MX * max(hs) / 2 / IxS + MZ * (ds + tS(2) / 2 + tsc) / IzS;
% [lb/ft^2]

        % moment of inertia
        IxSpar = hs .^ 3 * tS(3) / 12 + 4 * tsc ^ 2 * (.5 .* hs - .5 * tsc)
.^ 2;            % moment of inertia single spar [ft^4]
        IzSpar = hs .* tS(3) ^ 3 / 12 + 4 * tsc ^ 2 * (.5 * tS(3) + .5 * tsc)
^ 2;          % moment of inertia single spar [ft^4]
        IxS = sum(IxSpar);
% moment of inertia combined spars [ft^4]
        IzS = sum(IzSpar) + (sum(hs) * tS(3) + 8 * tsc ^ 2) * ds ^ 2;
% moment of inertia combined spars [ft^4]
        SIG(3) = MX * max(hs) / 2 / IxS + MZ * (ds + tS(3) / 2 + tsc) / IzS;
% [lb/ft^2]
        ts = tS(3);
        sigma = SIG(3);
        if SIG(1) > allow
            tS(2) = tS(1);
            tS(1) = 2 * tS(1);
            tS(3) = (tS(1) + tS(2)) / 2;
        elseif SIG(2) < allow
            tS(1) = tS(2);
            tS(2) = .5 * tS(2);
            tS(3) = (tS(1) + tS(2)) / 2;
        elseif SIG(1) < allow && SIG(3) > allow
            tS(2) = tS(3);
            tS(3) = (tS(1) + tS(2)) / 2;
        elseif SIG(3) < allow && SIG(2) > allow
            tS(1) = tS(3);
            tS(3) = (tS(1) + tS(2)) / 2;
        end
        % check for infinitely thin spar
        if ts < .001
            fprintf('WARNING: infinitely thin spar!\n')
            break
        end

    end

    % check FOS on all conditions
    j = 1;
    for i = 1 : 12
        sig = (abs(Mx(i, end)) * max(hs) / 2 / IxS + abs(Mz(i, end)) * (ds +
ts / 2 + tsc) / IzS); % worst point stress [lb/ft^2]
        fac = yield / sig;      % factor of safety
        if fac < (FOS - .0001)
            fprintf('Condition # %d: FOS = %.3f\n', i, fac)
            UNSAFE(j) = fac;
            UNSAFECONF(j) = i;
            j = j + 1;
        end
    end
    if FOS < min(UNSAFE)
        break
    end
    column2 = UNSAFECONF(find(UNSAFE == min(UNSAFE)));
    MX = abs(Mx(column2, end));      % Mx in the worst case [lb*ft]
    MZ = abs(Mz(column2, end));      % Mz in the worst case [lb*ft]
    fprintf('Sized for condition # %d\n', column2)
    EVAL = column2;
    UNSAFE = UNSAFE * 1000;
end

if UNSAFE ~= 1000
    column2 = UNSAFECONF(find(UNSAFE == min(UNSAFE)));
    EVAL = column2;
else
    EVAL = column1;
end

EVAL = 1;

ts;
```

```
yy = 0;
XSTRING      = zeros(1, 16);
ZSTRING      = zeros(1, 16);
STRARR       = zeros(1, 16);

for k = 1 : N
    % loop begins here

    % INPUT
    [xs, xstring, strarr, Asc, Astring, ts, tskin, Nstc, Nrib] =
inputParameters(c);
    fid     = 'MCoptimization_10kt1.mat';
    load(fid)
    ind = find(success.weight == min(success.weight));
    nstring = length(xstring);

    % xstring = [.1 .3 .4 .75 .1 .3 .4 .75] .* 5;
    % strarr = [1 1 1 1 0 0 0 0];

    % CENTROID & MOMENT OF INERTIA
    tair1 = NACAthick(t, (xs(1) / c)) * c;  % airfoil thickness at spar 1
[ft]
    tair2 = NACAthick(t, (xs(2) / c)) * c;  % airfoil thickness at spar 2
[ft]
    hs(1) = tair1 - 2 * tskin;      % 1st spar height [ft]
    hs(2) = tair2 - 2 * tskin;      % 2nd spar height [ft]
    tsc = sqrt(Asc);            % [ft]
    IxSpar = hs .^ 3 * ts / 12 + 4 * tsc ^ 2 * (.5 .* hs - .5 * tsc) .^ 2;
% moment of inertia single spar [ft^4]
    IzSpar = hs .* ts ^ 3 / 12 + 4 * tsc ^ 2 * (.5 * ts + .5 * tsc) ^ 2;

    Ix = 0;
    Iz = 0;
    Ixz = 0;
    Askin = 0;

    for i = 1 : (length(x) - 1)
        l(1, i) = sqrt((x(i) - x(i + 1)) ^ 2 + (z(1, i) - z(1, i + 1)) ^ 2);
        l(2, i) = sqrt((x(i) - x(i + 1)) ^ 2 + (z(2, i) - z(2, i + 1)) ^ 2);
        A(1, i) = l(1, i) * tskin;
        A(2, i) = l(2, i) * tskin;
        Askin = Askin + A(1, i) + A(2, i);
        xp(i) = (x(i) + x(i + 1)) / (2 * c);
        zp(1, i) = (.5 * NACAthick(t, xp(i)) + NACAcamber(m, p, xp(i))) * c;
        zp(2, i) = (-.5 * NACAthick(t, xp(i)) + NACAcamber(m, p, xp(i))) * c;
        xp(i) = xp(i) * c;
        Ix = Ix + zp(1, i) ^ 2 * A(1, i) + zp(2, i) ^ 2 * A(2, i);  % moment
of inertia [ft^4]
        Iz = Iz + xp(i) ^ 2 * (A(1, i) + A(2, i));                  % moment
of inertia [ft^4]
        Ixz = Ixz + A(1, i) * xp(i) * zp(1, i) + A(2, i) * xp(i) * zp(2, i);
    end

    % centroid calculation
    zs(1) = NACAcamber(m, p, (xs(1) / c));
    zs(2) = NACAcamber(m, p, (xs(2) / c));
    zs = zs * c;
    xstring = xstring ./ 5;
    for i = 1 : length(xstring)
        if strarr(i) == 1
            zstring(i) = (.5 * NACAthick(t, xstring(i)) + NACAcamber(m, p,
xstring(i))) * c - tskin;
        elseif strarr(i) == 0
            zstring(i) = (-.5 * NACAthick(t, xstring(i)) + NACAcamber(m, p,
xstring(i))) * c + tskin;
        end
    end
    xsp3 = .75;
    zsp3(1) = (.5 * NACAthick(t, xsp3) + NACAcamber(m, p, xsp3)) * c - tskin;
    zsp3(2) = (-.5 * NACAthick(t, xsp3) + NACAcamber(m, p, xsp3)) * c +
tskin;
    xstring = xstring .* c;
    As(1) = hs(1) * ts + 4 * Asc;
    As(2) = hs(2) * ts + 4 * Asc;
    Atot = Askin + As(1) + As(2) + nstring * Astring + 2 * Asc;
    xcskin = (sum(xp .* A(1, :)) + sum(xp .* A(2, :))) / Askin;
    zcskin = (sum(zp(1, :) .* A(1, :)) +  sum(zp(2, :) .* A(2, :))) / Askin;
    xc = (sum(xp .* A(1, :)) + sum(xp .* A(2, :)) + xs(1) * As(1) + xs(2) *
As(2)...
        + sum(xstring .* Astring) + 2 * xsp3 * Asc) / Atot;
    zc = (sum(zp(1, :) .* A(1, :)) + sum(zp(2, :) .* A(2, :)) + zs(1) * As(1)
+ zs(2) * As(2) ...
        + sum(zstring .* Astring) + sum(zsp3 .* Asc)) / Atot;

    % move the moment of inertia of the spars and skin to the centroid
    IxSp = IxSpar + As .* abs(zs - zc) .^ 2;
    IzSp = IzSpar + As .* abs(xs - xc) .^ 2;
    IxzSp = As .* abs(xs - xc) .* abs(zs - zc);
    Ix = Ix + Askin * (-zcskin ^ 2 + zc ^ 2);
    Iz = Iz + Askin * (-xcskin ^ 2 + xc ^ 2);
    Ixz = Ixz + Askin * (-xcskin * zcskin + xc * zc);
    IxTot = sum(IxSp) + Ix + sum(Astring .* (zstring - zc) .^ 2) + sum(Asc .*
(zsp3 - zc) .^ 2);
    IzTot = sum(IzSp) + Iz + sum(Astring .* (xstring - xc) .^ 2) + 2 * (Asc *
(xsp3 - xc) ^ 2);
    IxzTot = sum(IxzSp) + Ixz + sum(Astring .* (xstring - xc) .* (zstring -
zc)) + sum(Asc .* (zsp3 - zc) .* (xsp3 - xc));

    % STRESS CALCULATION

    % moments
    momX = Mx(EVAL, end);
    momZ = Mz(EVAL, end);

    % shear force shear stresses
    stringer = [xstring; strarr];

    jj = 0;
    kk = 0;
    for ii = 1 : nstring
        if stringer(2,ii) == 1
```

```
            jj = jj + 1;
            xat(jj) = stringer(1,ii);
        else
            kk = kk + 1;
            xab(kk) = stringer(1,ii);
        end
    end

    xat = sort(xat);
    xab = sort(xab);

    xsp = [xs (.75 * c)];
    At = zeros(1, length(xat)) + .0015;
    Ab = zeros(1, length(xab)) + .0015;
    Asp = [2*Asc 2*Asc Asc; 2*Asc 2*Asc Asc];
    tsp = [ts ts tskin];
    VZ = Vz(EVAL, end);
    VX = Vx(EVAL, end);
    xVz = .25 * c;
    zVx = 0;
    [xaat, xaab, Aat, Aab, Aasp] = ShearFlowInput(xat, xab, xsp, At, Ab, Asp,
tskin, ...
        tsp, Nstc, c, t, m, p, momX, momZ, IxTot, IzTot, IxzTot, xc, zc);
    tat = zeros(1,length(xaat)+3) + tskin;
    tab = zeros(1,length(xaab)+3) + tskin;

[x1,x2,x3,z1,z2,z3,tau1,tau2,tau3,junk1,junk2,dTdymax,Npa,xcshear,zcshear] =
ShearFlowCenter(xaat, xaab, xsp, Aat, Aab, Aasp, tat, tab, tsp, VZ, VX, xVz,
zVx);

    x1=x1+xcshear-xc;
    x2=x2+xcshear-xc;
    x3=x3+xcshear-xc;
    z1=z1+zcshear-zc;
    z2=z2+zcshear-zc;
    z3=z3+zcshear-zc;

    % positions to be evaluated
    xev1 = (x1(2 : length(x1)) + x1(1 : (length(x1) - 1))) / 2;
    xev2 = (x2(2 : length(x2)) + x2(1 : (length(x2) - 1))) / 2;
    xev3 = (x3(2 : length(x3)) + x3(1 : (length(x3) - 1))) / 2;
    xev3(length(x3)) = (x3(1)+x3(length(x3)))/2;
    zev1 = (z1(2 : length(z1)) + z1(1 : (length(z1) - 1))) / 2;
    zev2 = (z2(2 : length(z2)) + z2(1 : (length(z2) - 1))) / 2;
    zev3 = (z3(2 : length(z3)) + z3(1 : (length(z3) - 1))) / 2;
    zev3(length(z3)) = (z3(1)+z3(length(z3)))/2;

    % bending normal stresses
    for i = 1 : length(xev1)
        sigmaBend1(i) = bendStress(momX, momZ, IxTot, IzTot, IxzTot, xev1(i),
zev1(i));
    end
    for i = 1 : length(xev2)
        sigmaBend2(i) = bendStress(momX, momZ, IxTot, IzTot, IxzTot, xev2(i),
zev2(i));
    end
    for i = 1 : length(xev3)
        sigmaBend3(i) = bendStress(momX, momZ, IxTot, IzTot, IxzTot, xev3(i),
zev3(i));
    end

    % torsional shear stresses
    [qMy, dTdyMac, Lskin, Arib] = torqueFlow(My1, xs, hs, ts, c, tskin, t, m,
p, G);
    tau1tot = tau1(1 : (length(tau1) - 1)) + qMy(1) / tskin;
    tau2tot = tau2(1 : (length(tau2) - 1)) + qMy(2) / tskin;
    tau3tot = tau3 + qMy(3) / tskin;

