NASA/TM–2017-219375

Parametric Structural Model for a Mars Entry Concept

Brittney M. Lane and Samee W. Ahmed
Langley Research Center, Hampton, Virginia

February 2017

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM–2017-219375

Parametric Structural Model for a Mars Entry Concept

Brittney M. Lane and Samee W. Ahmed
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

February 2017

Acknowledgments

This work was made possible by the NASA Internships, Fellowships, Scholarships (NIFS) program at NASA Langley Research Center. I would further like to acknowledge Jamshid Samareh and Sandra Walker for giving us exciting work for our internship and mentoring us along the way. This work would not have been completed without the help of Sasan Armand and Lance Proctor. They were extremely helpful in creating the Nastran model. I would also like to thank the rest of the Vehicle Analysis Branch and NASA Langley coworkers for always being willing to help.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Table of Contents

Abstract

This paper outlines the process of developing a parametric model for a vehicle that can withstand Earth launch and Mars entry conditions. This model allows the user to change a variety of parameters ranging from dimensions and meshing to materials and atmospheric entry angles to perform finite element analysis on the model for the specified load cases. While this work focuses on an aeroshell for Earth launch aboard the Space Launch System (SLS) and Mars entry, the model can be applied to different vehicles and destinations. This specific project derived from the need to deliver large payloads to Mars efficiently, safely, and cheaply. Doing so requires minimizing the structural mass of the body as much as possible. The code developed for this project allows for dozens of cases to be run with the single click of a button. The end result of the parametric model gives the user a sense of how the body reacts under different loading cases so that it can be optimized for its purpose. The data are reported in this paper and can provide engineers with a good understanding of the model and valuable information for improving the design of the vehicle. In addition, conclusions show that the frequency analysis drives the design and suggestions are made to reduce the significance of normal modes in the design.

1 Introduction

It has become increasingly important to develop the technologies and capabilities to land large payloads, approximately 100 metric tons (Mt), on Mars for both robotic and human missions. The Entry, Descent, and Landing Systems Analysis reviewed several preliminary methods of successfully landing these payloads [1]. This paper focuses on the rigid Mid Lift-to-Drag (L/D) ratio aeroshell. It is based on the "ellipsled" aeroshell for an Ares V shroud used in the Design Reference Architecture 5.0 (DRA5) study. With this being a high cost and high risk mission, the need for a parametric model arose.

The parametric model developed in this paper has several uses. The primary goal was the reduction of mass. Using this parametric model, the structure can be optimized for mass according to the loads applied to it. A secondary goal was the ability to test differently sized aeroshells and vehicles with different structures. With this automated analysis, the user can feed the program a spreadsheet of data with changing diameters, lengths, combinations of stiffeners, payload masses, etc. and each of these designs can be optimized for mass. Lastly, another goal of this project was the ability to test new materials for the structure and thermal protection system (TPS). This last point was not explored due to time constraints; however, the model is set up for this analysis.

The parametric model was developed using the Python programming language. The Python code outputs bulk data files and necessary case control input files for three different Nastran analyses: Static, Frequency Response, and Design Optimization. After creating the necessary Nastran analysis input files for each model, the analyses are run in Nastran. The data files are saved to a folder titled with the run number. This automation is advantageous in that hundreds of cases can be run without the need to be monitored by the user.

2 Model Overview

2.1 Shape & Structure

The body of the vehicle is cylindrical with a hemispherical nose. The baseline model is 10 meters in diameter and 30 meters in length. This is an idealized representation of the structure's geometry.

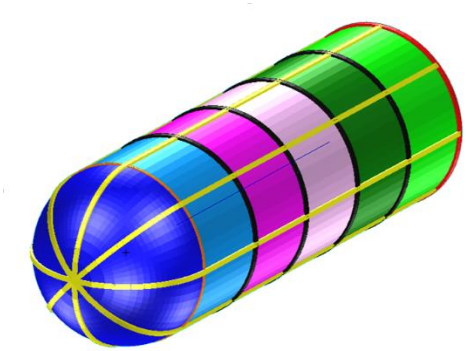

Figure 1: Mid-L/D Mars entry concept shape with color-coded properties.

The Mars Entry concept is divided into a number of sections that the user can specify. The baseline model has six shell sections: one nose section and five body sections. It is divided in this manner in order to assign different materials and properties to different sections. Frames separated the different sections. This grouping is beneficial when using SOL200 (Nastran Design Optimization solution) to optimize shell thicknesses.

At equally spaced instances along the length, there are frames (or rings) supporting the structure. The number of frames can be specified by the user. The same material is assigned to each frame. However, the frames are divided into three groups with different properties in order to optimize the cross-section dimensions independently. The payload mass will be carried by all but the first and last frames of the vehicle. The load carrying frames are colored black in Figure 1. The first frame is colored orange and the last frame is colored red. The first frame, middle frames, and last frame make-up the three groups.