    % Von Mises equivalent normal stress and factor of safety
    for i = 1 : length(xev1)
        sigmaVM1(i) = VonMises(0, sigmaBend1(i), 0, 0, 0, tau1tot(i));
        FOScheck1(i) = yield / sigmaVM1(i);
    end
    for i = 1 : length(xev2)
        sigmaVM2(i) = VonMises(0, sigmaBend2(i), 0, 0, 0, tau2tot(i));
        FOScheck2(i) = yield / sigmaVM2(i);
    end
    for i = 1 : length(xev3)
        sigmaVM3(i) = VonMises(0, sigmaBend3(i), 0, 0, 0, tau3tot(i));
        FOScheck3(i) = yield / sigmaVM3(i);
    end

    FOSmin1 = min([min(abs(FOScheck1(:))), min(abs(FOScheck1(:)))]);
    FOSmin2 = min([min(abs(FOScheck2(:))), min(abs(FOScheck2(:)))]);
    FOSmin3 = min([min(abs(FOScheck3(:))), min(abs(FOScheck3(:)))]);
    FOSmin(k) = min([FOSmin1, FOSmin2, FOSmin3]);


    %% BUCKLING CHECK

    spL = b / (2 * Nrib);

[sigColCR_t,sigColCR_b,skShrStrssCR_T,skShrStrssCR_B,xStuffT,xStuffB,dt,db]
...
        = bucklingCheck(xstring,xs,strarr,Astring,Asc,spL,tskin,c,Nstc,E);

    % 2 loops to calc axial stress at top and bottom stringers
    for uu = 2 : (length(xStuffT)-1)
        zStuffT              = c*(.5 * NACAthick(t, xStuffT(uu)/c) + ...
            NACAcamber(m, p, xStuffT(uu)/c));
        sigAxialBuckT(uu-1) = bendStress(momX, momZ, IxTot, IzTot, ...
            IxzTot, xStuffT(uu),zStuffT);
    end
    for ww = 2 : (length(xStuffB)-1)
        zStuffB              = c*(.5 * NACAthick(t, xStuffB(ww)/c) + ...
            NACAcamber(m, p, xStuffB(ww)/c));
        sigAxialBuckB(ww-1) = bendStress(momX, momZ, IxTot, IzTot, ...
            IxzTot, xStuffB(ww),zStuffB);
    end
        clear zStuffT zStuffB xStuffT xStuffB
    %
```

```
for tt = 1 : (Nstc(1,1)+1)
    initVal = (tt-1)*10 + 1;
    finVal  = tt*10;
    shrStressT(tt) = max(tau1tot(initVal:finVal));
end
shrStressT = fliplr(shrStressT);
for pp = 1:(Nstc(1,2)+1)
    initVal = (pp-1)*10 + 1;
    finVal  = pp*10;
    shrStress2T(pp) = max(tau2tot(initVal:finVal));
end
shrStress2T = fliplr(shrStress2T);
shrStressT = [shrStressT shrStress2T];
for eee = 1:(Nstc(1,3)+1)
    initVal = (eee-1)*10 + 1;
    finVal  = eee*10;
    shrStress3T(eee) = max(tau3tot(initVal:finVal));
end
shrStress3T = fliplr(shrStress3T);
shrStressT = [shrStressT shrStress3T];

for qqq = 1 : (Nstc(2,1)+1)
    initVal = (qqq-1)*10 + (Nstc(1,1)+1)*10 + 1;
    finVal  = qqq * 10 +(Nstc(1,1)+1)*10;
    shrStressB(qqq) = max(tau1tot(initVal:finVal));
end
shrStressB = fliplr(shrStressB);
for ppp = 1 : (Nstc(2,2)+1)
    initVal = (ppp-1)*10 + 2 + (Nstc(1,2)+1)*10;
    finVal  = ppp*10 + 1 + (Nstc(1,2)+1)*10;
    shrStress2B(ppp) = max(tau2tot(initVal:finVal));
end
shrStress2B = fliplr(shrStress2B);
shrStressB  = [shrStressB shrStress2B];
for ttt = 1 : (Nstc(2,3)+1)
    initVal = (ttt-1)*10 + 2 + (Nstc(1,3)+1)*10;
    finVal  = ttt*10 + 1 + (Nstc(1,3)+1)*10;
    shrStress3B(ttt) = max(tau3tot(initVal:finVal));
end
shrStress3B = fliplr(shrStress3B);
shrStressB = [shrStressB shrStress3B];
clear shrStress2T shrStress3T shrStress2B shrStress3B
% check buckling against measured stresses
for yyy = 1 : (length(sigColCR_t)-1)
    if abs(sigColCR_t(yyy)) > abs(sigAxialBuckT(yyy))
        colBuckCheckT(yyy) = 0;
    else
        colBuckCheckT(yyy) = 1;
    end
end
for hhh = 1 : (length(sigColCR_b)-1)
    if abs(sigColCR_b(hhh)) > abs(sigAxialBuckB(hhh))
        colBuckCheckB(hhh) = 0;
    else
        colBuckCheckB(hhh) = 1;
    end
end
```

```
    for jjj = 1 : (length(shrStressT))
        if  skShrStrssCR_T(jjj) > abs(shrStressT(jjj))
            skinBuckCheckT(jjj) = 0;
        else
            skinBuckCheckT(jjj) = 1;
        end
    end
    for nnn = 1 : (length(shrStressB))
        if  skShrStrssCR_B(nnn) > abs(shrStressB(nnn))
            skinBuckCheckB(nnn) = 0;
        else
            skinBuckCheckB(nnn) = 1;
        end
    end
    clear shrStressT shrStressB sigAxialBuckT sigAxialBuckB
    BuckCheck = [colBuckCheckT colBuckCheckB skinBuckCheckT skinBuckCheckB];
    % Returns 1 if buckled, 0 if not
    Buckle     = any(BuckCheck);
%     if Buckle
%         error(sprintf('Buckling Failure\n'))
%     end

    %       sigBuck = max([sigColCR_t sigColCR_b skShrStrssCR_T ...
    %       skShrStrssCR_B]);
    %% CHECK

    if FOSmin(k) >= FOS && ~Buckle

        yy = yy + 1;
        XS(yy, :) = xs;
        XSTRING(yy, 1:length(xstring)) = xstring;
        ZSTRING(yy, 1:length(xstring)) = zstring;
        STRARR(yy, 1:length(strarr)) = strarr;
        NSTRING(yy) = length(xstring);
        ASC(yy, :) = Asc;
        ASTRING(yy, :) = Astring;
        TS(yy, :) = ts;
        TSKIN(yy, :) = tskin;
        ATOT(yy) = Atot;
        XC(yy) = xc;
        ZC(yy) = zc;
        NRIB(yy) = Nrib;
%         Istore(1,yy) = IxTot;
%         Istore(2,yy) = IzTot;
%         Istore(3,yy) = IxzTot;

        % COST FUNCTION
        [weight(yy)] = funzioneDiCosto(b, c, xs, xstring, Asc, Astring, ts,
tskin, spL, Lskin, Arib);

        if weight(yy) <= min(weight)
            xaatbest=xaat;
            xaabbest=xaab;
            xspbest=xsp;
            Aatbest=Aat;
            Aabbest=Aab;
            Aaspbest=Aasp;
            tatbest=tat;
            tabbest=tab;
            tspbest=tsp;
            VZbest=VZ;
            VXbest=VX;
            Fz1best=Fz1;
            Fx1best=Fx1;
            My1best=My1;
        end
    end


    % WING TWIST

    Vzflip=fliplr(Vz(EVAL,:));
    Vxflip=fliplr(Vx(EVAL,:));
    [Thetadistr, dTdy] = 
AngleofTwist(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vzflip,Vxflip,xVz,zVx,dTdyMac)
;
    Thetadistr = Thetadistr * 180 / pi;

    figure
    hold on
    grid on
    plot(y, Thetadistr, 'LineWidth', 2)
    title('TWIST ANGLE DISTRIBUTION', 'fontsize', 18)
    xlabel('Spanwise Location (from wing root) [ft]', 'fontsize', 16)
    ylabel('Angle of Twist [deg]', 'fontsize', 16)
    set(gca,'fontsize',16)
    set(gcf,'position',[150 150 800 600])

    %WING DEFLECTION

    Imatrix = [-IxzTot IxTot; ...
        IzTot -IxzTot];
    for i = 1 : max(size(Mx))
        d2u_dy2(:, i) = -1 / (E * (IxTot * IzTot - IxzTot ^ 2)) .* Imatrix 
...
            * [Mx(EVAL, i); Mz(EVAL, i)];
    end

    d2u_dy2 = fliplr(d2u_dy2);

    [du_dy(1, :) du_dy(2, :)] = numInt(d2u_dy2(1, :), d2u_dy2(2, :));
    [u(1, :) u(2, :)] = numInt(du_dy(1, :), du_dy(2, :));
    u(1, :) = -u(1, :);

            figure
            hold on
            grid on
            plot(y, u, 'LineWidth', 2)
            title('DEFLECTION DISTRIBUTION', 'fontsize', 18)
            xlabel('Spanwise Location (from wing root) [ft]', 'fontsize', 16)
            ylabel('Deflection [ft]', 'fontsize', 16)
            legend('deflection in X-dir', 'deflection in Z-dir', 'Location',
'NorthWest')
            set(gca,'fontsize',16)
            set(gcf,'position',[150 150 800 600])




    clear zstring xstring strarr Nstc xat xab xsp At Ab Nstc

    fprintf('%d\n', k)

    %% Save Data Every 100th step to file, clear workspace
    isItEnd = (k == N);
    isIt100 = (round(k/100) == k/100);


    if (isItEnd || isIt100) && exist('XS')
%          fid = fopen(sprintf('MCoptimization%g.txt',N),'a');
%          f
        if ~exist('MCoptimization.mat')
%              numPass          = length(weight);
            success.XS      = XS;
            success.XSTRING = XSTRING;
            success.ZSTRING = ZSTRING;
            success.STRARR  = STRARR;
            success.NSTRING = NSTRING;
            success.ASC     = ASC;
            success.ASTRING = ASTRING;
            success.TS      = TS;
            success.TSKIN   = TSKIN;
            success.ATOT    = ATOT;
            success.XC      = XC;
            success.ZC      = ZC;
            success.NRIB    = NRIB;
            success.weight  = weight;
            save 'MCoptimization.mat' success
            clear XS XSTRING ZSTRING STRARR NSTRING ASC ASTRING TS TSKIN ...
                ATOT XC ZC weight success
            yy = 0;
            XSTRING     = zeros(1, 16);
            ZSTRING     = zeros(1, 16);
            STRARR      = zeros(1, 16);

        else
            load MCoptimization.mat
%              numPass          = length(weight);
            success.XS      = [success.XS; XS];
            success.XSTRING = [success.XSTRING; XSTRING];
            success.ZSTRING = [success.ZSTRING; ZSTRING];
            success.STRARR  = [success.STRARR; STRARR];
            success.NSTRING = [success.NSTRING NSTRING];
            success.ASC     = [success.ASC; ASC];
            success.ASTRING = [success.ASTRING; ASTRING];
            success.TS      = [success.TS; TS];
            success.TSKIN   = [success.TSKIN;TSKIN];
            success.ATOT    = [success.ATOT ATOT];
            success.XC      = [success.XC XC];
            success.ZC      = [success.ZC ZC];
            success.NRIB    = [success.NRIB NRIB];
            success.weight  = [success.weight weight];
            save 'MCoptimization.mat' success
            clear XS XSTRING ZSTRING STRARR NSTRING ASC ASTRING TS TSKIN ...
                ATOT XC ZC NRIB weight success
            yy = 0;
            XSTRING     = zeros(1, 16);
            ZSTRING     = zeros(1, 16);
            STRARR      = zeros(1, 16);
        end
    end
end

%% WING DIVERGENCE

[x1,x2,x3,z1,z2,z3,tau1,tau2,tau3,ex,ez,dTdymax,Npa,xcshear,zcshear]...
    = Shear(xaatbest, xaabbest, xspbest, Aatbest, Aabbest, Aaspbest,
tatbest,...
    tabbest, tspbest, VZbest, VXbest, xVz, zVx);

xac     = 0.25*c;
zac     = 0*c;
Ex      = ex-xac;
Ez      = ez-zac;
Ed      = sqrt(Ex^2+Ez^2)/c;
Tsc     = -My1best+Fz1best/2*Ex-Fx1best/2*Ez;
GJ      = abs(Tsc/dTdymax);
Vdiv    = sqrt(pi^2*GJ/(2*aw*c^2*Ed*RHO(1)*(b/2)^2));


%% FATIUGE FAILURE

n_fat   = fatigue(a_crack,KIc,sigmaBend1,sigmaBend2,sigmaBend3,C_fat,m_fat);

%% FLUTTER

fFlut   = flutter('MCoptimization');

%%
load MCoptimization
ind = find(success.weight == min(success.weight));

HS(1) = c * NACAthick(t, (success.XS(ind, 1) / c));
HS(2) = c * NACAthick(t, (success.XS(ind, 2) / c));
ZS(1) = c * NACAcamber(m, p, (success.XS(ind, 1) / c));
ZS(2) = c * NACAcamber(m, p, (success.XS(ind, 2) / c));
TSC = sqrt(success.ASC(ind));

% airfoilplot
figure
hold on
axis equal
```

```
grid on
box on
plot(x, z(1, :), 'LineWidth', 2)
plot(x, z(2, :), 'LineWidth', 2)
plot([x(end) x(end)], [z(1, end) z(2, end)], 'blue', 'LineWidth', 2)
plot([success.XS(ind, 1) success.XS(ind, 1)], [(ZS(1) - HS(1) / 2) (ZS(1) +
HS(1) / 2)], ...
    'red', 'LineWidth', 2)
plot([success.XS(ind, 2) success.XS(ind, 2)], [(ZS(2) - HS(2) / 2) (ZS(2) +
HS(2) / 2)], ...
    'red', 'LineWidth', 2)
plot([(success.XS(ind, 1) - TSC / 2) (success.XS(ind, 1) - TSC / 2)
(success.XS(ind, 1) + TSC / 2) (success.XS(ind, 1) + TSC / 2) ...
    (success.XS(ind, 2) - TSC / 2) (success.XS(ind, 2) - TSC / 2)
(success.XS(ind, 2) + TSC / 2) (success.XS(ind, 2) + TSC / 2)], ...
    [(ZS(1) - HS(1) / 2  + TSC / 2) (ZS(1) + HS(1) / 2  - TSC / 2) ...
    (ZS(1) - HS(1) / 2  + TSC / 2) (ZS(1) + HS(1) / 2  - TSC / 2) ...
    (ZS(2) - HS(2) / 2  + TSC / 2) (ZS(2) + HS(2) / 2  - TSC / 2) ...
    (ZS(2) - HS(2) / 2  + TSC / 2) (ZS(2) + HS(2) / 2  - TSC / 2)], ...
    'rs', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', ...
    'MarkerSize', 5)
plot(success.XSTRING(ind, 1 : success.NSTRING(ind)), success.ZSTRING(ind, 1 :
success.NSTRING(ind)), 'rs', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g',
...
    'MarkerSize', 4)
plot(success.XC(ind), success.ZC(ind), '+r', 'Markersize', 7)
set(gca,'fontsize',16)
set(gcf,'position',[150 150 800 600])
xlim([-.5 4.25])
ylim([-.5 .5])

toc
```

AngleofTwist.m--

```
%% Compute Angle of Twist

function [Thetadistr, dTdy] =
AngleofTwist(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vz,Vx,xVz,zVx,dTdyMac)

for ii=1:length(Vz)
    VZ=Vz(ii);
    VX=Vx(ii);

[trash4,trash5,trash6,s1,s2,s3,trash1,trash2,trash3,junk1,junk2,dTdyshear(ii)
,junk3,junk4,junk5] =
ShearFlowCenter(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,VZ,VX,xVz,zVx);
end

dTdy=dTdyshear+dTdyMac;
[Thetadistr junk1]=numInt(dTdy,dTdy);
```

Atm.m---

```
% Determines density, viscosity, speed of sound in air, temperature and
pressure
% at an altitude h above sea level.

% rho - density
% mu  - viscocity
% a0  - speed of sound
% h   - altitude
% T   - temperature
% P   - pressure
% All are in British units

function [rho, mu, a0, t, P] = atm(Alt)

h = Alt;
if (h<36089)
         rho = .002377 * ((1.- .00000686*h)^4.26);
         mu = (3.744-.217*h/10000.)*1.E-7;
         t = 59 - 0.00356 * h;
         T = t + 459.67;
         P = 2116 * ((t + 459.7) / 518.6) ^ 5.256;
else
         rho = .0007076/ (2.718^((.00004795*h)-1.7304));
         mu = 2.9603E-7;
         t = -70;
         T = t + 459.67;
         P = 473.1 * exp(1.73 - .000048 * h);
end

a0 = 49.043*sqrt(T);
```

BendStress.m---

```
%% Compute Bending Stress

function [sigmaBend] = bendStress(Mx, Mz, Ix, Iz, Ixz, x, z)

% INPUT:
% Mx & Mz    -   bending moment about x and z axis respectively
% Ix & Iz    -   moment of inertia about x and z axis respectively
% Ixz        -   symmetric moment of inertia
% x & z      -   location in the section to be evaluated
%
% OUTPUT:
% sigmaBend      -   stress due to bending

sigmaBend = - (Mz * Ix + Mx * Ixz) * x / (Ix * Iz - Ixz ^ 2) + ...
    (Mx * Iz + Mz * Ixz) * z / (Ix * Iz - Ixz ^ 2);
```

bucklingCheck.m--

```
%% Check Buckling Failure Criteria

function
[sigColCR_t,sigColCR_b,skShrStrssCR_T,skShrStrssCR_B,xatall,xaball,dt,db] =
bucklingCheck(xst,xsp,torb,Ast,Asp,spL,tsk,chord,Nstc,E)

% INPUTS:
% xst       = x locations of stringers
% xsp       = x locations of spars
% torb      = vector with corresponding top or bottom information for
%             stringers. Should be same length as xst.
% Ast       = area of stringer cross sections
% Asp       = Area of spar cap cross section
% spL       = spanwise length of skin/stringer sections. You could think of
%             this as the spacing between each rib.
% tsk       = skin thickness
% chord     = chord length
% Nstc      = The number of stringers per cell
%
% OUTPUTS:
% sigColCR_top  = the maximum compressive stresses for column buckling at
%                 each stringer along the top of the wing cross section.
% sigColCR_bot  = same as above for the bottom of the cross section
% skinShearCR_T = the maximum shear stress on the skin panels on top of the
%                 wing cross section
% skinShearCR_B = the maximum shear stress on the skin panels on the bottom
%                 of the wing cross section
%
% Input the stringer area, plate dimensions (spanwise and chordwise
% directions respectively), and the plate thickness (skin thickness). The
% output is the critical stress for the given geometry. Assumes loading is
% compressive. Also assumes loading in the spanwise direction, if loaded in
% longitudonal direction switch the order of the plate dimetions.
%
% [critStress] = bucklingCheck(As,a,b,t)

%% Load Material Properties

% Aluminum 7475-T61
% E       = E;
nu      = 0.33;

%% Airfoil Properties
c           = chord;
xsp         = [xsp c*.75];
m           = .02;
p           = .4;
t           = .12;
numstr      = length(xst);

%% Compute z-locations of all Stringers and spar caps

jj = 0;
kk = 0;
```

```
for ii = 1 : numstr
    if torb(ii) == 1
        jj = jj + 1;
        xat(jj) = xst(ii);
    else
        kk = kk + 1;
        xab(kk) = xst(ii);
    end
end

xat = sort(xat);
xab = sort(xab);

for ii=1:length(xat)
    zat(ii)= c*(.5 * NACAthick(t, xat(ii)/c) + NACAcamber(m, p, xat(ii)/c));
end

for ii=1:length(xab)
    zab(ii) = c*(-.5 * NACAthick(t, xab(ii)/c) + NACAcamber(m, p,
xab(ii)/c));
end

for ii=1:length(xsp)
    zsp(1,ii)= c*(.5 * NACAthick(t, xsp(ii)/c) + NACAcamber(m, p,
xsp(ii)/c));
    zsp(2,ii)= c*(-.5* NACAthick(t, xsp(ii)/c) + NACAcamber(m, p,
xsp(ii)/c));
end

%% Cell Stringer Calcs

N1t     = Nstc(1,1);
N1b     = Nstc(2,1);

N2t     = Nstc(1,2);
N2b     = Nstc(2,2);

N3t     = Nstc(1,3);
N3b     = Nstc(2,3);

%% Swept Areas

iinc=0.001;
xatall=[0 xat(1:N1t) xsp(1) xat(N1t+1:N1t+N2t) xsp(2)
xat(N1t+N2t+1:N1t+N2t+N3t) xsp(3)];
xaball=[0 xab(1:N1b) xsp(1) xab(N1b+1:N1b+N2b) xsp(2)
xab(N1b+N2b+1:N1b+N2b+N3b) xsp(3)];

xatall = sort(xatall);
xaball = sort(xaball);

for ii=1 : (length(xatall)-1)
    x= xatall(ii) : iinc : xatall(ii+1);
    d=0;
    z=c*(.5 * NACAthick(t, x(1)/c) + NACAcamber(m, p, x(1)/c));
    for jj=2 : (length(x))
        z(jj)=c*(.5 * NACAthick(t, x(jj)/c) + NACAcamber(m, p, x(jj)/c));
        dtemp=sqrt( (x(jj)-x(jj-1))^2 + (z(jj)-z(jj-1))^2);
        d=d+dtemp;
    end
    dt(ii)=d;
end

for ii=1 : (length(xaball)-1)
    x= xaball(ii) : iinc : xaball(ii+1);
    d=0;

    z=c*(-.5 * NACAthick(t, x(1)/c) + NACAcamber(m, p, x(1)/c));
    for jj= 2 : (length(x))
        z(jj)=c*(-.5 * NACAthick(t, x(jj)/c) + NACAcamber(m, p, x(jj)/c));
        dtemp=sqrt( (x(jj)-x(jj-1))^2 + (z(jj)-z(jj-1))^2);
        d=d+dtemp;
    end
    db(ii)=d;
end

dsp=zsp(1,:)-zsp(2,:);

for ii=1:length(xsp)
    atemp=sqrt( (xsp(ii)-xsp(1))^2 + (zsp(1,ii)-zsp(1,1))^2);
    btemp=sqrt( (xsp(ii)-xsp(1))^2 + (zsp(2,ii)-zsp(1,1))^2);
    dtemp=dsp(ii);
end

%% Organize Arc Lengths and Sweep Areas for Shear Flow Calcs

for ii=1:N1t+1
    d1t(ii)=dt(N1t+2-ii);
end

for ii=1:N1b+1
    d1b(ii)=db(ii);
end

for ii=1:N2t+1
    d2t(ii)=dt(N1t+N2t+3-ii);
end

for ii=1:N2b+1
    d2b(ii)=db(N1b+1+ii);
end

for ii=1:N3t+1
    d3t(ii)=dt(N1t+N2t+N3t+4-ii);
end

for ii=1:N3b+1
```

```
    d3b(ii)=db(N1b+N2b+2+ii);
end

n1t=length(d1t);
n1b=length(d1b);
n2t=length(d2t);
n2b=length(d2b);
n3t=length(d3t);
n3b=length(d3b);

d1=[d1t(1:n1t-1) , d1t(n1t)+d1b(1) , d1b(2:n1b) , dsp(1)];

d2=[d2t dsp(1) d2b dsp(2)];

d3=[d3t dsp(2) d3b dsp(3)];

%% Stringer Cross Section & Centroid Calc
% Below is the stringer cross section with local skin sections for
% calculating the centroid, y-axis down the centerline of the stringer
% +++L1+++ +++++L2++++
%   -L3-|    |----
%       |    |
%      L4    |
%       |_L5_|

L3      = 9.5  * 0.0032808399;                  % [ft]
L4      = 31.8 * 0.0032808399;                  % [ft]
L5      = 19   * 0.0032808399;                  % [ft]
tst     = Ast / (2 * L3 + 2 * L4 + L5);         % [ft]

for ii = 2:length(xatall)

    if ii == 2
        Lt(ii-1,1)    = dt(ii-1);
        Lt(ii-1,2)    = dt(ii)/2;

    elseif ii == length(xatall)
        Lt(ii-1,1)    = dt(ii-1);
        Lt(ii-1,2)    = 0;

    else
        Lt(ii-1,1)    = dt(ii-1)/2;
        Lt(ii-1,2)    = dt(ii)/2;
    end

    At(ii-1)   = Ast + tsk*(Lt(ii-1,1)+Lt(ii-1,2));