Longerons run the length of the vehicle from the nose tip to the end. In order to simplify optimization, the longerons are assigned the same property and material. This property group is colored yellow in Figure 1. As with the frames, the user can specify the number of longerons to be generated in the model.

In the baseline model, all the shell and bar elements are assigned the same materials. However, this can be changed by the user. The ability to change materials allows the user to test new materials to ensure the load carrying capabilities of the material.

2.2 Purpose

A parametric model for the Mid-L/D model was created in order to optimize the design for critical load cases and payload sizes. With this parametric model, the user has the ability to test various designs, launch vehicles, entry conditions, materials, and structures in order to develop the most feasible design for a Mars Entry vehicle. Data from the simulations and sensitivity analysis allow the significant factors to be identified. Knowing the most influential design factors will aid design engineers in creating the optimal vehicle with the optimal

structure, thermal protection system, materials, and payload mass for the given mission. The parametric model may even be used to reduce the mass of the structure to perform under the critical load cases rather than sacrificing the payload and requiring that its weight be minimized.

3 Parametric Model Attributes

3.1 User Inputs

The user inputs template is a Microsoft Excel workbook that organizes all the design variables that the parametric model can accept. The title row explains each of the inputs and the corresponding units. All the columns must be filled with the required information in order for the program to run a case. Table 1 shows the first seven columns in the excel template. The user has the ability to enter as many cases as needed with each row signifying a new case. Values must be entered in each column for there to be a successful run. When all the information has been entered, the file must be saved as a comma separated file entitled "UserInputs.csv". All the inputs are now ready to be read in by Python. A list of user inputs can be seen in Table 2.

Table 1: User Inputs template format.

Case	Dimensions		Meshing		Structure		
#	Diameter (m)	Length (m)	Number of Elements Along Cylindrical Edge	Number of Elements Along Circumference	Number Of Frame Rings	Number of Longerons	Number of Shell Sections
1	10	30	60	64	6	8	6
2	12	30	60	64	6	8	6

Initially, the only inputs were two dimensions (length and diameter) and two variables controlling the mesh. This was expanded to include additional design inputs. The goal was to make the parametric model as user friendly and versatile as possible in order to test unique design ideas and scenarios. Allowing the user to specify information such as angle of attack, dynamic pressure, number of longerons, and the materials made the parametric model more functional.

3.1.1 Units

All measurements are in the base SI units. The dimensions are in meters and the pressure is in Pascals. All material properties are stated in the SI units and must remain that way in order to stay consistent with units and maintain accurate results. Angles take a user input of degrees and the code changes it to radians for the purpose of the calculations.

3.1.2 User Inputs in Coding

Table 2 shows how the user inputs are implemented in the code. All the user inputs are stored in an array. Some of the elements in the array are assigned a specific variable that is used

throughout the code. The notes list some important relations as well as figures that depict the use of that input.

Table 2: Using inputs in the code.

User Input	Units	Stored in Array	Corresponding Variable in code	Notes
Diameter	meters	Dimensions	d	Used to calculate r: r = d/2 Figure 3 May be an integer or decimal value.
Length	meters	Dimensions	L	Used to calculate l: l=L-(d/2) Figure 3 May be an integer or decimal value.
Number of Elements along Cylindrical Edge	#	Meshing	edgeelems	Used to calculate edgeelem: edgeelem = l/edgeelems Figure 3
Number of Elements along Circumference	#	Meshing	ringelem	Figure 5
Number of Frame Rings	#	Structure	NumOfRings	Figure 15
Number of Longerons	#	Structure	NumOfLongerons	Figure 1
Number of Shell Sections	#	Structure	NumOfSections	Figure 15
Payload Mass	kilograms	Mass	PayloadMass	
Area Density of TPS	kg/m^2	TPS	TPS	
Density	kg/m^3	ShellMatl, LongMatl, Frame Matl	ShellMatl[0], ShellMatl[6], LongMatl[0], FrameMatl[0]	
Young's Modulus	Pascals	ShellMatl, LongMatl, Frame Matl	ShellMatl[1], ShellMatl[7], LongMatl[1], FrameMatl[1]	
Shear Modulus	Pascals	ShellMatl, LongMatl, Frame Matl	ShellMatl[2], ShellMatl[8], LongMatl[2], FrameMatl[2]	