    %Ctx is the centroid position of the x-coordinate as measured from the
    %leftemost point of L3
    Ctx(ii-1)  = ((L3/2*L3 + L3*L4 + (L3 + L5/2)*L5 + (L3 + L5)*L4 + ...
            (L3 + L5 + L3/2)*L3)*tst + (L3 - Lt(ii-1,1)/2)*Lt(ii-1,1)*tsk +
...
            (L3 + L5 + Lt(ii-1,2)/2)*Lt(ii-1,2)*tsk)/((2*L3 + 2*L4 + L5)*tst
+ ...
```

```
            (Lt(ii-1,1)+Lt(ii-1,2))*tsk);

    %Cty is the centroid position of the y-coordinate as measured from the
    %skin (which, we assume here, is coincident with upper flanges of the
    %stringer)
    Cty(ii-1) = (2*L4/2*L4*tst + L4*L5*tst)/(2*L4*tst+L5*tst);

    Ixt(ii-1) = (1/12*Lt(ii-1,1)*tsk^3 + Lt(ii-1,1)*tsk*Cty(ii-1)^2) + ...
                2*(1/12*L3*tst^3 + Cty(ii-1)^2*tst*L3) + ...
                2*(1/12*tst*L4^2 + tst*L4*(L4/2)^2) + ...
                (1/12*L5*tst^3 + (L5-Cty(ii-1))^2*tst*L5) + ...
                (1/12*Lt(ii-1,2)*tsk^3 + Lt(ii-1,2)*tsk*Cty(ii-1)^2);

end

for ii = 2:length(xaball)

    if ii == 2
        Lb(ii-1,1)     = db(ii-1);
        Lb(ii-1,2)     = db(ii)/2;

    elseif ii == length(xaball)
        Lb(ii-1,1)     = db(ii-1);
        Lb(ii-1,2)     = 0;

    else
        Lb(ii-1,1)     = db(ii-1)/2;
        Lb(ii-1,2)     = db(ii)/2;
    end

    Ab(ii-1)   = Ast + tsk*(Lb(ii-1,1)+Lb(ii-1,2));

    Cbx(ii-1) = ((L3/2*L3 + L3*L4 + (L3 + L5/2)*L5 + (L3 + L5)*L4 + ...
            (L3 + L5 + L3/2)*L3)*tst + (L3 - Lb(ii-1,1)/2)*Lb(ii-1,1)*tsk +
...
            (L3 + L5 + Lb(ii-1,2)/2)*Lb(ii-1,2)*tsk)/((2*L3 + 2*L4 + L5)*tst
+ ...
            (Lb(ii-1,1)+Lb(ii-1,2))*tsk);

    Cby(ii-1) = (2*L4/2*L4*tst + L4*L5*tst)/(2*L4*tst+L5*tst);

    Ixb(ii-1) = (1/12*Lb(ii-1,1)*tsk^3 + Lb(ii-1,1)*tsk*Cby(ii-1)^2) + ...
                2*(1/12*L3*tst^3 + Cby(ii-1)^2*tst*L3) + ...
                2*(1/12*tst*L4^2 + tst*L4*(L4/2)^2) + ...
                (1/12*L5*tst^3 + (L5-Cby(ii-1))^2*tst*L5) + ...
                (1/12*Lb(ii-1,2)*tsk^3 + Lb(ii-1,2)*tsk*Cby(ii-1)^2);

end

%% Column Buckling of stringer/skin combo
Lrib        = spL;
Pcr_t       = (pi^2 * E .* Ixt) ./ Lrib^2;
Pcr_b       = (pi^2 * E .* Ixb) ./ Lrib^2;
```

```
for jj = 1:length(Pcr_t)
    sigColCR_t(jj)     = Pcr_t(jj) / At(jj);
end

for kk = 1:length(Pcr_b)
    sigColCR_b(kk)     = Pcr_b(kk) / Ab(kk);
end


%% Plate Buckling

% assume clamped plate on all 4 sides. From figue 9.3(a) in Megson

%% Plate Buckling of Stringer Panels

for jj = 1:3

    a          = spL;

    if jj == 1
        bst      = L3;
    elseif jj == 2
        bst    = L4;
    else
        bst    = L5;
    end

    % for compressive loading, pinned on all 4 sides from figure 9.3(a) in
    % Megson:

    if a/bst < 0.5
        k    = 14;
    elseif a/bst < 0.75
        k    = 11;
    elseif a/bst < 1
        k    = 10.5;
    elseif a/bst < 1.5
        k    = 9;
    elseif a/bst < 2
        k    = 8;
    else
        k    = 7.75;
    end

    D            = E * tst^3 / (12*(1-nu^2));
    st_N_CR(jj)  = k * pi^2 * D  / bst^2;
%     st_N_CR(jj) = k * pi^2 * E / (12 * (1-nu^2)) * (tst/bst)^2;
end

avg_pl_CR    = (sum(st_N_CR)+sum(st_N_CR(1:2))) / (tst*(2*L3 + 2*L4 + L5));

check_stT    = find(avg_pl_CR < sigColCR_t);
check_stB    = find(avg_pl_CR < sigColCR_b);
```

```
if ~isempty(check_stT)
    for ii = check_stT
        sigColCR_t(ii)   = avg_pl_CR;
    end
end

if ~isempty(check_stB)
    for ii = check_stB
        sigColCR_b(ii)   = avg_pl_CR;
    end
end


%% Find Buckling Shear Stress on Skin

for ii = 1 : length(dt)

    a = spL;
    b = dt(ii);

    % buckling coefficients for a flat plate under shear from Megson figure
    % 9.3(c)
    if a/b < 1
        k = 15;
    elseif a/b < 1.25
        k = 13.5;
    elseif a/b < 1.5
        k = 12;
    elseif a/b < 1.75
        k = 11;
    elseif a/b < 2
        k = 10.5;
    elseif a/b < 2.5
        k = 10;
    else
        k = 9.5;
    end

    D                = E * tsk^3 / (12*(1-nu^2));
    NskShrCR_T(ii)     = k * pi^2 * D  / b^2;
    skShrStrssCR_T(ii) = NskShrCR_T(ii) / (tsk*b);

end

for ii = 1 : length(db)

    a = spL;
    b = db(ii);

    % buckling coefficients for a flat plate under shear from Megson figure
    % 9.3(c)
    if a/b < 1
        k = 15;
    elseif a/b < 1.25
        k = 13.5;
    elseif a/b < 1.5
        k = 12;
    elseif a/b < 1.75
        k = 11;
    elseif a/b < 2
        k = 10.5;
    elseif a/b < 2.5
        k = 10;
    else
        k = 9.5;
    end

    D                = E * tsk^3 / (12*(1-nu^2));
    NskShrCR_B(ii)     = k * pi^2 * D  / b^2;
    skShrStrssCR_B(ii) = NskShrCR_B(ii) / (tsk*b);

end
```

cellArea.m---

```
%% Compute Cell Areas

function [Acell1, Acell2, Acell3] = cellArea(xsp, c, t, m, p)

hsp=xsp/5;
iinc=0.001;

x=0 : iinc : hsp(1);
for i = 1 : length(x)
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;

Acell1=iinc*5*(abs(sum(z(1,:)))+abs(sum(z(2,:))));

z=0;
x=hsp(1):iinc:hsp(2);
for i = 1 : length(x)
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;

Acell2=iinc*5*(abs(sum(z(1,:)))+abs(sum(z(2,:))));

z=0;
x = [hsp(2) : iinc : .75];
for i = 1 : length(x)
```

```
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;

Acell3=iinc*5*(abs(sum(z(1,:)))+abs(sum(z(2,:))));
```

Fatigue.m--

```
%% Compute fatigue Cycles to failure

function n = fatigue(a,KIc,sigmaBend1,sigmaBend2,sigmaBend3,C,m)

% INPUT
%   a           = crack length [ft]
%   KIc         = fracture toughness [psf]
%   sigmaBend1  = bending stresses in cell 1
%   sigmaBend2  = bending stresses in cell 2
%   sigmaBend3  = bending stresses in cell 3
%   C           = Pre-exponential fatigue crack growth factor
%                   3.15e-11 for 2024T3
%   m           = Exponential fatigue crack growth factor
%                   3.59 for 2024T3
%
% OUTPUT
%   n           = number of fatigue cycles to failure

Tensile1    = sigmaBend1(find(sigmaBend1 > 0))*0.00689475729/144; %[MPa]
Tensile2    = sigmaBend2(find(sigmaBend2 > 0))*0.00689475729/144; %[MPa]
Tensile3    = sigmaBend3(find(sigmaBend3 > 0))*0.00689475729/144; %[Mpa]
maxTensile  = max([Tensile1 Tensile2 Tensile3]);
sigInf      = maxTensile;

KI          = 1.2 * sigInf * sqrt(pi * a);

if KI > KIc
    warning('Apparent structural failure due to unconstrained crack
propogation');
end

aCR         = 1 / pi * sqrt(KIc / (1.2 * sigInf));

l           = 1-m/2;

n           = (aCR^l-a^l)/(l*C*(sigInf^2*pi)^(m/2));
```

flutter.m--

```
%% Compute Flutter Natural Frequency

function f = flutter('filename')
```

```
%% Flutter Stuffs

load('filename')

F_half      = 4.4*W/2; %Force on half-span

k_spring    = F_point/max(max(u));

m           = min(success.weight)/32.2;

f_flutter   = sqrt(k_spring/m)/2/pi;
```

funzioneDiCosto.m--

```
%% Compute Weight of Wing

function [weight] =
funzioneDiCosto(b,c,xsp,xst,Asc,Ast,tsp,tsk,Lrib,Lskin,Arib)

% INPUT:
% b         = Wing span in feet
% c         = Chord length [ft]
% xsp       = x-spar locations
% xst       = x-stringer locations
% Asc       = Area of spar cap cross section [sq ft]
% Ast       = Area of stringer cross section [sq ft]
% tsp       = Thickness of spar [ft]
% tsk       = Thickness of skin [ft]
% Lrib      = Rib spacing in feet [ft]
% Lskin     = Perimeter length of skin [ft]
% Arib      = Total cross sectional area of airfoil [sq ft]
%
% OUTPUT:
% weight    = Approximate wing weight in pounds
%
% [weight] = funzioneDiCosto(b,xsp,xst,Asc,Ast,tsp,tsk,Lrib)

%% Material Properties

rhoAl   = 0.102 * 12^3;

%% Calculate Stringer & Spar Cap Volume

b       = b/2;

NStr    = length(xst);
NSprCp  = 4*length(xsp) + 2;

strVol      = NStr * Ast * b;
sprCpVol    = NSprCp * Asc * b;

%% Calculate Spar Volume
```

```
m       = .02;
p       = .4;
t       = .12;
x       = xsp ./ c;
spVol   = 0;

for i = 1 : length(x)
    hsp     = NACAthick(t, x(i));
    spVol   = spVol + hsp * c * tsp * b;
end

%% Calculate Skin Volume

skVol   = Lskin * tsk * b;

%% Calculate Rib Volume

trb     = 0.016 / 24; % its 0.016 in div 12 -> ft and div 2 -> holes
Nrib    = floor(b / Lrib);

ribVol  = Nrib * trb * Arib;

%% Calculate Weight

weight = rhoAl * (strVol + spVol + sprCpVol + skVol + ribVol);
```

inputParameters.m---

```
%% Generate Random Wing Geometry

function [xsp, xst,torb, Asc, Ast, tsp, ts1, Nstc, Nrib]=inputParameters(c)

%INPUTS:
% c   = chord length [ft]
%
% OUTPUTS:
% xsp = x spar locations
% xst = x stringer locations
% torb= stringers in xst assigned to top or bottom of cross section
% Asc = Area spar cap
% Ast = Area stringers
% tsp = spar thickness
% ts# = Skin thickness ts1 ts2 for upper and lower surfaces (more or less)
%       respectively

xspm    = [.1 .4]; % ensure at least 0.05% chord lies within ea. cell
xspM    = [.3 .65]; % do not exceed ?? spar 2 maximum

AscmM   = [250 350] .* 1.07639104e-5;   % [sq ft]
AstmM   = [50 150] .* 1.07639104e-5;    % [sq ft]
```

```
tspmM   = [.04 .08] ./ 12;                % [ft]
ts1mM   = [.03 .06] ./ 12;                % [ft]
% ts2mM   = [.002 .006] ./ 12;              % [ft]

NribmM  = [10 15];

xsp(1)  = rand * (xspM(1) - xspm(1)) + xspm(1);
xsp(2)  = rand * (xspM(2) - xspm(2)) + xspm(2);

Asc     = rand * (AscmM(2) - AscmM(1)) + AscmM(1);
Ast     = rand * (AstmM(2) - AstmM(1)) + AstmM(1);

tsp     = rand * (tspmM(2) - tspmM(1)) + tspmM(1);
ts1     = rand * (ts1mM(2) - ts1mM(1)) + ts1mM(1);
% ts2     = rand * (ts2mM(2) - ts2mM(1)) + ts2mM(1);

Nrib    = rand * (NribmM(2) - NribmM(1)) + NribmM(1);

[temp Nstc]    = placeStringers(c, xsp(1), xsp(2));

numstr      = length(temp(1, :));

jj = 0;
kk = 0;
for ii = 1 : numstr
    if temp(2,ii) == 1
        jj = jj + 1;
        xat(jj) = temp(1,ii);
    else
        kk = kk + 1;
        xab(kk) = temp(1,ii);
    end
end

xat = sort(xat);
xab = sort(xab);

xst     = [xat xab];
torb    = temp(2,:);

xsp = xsp .* c;
```

NACAcamber.m---

```
% Computes the camber ordinate for a NACA 4-digit airfoil

function [y] = NACAcamber(m, p, x)

if x <= p
    y = m * x / p ^ 2 * (2 * p - x);
else
    y = m * (1 - x) / (1 - p) ^ 2 * (1 + x - 2 * p);
```

```
end
```

NACAthick.m--

```
% Computes the thickness of a NACA 4-digit airfoil

function [t] = NACAthick(tmax, x)

t = 2 * tmax / .2 * (.2969 * sqrt(x) - .126 * x - .3516 * x ^ 2 ...
    + .2843 * x ^ 3 - .1015 * x ^ 4);
```

numInt.m--

```
% Computes the integration of a given x and z loading for a wing

function [x,z] = numInt(Xload,Zload)

span         = [0:0.1:15];

[a,b]        = size(Xload);

x            = zeros(a, b);
z            = zeros(a, b);

for ii = 1:a

    intTotX = 0;
    intTotZ = 0;

    for jj = 2:b

    intX         = Xload(ii,jj-1:jj);
    intZ         = Zload(ii,jj-1:jj);
    intSpan      = span(jj-1:jj);

    intStepX     = trapz(intSpan,intX);
    intStepZ     = trapz(intSpan,intZ);

    intTotX      = intTotX + intStepX;
    intTotZ      = intTotZ + intStepZ;

    x(ii,jj)     = intTotX;
    z(ii,jj)     = intTotZ;

    end

end
```

placeStringers.m--

```
% Generates random stringers locations

function [xst, Nstc] = placeStringers(c, s1, s2)

MinDist=0.02*c; %Maximum Distance between any adjacent stringers and spar
caps

%% How Many Stringers per Cell

for kk = 1:2                                    % cycle top & bottom surfaces
    Nc1 = [1 2 3];
    Nc2 = [1 2 3];
    Nc3 = [0 1 2];

    r = ceil(rand([1,3]) * 3);

    Nc1r = Nc1(r(1));
    Nc2r = Nc2(r(2));
    Nc3r = Nc3(r(3));

    Nstc(kk,:) = [Nc1r Nc2r Nc3r];

%%

    xmin=[0 s1 s2]*c;
    xmax=[s1 s2 0.75]*c;
    xcheck=0.01;

    while min(xcheck) <= MinDist;
        for ii=1:3
            if ii==1
                xst1=rand([1,Nstc(kk,ii)])*(xmax(ii)-xmin(ii))+xmin(ii);
            elseif ii==2
                xst2=rand([1,Nstc(kk,ii)])*(xmax(ii)-xmin(ii))+xmin(ii);
            else
                xst3=rand([1,Nstc(kk,ii)])*(xmax(ii)-xmin(ii))+xmin(ii);
            end
        end

        xall=[xst1 s1*c xst2 s2*c xst3 0.75*c];
        xall=sort(xall);
        xcheck=xall(2:length(xall)) - xall(1:(length(xall)-1));
    end

    if kk==1
        xsttop=[xst1 xst2 xst3];
    else
        xstbot=[xst1 xst2 xst3];
    end
end

sttop=ones([1,length(xsttop)]);
stbot=zeros([1,length(xstbot)]);
```

```
xst=[sort(xsttop) sort(xstbot) ; sttop stbot];
```

postProcess.m---

```
% Performs post-processing and plots

%% Housekeeping
clc;
clear;
close all;
set(0,'DefaultFigureWindowStyle','docked')

%% User Defined
n       = 1;                %number of trials

for ii = 1:n
     fid     = sprintf('MCoptimization_10kt%i.mat',ii);
    load(fid);

    if ii == 1
    xs           = success.XS;
    xString      = success.XSTRING;
    zString      = success.ZSTRING;
    strArr       = success.STRARR;
    nString      = success.NSTRING;
    Asc          = success.ASC;
    Astring      = success.ASTRING;
    tspar        = success.TS;
    tskin        = success.TSKIN;
    Atot         = success.ATOT;
    xc           = success.XC;
    zc           = success.ZC;
    Nrib         = success.NRIB;
    weight       = success.weight;
    elseif ii == 2;
    xs           = [xs;             success.XS];
    xString      = [xString;   success.XSTRING];
    zString      = [zString;   success.ZSTRING];
    strArr       = [strArr;     success.STRARR];
    nString      = [nString'; success.NSTRING'];
    Asc          = [Asc;           success.ASC];
    Astring      = [Astring;   success.ASTRING];
    tspar        = [tspar;          success.TS];
    tskin        = [tskin;       success.TSKIN];
    Atot         = [Atot';       success.ATOT'];
    xc           = [xc';           success.XC'];
    zc           = [zc';           success.ZC'];
    Nrib         = [Nrib';       success.NRIB'];
    weight       = [weight';   success.weight'];
    else
    xs           = [xs;             success.XS];
    xString      = [xString;   success.XSTRING];
    zString      = [zString;   success.ZSTRING];
    strArr       = [strArr;     success.STRARR];
    nString     = [nString;  success.NSTRING'];
    Asc         = [Asc;           success.ASC];
    Astring     = [Astring;   success.ASTRING];
    tspar       = [tspar;          success.TS];
    tskin       = [tskin;       success.TSKIN];
    Atot        = [Atot;        success.ATOT'];
    xc          = [xc;            success.XC'];
    zc          = [zc;            success.ZC'];
    Nrib        = [Nrib;        success.NRIB'];
    weight      = [weight;    success.weight'];
    end

end

%% Plot 1: Number of Weight Occurancs

figure(1);
set(gcf,'OuterPosition',[100 100 900 700],'name','Weight Histogram');
hist(weight,20)
xlabel('Weight of Successful Half-Span [lbs]','Fontsize',16)
ylabel('Number of Occurances','Fontsize',16)
set(gca,'fontsize',16)

%% Plot 2: Weight vs. Number Stringers

[sortW, I] = sort(weight);
for ii = 1:length(I)
    nStSort(ii) = nString(I(ii));
end

figure(2);
set(gcf,'name','Num Stringers Scatter')
scatter(sort(weight),nStSort)
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Number of Stringers','Fontsize',16)
r.nStrVsW   = corr2(weight, nString);

%% Plot 3: Weight vs. Spar Cap Area

figure(3);
set(gcf,'name','Spar Cap Area Scatter')
scatter(weight,Asc*144)
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Spar Cap Cross-Sectional Area [in^2]','Fontsize',16)
% r.AscVsW = corr2(weight, Asc);

%% Plot 4: Weight vs. Spar Spacing

deltaXS = xs(:,2)-xs(:,1);

figure(4)
set(gcf,'name','Spar Spacing Scatter')
scatter(weight,deltaXS)
```

```
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Distance Between Spars [ft]','fontsize',16)
% r.deltaSprVsW = corr2(weight,deltaXS);

%% Plot 5: Front Spar Position vs. Weight

figure(5)
set(gcf,'name','Front Spar Position vs. Weight')
scatter(weight,xs(:,1))
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Front Spar Distance from Leading Edge [ft]','fontsize',16)
% r.fntSparVsW = corr2(weight,xs(:,1));

%% Plot 6: Rear Spar Position vs. Weight

figure(6)
set(gcf,'name','Rear Spar Position vs. Weight')
scatter(weight,xs(:,2))
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Rear Spar Distance from Leading Edge [ft]','fontsize',16)
% r.rearSprVsW = corr2(weight,xs(:,2));

%% Plot 7: Weight vs. Stringer Area

figure(7)
set(gcf,'name','Stringer Cross-Section Area vs. Weight')
scatter(weight,Astring)
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Stringer Area [ft^2]','Fontsize',16)
% r.AstringVsW = corr2(weight,Astring);

%% Plot 8: Skin Thickness vs. Weight

figure(8)
set(gcf,'name','Skin Thickness vs. Weight')
scatter(weight,tskin)
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Skin Thickness [ft]','Fontsize',16)
% r.tSkinVsW  = corr2(weight,tskin);

%% Plot 9: Spar Thickness vs. Weight

figure(9)
set(gcf,'name','Spar Thickness vs. Weight')
scatter(weight,tspar)
set(gca,'fontsize',16)
xlabel('Weight of Half-Span [lb]','Fontsize',16)
ylabel('Spar Thickness [ft]','Fontsize',16)
% r.tSparVsW = corr2(weight,tspar);
```

postProcessVariation.m--

```
% Performs post-processing and plots

%% Housekeeping
clc;
clear all;
close all;
format compact;

%% Begin Loop

for ii = 4:4

%     save ii
%     clear
%     load ii

%% Load Data

    if ii == 1
        variation = 'Nrib';

    elseif ii == 2
        variation = 'XS';

    elseif ii == 3
        variation = 'Xstring';

    elseif ii == 4
        variation = 'Asc';

    elseif ii == 5
        variation = 'Astring';

    elseif ii == 6
        variation = 'tspar';

    elseif ii == 7
        variation = 'tskin';

    end

    cd(sprintf('%s/PostProcess/%s',pwd,variation));
    load('MCoptimization.mat');

%% Establish Variables

PorF                = success.PorF;
Ix                  = success.IX;
Iz                  = success.IZ;
Ixz                 = success.IXZ;
sigmaMax            = success.SIGMAX;
tauMax              = success.TAUMAX;
```

```
vmMax           = success.VMMAX;

indP            = find(PorF == 1);
indF            = find(PorF == 0);

    if ii == 1
        xdata    = success.NRIB;
        xdataP   = success.NRIB(indP);
        xdataF   = success.NRIB(indF);
        xstring = 'Number of Ribs';

    elseif ii == 2
        xdata    = success.XS(:,1)';
        xdataP   = success.XS(indP);
        xdataF   = success.XS(indF);
        xstring = 'Front Spar Location [ft]';

    elseif ii == 3
        xdata    = success.NSTRING;
        xdataP   = success.NSTRING(indP);
        xdataF   = success.NSTRING(indF);
        xstring = 'Number of Stringers';

    elseif ii == 4
        xdata    = success.ASC';
        xdataP   = success.ASC(indP);
        xdataF   = success.ASC(indF);
        xstring = 'Spar Cap Area [ft^2]';

    elseif ii == 5
        xdata    = success.ASTRING';
        xdataP   = success.ASTRING(indP);
        xdataF   = success.ASTRING(indF);
        xstring = 'Stringer Area [ft^2]';

    elseif ii == 6
        xdata    = success.TS';
        xdataP   = success.TS(indP);
        xdataF   = success.TS(indF);
        xstring = 'Spar Thickness [ft]';

    elseif ii == 7
        xdata    = success.TSKIN';
        xdataP   = success.TSKIN(indP);
        xdataF   = success.TSKIN(indF);
        xstring = 'Skin Thickness [ft]';
    end

%% Plot 1: Ix vs. Parameter
figure;
set(gcf,'Name','Ix Scatter','Position',[75 75 800 600])

scatter(xdataP,Ix(indP))
hold on;
scatter(xdataF,Ix(indF))
```

```
set(gca,'fontsize',16)
xlabel(sprintf('%s',xstring),'Fontsize',16)
ylabel('I_x [ft^4]','Fontsize',16)
legend('Pass','Fail')

r.Ix        = corr2(xdata,Ix);
% eval(sprintf('print ''IxVs%s.jpg'' -djpeg',variation))
% close all;

%% Plot 2: Iz vs. Parameter

figure;
set(gcf,'Name','Iz Scatter','Position',[75 75 800 600])

scatter(xdataP,Iz(indP))
hold on;
scatter(xdataF,Iz(indF))

set(gca,'fontsize',16)
xlabel(sprintf('%s',xstring),'Fontsize',16)
ylabel('I_z [ft^4]','Fontsize',16)
legend('Pass','Fail')

r.Iz        = corr2(xdata,Iz);
% eval(sprintf('print ''IzVs%s.jpg'' -djpeg',variation))
% close all;

%% Plot 3: Ixz vs. Parameter

figure;
set(gcf,'Name','Ixz Scatter','Position',[75 75 800 600])

scatter(xdataP,Ixz(indP))
hold on;
scatter(xdataF,Ixz(indF))

set(gca,'fontsize',16)
xlabel(sprintf('%s',xstring),'Fontsize',16)
ylabel('I_x_z [ft^4]','Fontsize',16)
legend('Pass','Fail')

r.Ixz       = corr2(xdata,Ixz);
% eval(sprintf('print ''IxzVs%s.jpg'' -djpeg',variation))
% close all;

%% Plot 4: Bending Stresses vs. Parameter

figure;
set(gcf,'Name','Bending Stress','Position',[75 75 800 600])

scatter(xdataP,sigmaMax(indP));
hold on;
scatter(xdataF,sigmaMax(indF));
```

```
set(gca,'fontsize',16)
xlabel(sprintf('%s',xstring),'Fontsize',16)
ylabel('Bending Stress [psf]','fontsize',16);
legend('Pass','Fail')

r.sigma       = corr2(xdata,sigmaMax);
% eval(sprintf('print ''SigmaVs%s.jpg'' -djpeg',variation))
% close all;

%% Plot 5: Shear Stresses vs. Parameter

figure;
set(gcf,'Name','Shear Stress','Position',[75 75 800 600])

scatter(xdataP, tauMax(indP));
hold on;
scatter(xdataF, tauMax(indF));

set(gca,'fontsize',16)
xlabel(sprintf('%s',xstring),'Fontsize',16)
ylabel('Shear Stress [psf]','fontsize',16)
legend('Pass','Fail')

r.tau       = corr2(xdata,tauMax);
% eval(sprintf('print ''TauVs%s.jpg'' -djpeg',variation))
% close all;

%% Plot 6: Von Mises Stress vs. Parameter

figure;
set(gcf,'Name','Von Mises Stress','Position',[75 75 800 600])

scatter(xdataP,vmMax(indP))
hold on;
scatter(xdataF,vmMax(indF))

set(gca,'fontsize',16)
xlabel(sprintf('%s',xstring),'Fontsize',16)
ylabel('Von Mises Stress [psf]','fontsize',16)
legend('Pass','Fail')

r.vm       = corr2(xdata,vmMax);
% eval(sprintf('print ''VonMisesVs%s.jpg'' -djpeg',variation))
% close all;

save 'Correlation.mat' r
cd('../..');
clear r success
end
```

shear.m--

```
%% Shear Flow and Shear Center Calculator

function [x1,x2,x3,z1,z2,z3,tau1,tau2,tau3,ex,ez,dTdy,Npa,xc,zc] =
Shear(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vz,Vx,xVz,zVx)

% Compute Shear Flows and Shear Center

% xat:  x-locations of all stringers and skin subdivisions (NOT INCLUDING
%           SPAR CAPS) along top of wing, in order from LE to TE [ft]
% xab:  Same as xat except along bottom of wing
% xsp:  x-locations of spars, including final skin section [ft]
% At:   Cross-sectional area of all stringers and skin subdivisions (NOT
%           INCLUDING SPAR CAPS) top of wing, in order from LE to TE [ft^2]
% Ab:   Same as At except along bottom of wing
% Asp:  Cross-sectional area of spar caps
% tat:  Thickness of skin sections between adjacent stringers or skin
%           subdivisions (INCLUDING SPAR CAPS), in order from LE to TE [ft]
% tab:  Same as tat, except along bottom of wing
% tsp:  Thickness of spar caps
% Vz:   Shear force in positive z-direction
% Vx:   Shear force in positive x-direction
% xVz:  x-location where Vz acts
% zVx:  z-location where Vx acts

[x1,x2,x3,z1,z2,z3,tau1,tau2,tau3,junk1,junk2,dTdy,Npa,xc,zc]=ShearFlowCenter
(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vz,Vx,xVz,zVx);
[trash1,trash2,trash3,s1,s2,s3,junk1,junk2,junk3,ex,junk4,junk5,junk6,junk7]=
ShearFlowCenter(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vz,0,xVz,zVx);
[trash1,trash2,trash3,s1,s2,s3,junk1,junk2,junk3,junk4,ez,junk5,junk6,junk7]=
ShearFlowCenter(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,0,Vx,xVz,zVx);
```

ShearFlowCenter.m--

```
%% Shear Flow and Shear Center Calculator FUNCTION

function [x1,x2,x3,z1,z2,z3,tau1,tau2,tau3,ex,ez,dTdy,Npa,xc,zc] =
ShearFlowCenter(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vz,Vx,xVz,zVx)

% function [tau1,tau2,tau3,ex,ez] =
ShearFlowCenter(xat,xab,xsp,At,Ab,Asp,tat,tab,tsp,Vz,Vx,xVz,zVx)

% This Function is used in Shear.m to compute shear flow and shear center
% individually

% xat:  x-locations of all stringers and skin subdivisions (NOT INCLUDING
%           SPAR CAPS) along top of wing, in order from LE to TE [ft]
% xab:  Same as xat except along bottom of wing
% xsp:  x-locations of spars, including final skin section [ft]
% At:   Cross-sectional area of all stringers and skin subdivisions (NOT
%           INCLUDING SPAR CAPS) top of wing, in order from LE to TE [ft^2]
% Ab:   Same as At except along bottom of wing
% Asp:  Cross-sectional area of spar caps
% tat:  Thickness of skin sections between adjacent stringers or skin
%           subdivisions (INCLUDING SPAR CAPS), in order from LE to TE [ft]
% tab:  Same as tat, except along bottom of wing
% tsp:  Thickness of spar caps
% Vz:   Shear force in positive z-direction
% Vx:   Shear force in positive x-direction
% xVz:  x-location where Vz acts
% zVx:  z-location where Vx acts

%% NACA Airfoil Data

c = 5;
m = .02;
p = .4;
t = .12;

%% Compute z-locations of all Stringers, skin subdivisions, and spar caps

for ii=1:length(xat)
    zat(ii)= c*(.5 * NACAthick(t, xat(ii)/c) + NACAcamber(m, p, xat(ii)/c));
end

for ii=1:length(xab)
    zab(ii)  = c*(-.5 * NACAthick(t, xab(ii)/c) + NACAcamber(m, p,
xab(ii)/c));
end

for ii=1:length(xsp)
    zsp(1,ii)= c*(.5 * NACAthick(t, xsp(ii)/c) + NACAcamber(m, p,
xsp(ii)/c));
    zsp(2,ii)= c*(-.5* NACAthick(t, xsp(ii)/c) + NACAcamber(m, p,
xsp(ii)/c));
end

%% Organize Stringers / Skin Subdivisions / Spar Caps and Areas for Shear
Flow Calcs

% Cell 1
ii=1;
while (ii <= length(xat) && xat(ii) < xsp(1))
    xat1temp(ii)=xat(ii);
    zat1temp(ii)=zat(ii);
    At1temp(ii)=At(ii);
    ii=ii+1;
end
N1t=length(xat1temp);
for ii=1:N1t
    xat1(ii)=xat1temp(N1t+1-ii);
    zat1(ii)=zat1temp(N1t+1-ii);
    At1(ii)=At1temp(N1t+1-ii);
end

ii=1;
while (ii <= length(xab) && xab(ii) < xsp(1))
    xab1(ii)=xab(ii);
    zab1(ii)=zab(ii);
    Ab1(ii)=Ab(ii);
    ii=ii+1;
end
N1b=length(xab1);

x1=[xsp(1) xat1 xab1 xsp(1)];
z1=[zsp(1,1) zat1 zab1 zsp(2,1)];
A1=[Asp(1,1) At1 Ab1 Asp(2,1)];

% Cell 2
ii=1;
while (ii+N1t <= length(xat) && xat(ii+N1t) < xsp(2))
    xat2temp(ii)=xat(ii+N1t);
    zat2temp(ii)=zat(ii+N1t);
    At2temp(ii)=At(ii+N1t);
    ii=ii+1;
end
N2t=length(xat2temp);
for ii=1:N2t
    xat2(ii)=xat2temp(N2t+1-ii);
    zat2(ii)=zat2temp(N2t+1-ii);
    At2(ii)=At2temp(N2t+1-ii);
end

ii=1;
while (ii+N1b <= length(xab) && xab(ii+N1b) < xsp(2))
    xab2(ii)=xab(ii+N1b);
    zab2(ii)=zab(ii+N1b);
    Ab2(ii)=Ab(ii+N1b);
    ii=ii+1;
end
N2b=length(xab2);

x2=[xsp(2) xat2 xsp(1) xsp(1) xab2 xsp(2)];
z2=[zsp(1,2) zat2 zsp(1,1) zsp(2,1) zab2 zsp(2,2)];
A2=[Asp(1,2) At2 Asp(1,1) Asp(2,1) Ab2 Asp(2,2)];

%Cell 3
ii=1;
if 1+N1t+N2t <= length(xat)
    for jj=1+N1t+N2t : length(xat)
        xat3temp(ii)=xat(jj);
        zat3temp(ii)=zat(jj);
        At3temp(ii)=At(jj);
        ii=ii+1;
    end
    N3t=length(xat3temp);
    for ii=1:N3t
        xat3(ii)=xat3temp(N3t+1-ii);
        zat3(ii)=zat3temp(N3t+1-ii);
        At3(ii)=At3temp(N3t+1-ii);
    end
else
    xat3=1:0;    % Empty Vectors
    zat3=1:0;
    At3=1:0;
    N3t=0;
end

ii=1;
if 1+N1b+N2b <= length(xab)
    for jj=1+N1b+N2b : length(xab)
        xab3(ii)=xab(jj);
        zab3(ii)=zab(jj);
        Ab3(ii)=Ab(jj);
        ii=ii+1;
    end
    N3b=length(xab3);
else
    xab3=1:0;   %Empty Vectors
    zab3=1:0;
    Ab3=1:0;
    N3b=0;
end

x3=[xsp(3) xat3 xsp(2) xsp(2) xab3 xsp(3)];
z3=[zsp(1,3) zat3 zsp(1,2) zsp(2,2) zab3 zsp(2,3)];
A3=[Asp(1,3) At3 Asp(1,2) Asp(2,2) Ab3 Asp(2,3)];

N11=N1t+1;
N12=N2t+2;
N13=N3t+2;
N21=N1b+1;
N22=N2b+2;
N23=N3b+2;
Npa=[N11 N12 N13 ; N21 N22 N23];

%% Compute Arc Lengths and Sweep Area

% Sweep Area is area between top of 1st spar and the web between any two
% adjacent spars/skin subdivisions/stringers

iinc=0.001;
NumWebsTop=N1t+N2t+N3t+3;

xatall=[0 xat(1:N1t) xsp(1) xat((N1t+1):(N1t+N2t)) xsp(2)
xat((N1t+N2t+1):(N1t+N2t+N3t)) xsp(3)];
xaball=[0 xab(1:N1b) xsp(1) xab((N1b+1):(N1b+N2b)) xsp(2)
xab((N1b+N2b+1):(N1b+N2b+N3b)) xsp(3)];

% xatall=sort([0 xat xsp]);
% xaball=sort([0 xab xsp]);

for ii=1 : (length(xatall)-1)
    x= xatall(ii) : iinc : xatall(ii+1);
    d=0;
    Aswp=0;
    z=c*(.5 * NACAthick(t, x(1)/c) + NACAcamber(m, p, x(1)/c));
    for jj=2 : (length(x))
        z(jj)=c*(.5 * NACAthick(t, x(jj)/c) + NACAcamber(m, p, x(jj)/c));
        dtemp=sqrt( (x(jj)-x(jj-1))^2 + (z(jj)-z(jj-1))^2);
        atemp=sqrt( (x(jj)-xsp(1))^2 + (z(jj)-zsp(1,1))^2);
        btemp=sqrt( (x(jj-1)-xsp(1))^2 + (z(jj-1)-zsp(1,1))^2);
        stemp=(atemp+btemp+dtemp)/2;
        Aswptemp=sqrt(stemp*(stemp-atemp)*(stemp-btemp)*(stemp-dtemp));
%Heron's Formula for area of triangle
        Aswp=Aswp+Aswptemp;
        d=d+dtemp;
    end
    dt(ii)=d;
    Aswpt(ii)=Aswp;
end

for ii=1 : (length(xaball)-1)
    x= xaball(ii) : iinc : xaball(ii+1);
    d=0;
    Aswp=0;
    z=c*(-.5 * NACAthick(t, x(1)/c) + NACAcamber(m, p, x(1)/c));
    for jj= 2 : (length(x))
        z(jj)=c*(-.5 * NACAthick(t, x(jj)/c) + NACAcamber(m, p, x(jj)/c));
        dtemp=sqrt( (x(jj)-x(jj-1))^2 + (z(jj)-z(jj-1))^2);
        atemp=sqrt( (x(jj)-xsp(1))^2 + (z(jj)-zsp(1,1))^2);
        btemp=sqrt( (x(jj-1)-xsp(1))^2 + (z(jj-1)-zsp(1,1))^2);
        stemp=(atemp+btemp+dtemp)/2;
        Aswptemp=sqrt(stemp*(stemp-atemp)*(stemp-btemp)*(stemp-dtemp));
%Heron's Formula for area of triangle
        Aswp=Aswp+Aswptemp;
        d=d+dtemp;
    end
    db(ii)=d;
    Aswpb(ii)=Aswp;
end

dsp=zsp(1,:)-zsp(2,:);

for ii=1:length(xsp)
    atemp=sqrt( (xsp(ii)-xsp(1))^2 + (zsp(1,ii)-zsp(1,1))^2);
    btemp=sqrt( (xsp(ii)-xsp(1))^2 + (zsp(2,ii)-zsp(1,1))^2);
    dtemp=dsp(ii);
    stemp=(atemp+btemp+dtemp)/2;
    Aswpsp(ii)=sqrt(stemp*(stemp-atemp)*(stemp-btemp)*(stemp-dtemp));
end

%% Organize Arc Lengths and Sweep Areas for Shear Flow Calcs

for ii=1:N1t+1
    d1t(ii)=dt(N1t+2-ii);
    Aswp1t(ii)=Aswpt(N1t+2-ii);
    t1t(ii)=tat(N1t+2-ii);
end

for ii=1:N1b+1
    d1b(ii)=db(ii);
    Aswp1b(ii)=Aswpb(ii);
    t1b(ii)=tab(ii);
end
```

```
for ii=1:N2t+1
    d2t(ii)=dt(N1t+N2t+3-ii);
    Aswp2t(ii)=Aswpt(N1t+N2t+3-ii);
    t2t(ii)=tat(N1t+N2t+3-ii);
end

for ii=1:N2b+1
    d2b(ii)=db(N1b+1+ii);
    Aswp2b(ii)=Aswpb(N1b+1+ii);
    t2b(ii)=tab(N1b+1+ii);
end

for ii=1:N3t+1
    d3t(ii)=dt(N1t+N2t+N3t+4-ii);
    Aswp3t(ii)=Aswpt(N1t+N2t+N3t+4-ii);
    t3t(ii)=tat(N1t+N2t+N3t+4-ii);
end

for ii=1:N3b+1
    d3b(ii)=db(N1b+N2b+2+ii);
    Aswp3b(ii)=Aswpb(N1b+N2b+2+ii);
    t3b(ii)=tab(N1b+N2b+2+ii);
end

n1t=length(d1t);
n1b=length(d1b);
n2t=length(d2t);
n2b=length(d2b);
n3t=length(d3t);
n3b=length(d3b);

d1=[d1t(1:n1t-1) , d1t(n1t)+d1b(1) , d1b(2:n1b) , dsp(1)];
Aswp1=[Aswp1t(1:n1t-1) , Aswp1t(n1t)+Aswp1b(1) , Aswp1b(2:n1b) , Aswpsp(1)];
t1=[t1t(1:n1t-1) , t1t(n1t), t1b(2:n1b) , tsp(1)];
delta1=d1./t1;

d2=[d2t dsp(1) d2b dsp(2)];
Aswp2=[Aswp2t Aswpsp(1) Aswp2b Aswpsp(2)];
t2=[t2t tsp(1) t2b tsp(2)];
delta2=d2./t2;

d3=[d3t dsp(2) d3b dsp(3)];
Aswp3=[Aswp3t Aswpsp(2) Aswp3b Aswpsp(3)];
t3=[t3t tsp(2) t3b tsp(3)];
delta3=d3./t3;



%% Compute Cell Areas

hsp=xsp/c;
iinc=0.001;
```

```
%Cell 1
x = 0 : iinc : hsp(1);
for i = 1 : length(x)
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;
Acell1=iinc*c*(abs(sum(z(1,:)))+abs(sum(z(2,:))));

%Cell 2
z=0;
x=hsp(1):iinc:hsp(2);
for i = 1 : length(x)
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;
Acell2=iinc*c*(abs(sum(z(1,:)))+abs(sum(z(2,:))));

%Cell 3
z=0;
x = [hsp(2) : iinc : .75];
for i = 1 : length(x)
    z(1, i) = (.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
    z(2, i) = (-.5 * NACAthick(t, x(i)) + NACAcamber(m, p, x(i)));
end
x = x * c;
z = z * c;
Acell3=iinc*c*(abs(sum(z(1,:)))+abs(sum(z(2,:))));

%% Compute Centroid and Moment of Inertias, and Change Coordinates to
Centroid-Centered Axes

Atot=sum(At)+sum(Ab)+sum(sum(Asp));
xc=(sum(At.*xat)+sum(Ab.*xab)+sum(sum(Asp.*[xsp;xsp])))/Atot;
zc=(sum(At.*zat)+sum(Ab.*zab)+sum(sum(Asp.*zsp)))/Atot;

xatc=xat-xc;
xabc=xab-xc;
xspc=xsp-xc;
zatc=zat-zc;
zabc=zab-zc;
zspc=zsp-zc;

Ix=sum(At.*zatc.^2)+sum(Ab.*zabc.^2)+sum(sum(Asp.*zspc.^2));
Iz=sum(At.*xatc.^2)+sum(Ab.*xabc.^2)+sum(sum(Asp.*[xspc;xspc].^2));
Ixz=sum(At.*xatc.*zatc)+sum(Ab.*xabc.*zabc)+sum(sum(Asp.*[xspc;xspc].*zspc));

kx=Ix/(Ix*Iz-Ixz^2);
kz=Iz/(Ix*Iz-Ixz^2);
kxz=Ixz/(Ix*Iz-Ixz^2);

kA=kx*Vx-kxz*Vz;
```

```
kB=kz*Vz-kxz*Vx;

x1=x1-xc;
x2=x2-xc;
x3=x3-xc;
z1=z1-zc;
z2=z2-zc;
z3=z3-zc;

%% -------------Bending Shear Flows------------------

%% Cell 1 Except 1st Spar

N1=length(x1);

Qz=A1.*x1;
Qx=A1.*z1;

qadd1=-kA*Qz-kB*Qx;

qb1=0;
for ii=2:(N1-1)
    qb1(ii)=qb1(ii-1)+qadd1(ii);
end

%% Cell 2 Except 2nd Spar

N2=length(x2);

Qz=A2.*x2;
Qx=A2.*z2;

qadd2=-kA*Qz-kB*Qx;

qb2=0;
for ii = 2:(N2-1)
    if ii == N2t+3
        qb2(ii)=qb2(ii-1)+qadd2(ii)+qb1(N1-1);
    else
        qb2(ii)=qb2(ii-1)+qadd2(ii);
    end
end

%% Cell 3

N3=length(x3);

Qz=A3.*x3;
Qx=A3.*z3;

qadd3=-kA*Qz-kB*Qx;

qb3=0;
```

```
for ii = 2:N3
    if ii == N3t+3
        qb3(ii)=qb3(ii-1)+qadd3(ii)+qb2(N2-1);
    else
        qb3(ii)=qb3(ii-1)+qadd3(ii);
    end
end


qb1=[qb1 -qb2(N2t+2)];
qb2=[qb2 -qb3(N3t+2)];

%% ----------------- Torsional Shear Flows--------------------

Gref=564480000;%27.6*20885434.2; %Shear Modulus [GPa -> lbf/ft^2]

C1=1/(2*Acell1*Gref);
C2=1/(2*Acell2*Gref);
C3=1/(2*Acell3*Gref);

q1a=C1*sum(delta1);
q1b=-C1*delta1(length(delta1));
q1c=0;
q1coeff=[q1a q1b q1c -1];
q1const=C1*sum(qb1.*delta1);

q2a=-C2*delta1(length(delta1));
q2b=C2*sum(delta2);
q2c=-C2*delta2(length(delta2));
q2coeff=[q2a q2b q2c -1];
q2const=C2*sum(qb2.*delta2);

q3a=0;
q3b=-C3*delta2(length(delta2));
q3c=C3*sum(delta3);
q3coeff=[q3a q3b q3c -1];
q3const=C3*sum(qb3.*delta3);

qma=2*Acell1;
qmb=2*Acell2;
qmc=2*Acell3;
qmV=Vz*(xVz-x1(1)-xc)-Vx*(zVx-z1(1)-zc);  % torque from shear force about top
of 1st spar
qmcoeff=[qma qmb qmc qmV];
qmconst=2*sum(Aswp1(1:(N1-1)).*qb1(1:(N1-1)))+2*sum(Aswp2(1:(N2-
1)).*qb2(1:(N2-1)))...
    +2*sum(Aswp3(1:N3).*qb3(1:N3)); % torque from shear flows about top of
1st spar

qsmat=[q1coeff;q2coeff;q3coeff;qmcoeff];
qsconst=-[q1const;q2const;q3const;qmconst];

qs=inv(qsmat)*qsconst;
```

```
qs1=qs(1);
qs2=qs(2);
qs3=qs(3);
dTdy=qs(4);

qs1;
qs2;
qs3;

%% ----------------Total Shear Flow-------------------------

%Total Shear Flow in Cell 1

for ii=1:N1-1
    q1(ii)=qb1(ii)+qs1;
end
q1(N1)=qb1(N1)+qs1-qs2;

%Total Shear Flow in Cell 2

for ii=1:N2-1
    if ii < N2t+2 || ii > N2t+2
        q2(ii)=qb2(ii)+qs2;
    elseif ii == N2t+2
        q2(ii)=qb2(ii)+qs2-qs1;
    end
end
q2(N2)=qb2(N2)+qs2-qs3;

%Total Shear Flow in Cell 3

for ii=1:N3
    if ii < N3t+2 || ii > N3t+2
        q3(ii)=qb3(ii)+qs3;
    elseif ii == N3t+2
        q3(ii)=qb3(ii)+qs3-qs2;
    end
end

q1;

tau1=q1./t1;
tau2=q2./t2;
tau3=q3./t3;

%% Shear Center Calculator

qsmat1=[q1a q1b q1c ; q2a q2b q2c ; q3a q3b q3c];
qsconst1=-[q1const ; q2const ; q3const];
qssc=inv(qsmat1)*qsconst1;
qmcoeff=[qma qmb qmc];

if Vz == 0
    ex=0;
```

```
else
    ex=(qmconst+sum(qssc'.*qmcoeff))/Vz + xsp(1);
end
if Vx == 0
    ez=0;
else
    ez=(qmconst+sum(qssc'.*qmcoeff))/Vx + zsp(1,1);
end
```

ShearFlowInput.m--

```
function [xaat, xaab, Aat, Aab, Aasp] = ShearFlowInput(xt, xb, xsp, At, Ab,
Asp, tskin, tsp, Nstc, c, t, m, p, Mx, Mz, Ix, Iz, Ixz, xc, zc)

N1t=Nstc(1,1);
N2t=Nstc(1,1)+Nstc(1,2);
N3t=Nstc(1,1)+Nstc(1,2)+Nstc(1,3);
N1b=Nstc(2,1);
N2b=Nstc(2,1)+Nstc(2,2);
N3b=Nstc(2,1)+Nstc(2,2)+Nstc(2,3);

xallt=[0 xt(1:N1t) xsp(1) xt((N1t+1):N2t) xsp(2) xt((N2t+1):N3t) xsp(3)];
xallb=[0 xb(1:N1b) xsp(1) xb((N1b+1):N2b) xsp(2) xb((N2b+1):N3b) xsp(3)];
Aallt=[0 At(1:N1t) Asp(1) At((N1t+1):N2t) Asp(2) At((N2t+1):N3t) Asp(3)];
Aallb=[0 Ab(1:N1b) Asp(1) Ab((N1b+1):N2b) Asp(2) Ab((N2b+1):N3b) Asp(3)];

nt=length(xallt);

for ii=1:nt-1
    xinc=(xallt(ii+1)-xallt(ii))/10;
    for jj=1:10
            xat((ii-1)*10+jj)=xallt(ii)+xinc*(jj-1);
            zat((ii-1)*10+jj)=(NACAcamber(m, p,
xat(ii)/c)+NACAthick(t,xat(ii)/c)/2)*c;
    end
end
xat((nt-1)*10+1)=xallt(length(xallt));
zat((nt-1)*10+1)=(NACAcamber(m, p, xat((nt-1)*10+1)/c)+NACAthick(t,xat((nt-
1)*10+1)/c)/2)*c;

nb=length(xallb);

for ii=1:nb-1
    xinc=(xallb(ii+1)-xallb(ii))/10;
    for jj=1:10
            xab((ii-1)*10+jj)=xallb(ii)+xinc*(jj-1);
            zab((ii-1)*10+jj)=(NACAcamber(m, p, xab(ii)/c)-
NACAthick(t,xab(ii)/c)/2)*c;
    end
end
xab((nb-1)*10+1)=xallb(length(xallb));
zab((nb-1)*10+1)=(NACAcamber(m, p, xab((nb-1)*10+1)/c)-NACAthick(t,xab((nb-
1)*10+1)/c)/2)*c;

for ii=1:length(xat)
```

```
    if ii==1
        d1=sqrt((xat(ii+1)-xat(ii))^2+(zat(ii+1)-zat(ii))^2);
        sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii)-xc, zat(ii)-zc);
        sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii+1)-xc, zat(ii+1)-zc);
        At(ii)=tskin*d1/6*(2+sigma2/sigma1);
    elseif ii==length(xat)
        d1=sqrt((xat(ii)-xat(ii-1))^2+(zat(ii)-zat(ii-1))^2);
        sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii-1)-xc, zat(ii-1)-zc);
        sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii)-xc, zat(ii)-zc);
        At(ii)=tskin*d1/6*(2+sigma1/sigma2);
    else
        d2=sqrt((xat(ii+1)-xat(ii))^2+(zat(ii+1)-zat(ii))^2);
        d1=sqrt((xat(ii)-xat(ii-1))^2+(zat(ii)-zat(ii-1))^2);
        sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii-1)-xc, zat(ii-1)-zc);
        sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii)-xc, zat(ii))-zc;
        sigma3=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ii+1)-xc, zat(ii+1)-zc);
        At(ii)=tskin*d1/6*(2+sigma1/sigma2)+tskin*d2/6*(2+sigma3/sigma2);
    end
end

for ii=1:length(xab)
    if ii==1
        d1=sqrt((xab(ii+1)-xab(ii))^2+(zab(ii+1)-zab(ii))^2);
        sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii)-xc, zab(ii)-zc);
        sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii+1)-xc, zab(ii+1)-zc);
        Ab(ii)=tskin*d1/6*(2+sigma2/sigma1);
    elseif ii==length(xab)
        d1=sqrt((xab(ii)-xab(ii-1))^2+(zab(ii)-zab(ii-1))^2);
        sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii-1)-xc, zab(ii-1)-zc);
        sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii)-xc, zab(ii)-zc);
        Ab(ii)=tskin*d1/6*(2+sigma1/sigma2);
    else
        d2=sqrt((xab(ii+1)-xab(ii))^2+(zab(ii+1)-zab(ii))^2);
        d1=sqrt((xab(ii)-xab(ii-1))^2+(zab(ii)-zab(ii-1))^2);
        sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii-1)-xc, zab(ii-1)-zc);
        sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii)-xc, zab(ii))-zc;
        sigma3=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ii+1)-xc, zab(ii+1)-zc);
        Ab(ii)=tskin*d1/6*(2+sigma1/sigma2)+tskin*d2/6*(2+sigma3/sigma2);
    end
end

for ii=1:length(xallt)
    At(1+(ii-1)*10)=At(1+(ii-1)*10)+Aallt(ii);
end

for ii=1:length(xallb)
    Ab(1+(ii-1)*10)=Ab(1+(ii-1)*10)+Aallb(ii);
end



ispt(1)=N1t*10+11;
ispt(2)=N2t*10+21;
ispt(3)=N3t*10+31;
ispb(1)=N1b*10+11;
ispb(2)=N2b*10+21;
ispb(3)=N3b*10+31;
```

```
for ii=1:length(xsp)
    d=zat(ispt(ii))-zab(ispb(ii));
    sigma1=bendStress(Mx, Mz, Ix, Iz, Ixz, xat(ispt(ii))-xc, zat(ispt(ii))-
zc);
    sigma2=bendStress(Mx, Mz, Ix, Iz, Ixz, xab(ispb(ii))-xc, zab(ispb(ii))-
zc);
    Atadd=tskin*d/6*(2+sigma2/sigma1);
    Abadd=tskin*d/6*(2+sigma1/sigma2);
    At(ispt(ii))=At(ispt(ii))+Atadd;
    Ab(ispb(ii))=Ab(ispb(ii))+Abadd;
end

for ii=1:3
    Aasp(1,ii)=At(ispt(ii));
end

for ii=1:3
    Aasp(2,ii)=Ab(ispb(ii));
end



xaat(1:ispt(1)-1)=xat(1:ispt(1)-1);
xaat(ispt(1):ispt(2)-2)=xat(ispt(1)+1:ispt(2)-1);
xaat(ispt(2)-1:ispt(3)-3)=xat(ispt(2)+1:ispt(3)-1);

xaab(1:ispb(1)-1)=xab(1:ispb(1)-1);
xaab(ispb(1):ispb(2)-2)=xab(ispb(1)+1:ispb(2)-1);
xaab(ispb(2)-1:ispb(3)-3)=xab(ispb(2)+1:ispb(3)-1);

Aat(1:ispt(1)-1)=At(1:ispt(1)-1);
Aat(ispt(1):ispt(2)-2)=At(ispt(1)+1:ispt(2)-1);
Aat(ispt(2)-1:ispt(3)-3)=At(ispt(2)+1:ispt(3)-1);

Aab(1:ispb(1)-1)=Ab(1:ispb(1)-1);
Aab(ispb(1):ispb(2)-2)=Ab(ispb(1)+1:ispb(2)-1);
Aab(ispb(2)-1:ispb(3)-3)=Ab(ispb(2)+1:ispb(3)-1);
```

spanPolyInt.m--

```
% Computes a polynomial fit of wing loading and then integrates that fit to
% find shear and moment loading. Input X-loading, Z-loading, polynomial
% interpolation order respectively.

function [x,z] = spanPolyInt(Xload,Zload,n)

span         = [0:0.1:15];

[a,b]        = size(Xload);

x            = zeros(a, b);
z            = zeros(a, b);
```

```
for ii = 1:a

    xpoly       = polyfit(span,Xload(ii,:),n);
    zpoly       = polyfit(span,Zload(ii,:),n);
    intx        = polyint(xpoly);
    intz        = polyint(zpoly);

%     figure(1);
%     plot(span,xpoly)
%     hold on;

    for jj = 1:b

        xstep   = polyval(intx,span(jj));
        zstep   = polyval(intz,span(jj));

        x(ii,jj)= xstep;
        z(ii,jj)= zstep;

    end

%     figure(2);
%     plot(span,x)
%     hold on;

end
```

torqueFlow.m---

```
% Computes torsional shear flow due to pure torque

function [qMy, dTdyMac, Lskin, Arib] = torqueFlow(My, xs, hs, ts, c, tskin,
t, m, p, G)

% torsional shear flows
[A1, A2, A3] = cellArea(xs, c, t, m, p);    % cell area calculation

inc = .001;

xskin1 = [0 : inc : (xs(1) / c)] ;
xskin2 = [(xs(1) / c) : inc : (xs(2) / c)];
xskin3 = [(xs(2) / c) : inc : .75];

for i = 1 : max(size(xskin1))
    zskin1(1, i) = (.5 * NACAthick(t, xskin1(i)) + NACAcamber(m, p,
xskin1(i)));
    zskin1(2, i) = (-.5 * NACAthick(t, xskin1(i)) + NACAcamber(m, p,
xskin1(i)));
end
for i = 1 : max(size(xskin2))
    zskin2(1, i) = (.5 * NACAthick(t, xskin2(i)) + NACAcamber(m, p,
xskin2(i)));
```

```
    zskin2(2, i) = (-.5 * NACAthick(t, xskin2(i)) + NACAcamber(m, p,
xskin2(i)));
end
for i = 1 : max(size(xskin3))
    zskin3(1, i) = (.5 * NACAthick(t, xskin3(i)) + NACAcamber(m, p,
xskin3(i)));
    zskin3(2, i) = (-.5 * NACAthick(t, xskin3(i)) + NACAcamber(m, p,
xskin3(i)));
end

zskin1 = zskin1 * 5;
zskin2 = zskin2 * 5;
zskin3 = zskin3 * 5;
xskin1 = xskin1 * 5;
xskin2 = xskin2 * 5;
xskin3 = xskin3 * 5;

for i = 1 : (max(size(xskin1)) - 1)
    lskin1(i) = sqrt((xskin1(i) - xskin1(i + 1)) ^ 2 + (zskin1(1, i) -
zskin1(1, i + 1)) ^ 2) ...
        + sqrt((xskin1(i) - xskin1(i + 1)) ^ 2 + (zskin1(2, i) - zskin1(2, i
+ 1)) ^ 2);
end
for i = 1 : (max(size(xskin2)) - 1)
    lskin2(i) = sqrt((xskin2(i) - xskin2(i + 1)) ^ 2 + (zskin2(1, i) -
zskin2(1, i + 1)) ^ 2) ...
        + sqrt((xskin2(i) - xskin2(i + 1)) ^ 2 + (zskin2(2, i) - zskin2(2, i
+ 1)) ^ 2);
end
for i = 1 : (max(size(xskin3)) - 1)
    lskin3(i) = sqrt((xskin3(i) - xskin3(i + 1)) ^ 2 + (zskin3(1, i) -
zskin3(1, i + 1)) ^ 2) ...
        + sqrt((xskin3(i) - xskin3(i + 1)) ^ 2 + (zskin3(2, i) - zskin3(2, i
+ 1)) ^ 2);
end

L1 = sum(lskin1);
L2 = sum(lskin2);
Lbs = abs(zskin3(1, end) - zskin3(2, end));   % back skin length
L3 = sum(lskin3) + Lbs;
Lskin = L1 + L2 + L3;
Arib = A1 + A2 + A3;

SYSTEM(1, 1) = 2 * A1;
SYSTEM(1, 2) = 2 * A2;
SYSTEM(1, 3) = 2 * A3;
SYSTEM(2, 1) = L1 / (A1 * tskin) + hs(1) / (A1 * ts) + hs(1) / (A2 * ts);
SYSTEM(2, 2) = -L2 / (A2 * tskin) - hs(1) / (A1 * ts) + (-hs(1) - hs(2)) /
(A2 * ts);
SYSTEM(2, 3) = hs(2) / (A2 * ts);
SYSTEM(3, 1) = -hs(1) / (A2 * ts);
SYSTEM(3, 2) = L2 / (A2 * tskin) + hs(1) / (A2 * ts) + hs(2) / (A2 * ts) ...
    + hs(2) / (A3 * ts);
SYSTEM(3, 3) = -L3 / (A3 * tskin) - hs(2) / (A3 * ts) - hs(2) / (A2 * ts);

KNOWN = [My; 0; 0];
```

```
qMy = inv(SYSTEM) * KNOWN;

dTdyMac = 1 / (2 * A1 * G) * (qMy(1) * L1 / tskin + (qMy(1) - qMy(2)) * hs(1)
/ ts);
```

VonMises.m---

```
% Computes Von Mises equivalent stress

function [SIGvm] = VonMises(SIGx, SIGy, SIGz, TAUxy, TAUyz, TAUxz)

% INPUT:
% SIGx, SIGy & SIGz      -   normal stresses in the x, y and z directions
% TAUxy, TAUyz & TAUxz   -   shear stresses in the xy, yz and xz planes
%
% OUTPUT:
% SIGvm          -   equivalent Von Mises normal stress

SIGvm = sqrt(((SIGx - SIGy) ^ 2 + (SIGy - SIGz) ^ 2 + (SIGx - SIGz) ^ 2 + ...
    6 * (TAUxy ^ 2 + TAUyz ^ 2 + TAUxz ^ 2)) / 2);
```