Thickness Lower Limit	meters	ShellMatl, LongMatl, Frame Matl	ShellMatl[3], ShellMatl[9], LongMatl[3], FrameMatl[3]	
Thickness Upper Limit	meters	ShellMatl, LongMatl, Frame Matl	ShellMatl[4], ShellMatl[10], LongMatl[4], FrameMatl[4]	
Allowable Stress	Pascals	ShellMatl, LongMatl, Frame Matl	ShellMatl[5], ShellMatl[11], LongMatl[5], FrameMatl[5]	
Allowable Shear Stress	Pascals	ShellMatl	SB	
Entry Dynamic Pressure	Pascals	Pressure Loads	q	
Entry Angle of Attack	degrees	Pressure Loads	alpha	Figure 21
Entry Angle of Sideslip	degrees	Pressure Loads	beta	
Launch Dynamic Pressure	Pascals	Pressure Loads	q	
Launch Angle of Attack	degrees	Pressure Loads	alpha	Figure 21
Launch Angle of Sideslip	degrees	Pressure Loads	beta	
Launch G Loads	filepath	LaunchFiles	-	File is read by code and data are assigned to variables: axial and lateral
Vibrations	filepath	LaunchFiles	-	

3.1.3 Choosing Good Inputs

Special attention is required for the inputs relating to the shape and size of the model. Because of the way the aeroshell is calculated and modeled, the subsequent rules must be followed when selecting dimensions, meshing details, or structures.

1. The number of nodes in a ring should be evenly divisible by the number of longerons. This is done to ensure that the longerons are evenly spaced and can be defined by existing nodes.
2. The number of elements along the cylindrical edge must be evenly divisible by the number of shell sections. This guarantees that the shell sections are of equal length and can be defined using the existing nodes.

Equation 1: Rule #2

$$\frac{Number\ of\ Elements}{Number\ of\ Shell\ Sections} = integer \quad (1)$$

3. The number of elements along the cylindrical edge should be evenly divisible by the number of frames. Again, this ensures that the frames are equally spaced and can be defined using existing nodes.

Equation 2: Rule #3

$$\frac{Number\ of\ Elements}{Number\ of\ Frames - 1} = integer \quad (2)$$

3.2 Nastran Cards

Table 3 lists the Nastran cards used to develop the nodes, mesh, and load cases for the model. This table provides reference material and information related to the code implementation.

Table 3: Nastran cards used to develop model.

Card	Definition	Necessary Information
CBAR	Defines the bar elements in the model.	2 grid points, a reference grid point for orientation, and a property ID (PBARL)
CONM2	Defines a concentrated point mass element.	Node location and mass value
CQUAD4	Defines the shell elements on the body.	Element ID and 4 grid points
CROD	Defines a rod element connection.	2 grid points and a property ID (PROD)
CTRIA3	Defines the shell elements at the nose tip.	Element ID and 3 grid points
GRAV	Defines g loads.	Acceleration magnitude and direction vector
GRID	Defines the grid points (nodes) of the model.	Node ID, x-,y-, and z-locations
LOAD	Defines a load set.	Load ID numbers (SID)
MAT1	Defines the isotropic materials in the model.	Density, elastic modulus, and shear modulus or Poisson's ratio
PBARL	Assigns the bar properties, such as cross-section.	Material ID, cross-section shape, and dimensions
PCOMP	Assigns the shell elements a composite property.	Allowable shear stress and MAT1 entries
PLOAD4	Defines the pressure load on the face of a shell element.	Set ID, Element ID, Pressure loads at the nodes defining the element.

PROD	Defines the properties for the CRODs.	Material ID, cross-sectional area, and torsional constant
RBE3	Defines the rigid body attachments of the payload mass to the frames.	Node location of mass and attachment point to frame, constraints of each node
SPC1	Defines a set of single-point constraints.	Constraints (123456) and grid point ID's

3.3 Grid Points

The grid points of the Mars Entry Concept are generated by the Nodes.py python script. This portion of the code takes inputs from the User Inputs file (Table 4) and calculates the grid location of each node. It is important to note that the script generates node and element data without geometry features. The output is a text file with the grid points written to it in the correct format for a Nastran input file. This is later appended to the full BDF file titled "BDF_Case_#.BDF".

Table 4: Nodes function inputs.

Value	Type	Definition
Dimensions	Array	Contains user inputs of model dimensions from excel spreadsheet (length and diameter).
Mesh	Array	Contains user inputs for the mesh sizing (ringelem and edgeelems)
filename	String	Title for the nodes text file. It is called "nodesname" in the MidLOverD.py script.

3.3.1 Coordinate System

The origin is located on the axis of symmetry of the model, and the x=0 location is at the beginning edge of the cylindrical section (i.e. the dome is located where x<0 and the cylindrical section is located where x>0). The positive x-axis is directed towards the aft of the vehicle. The positive z-axis is directed out of the paper and the positive y-axis is orthogonal to both the x- and z-axes as illustrated in Figure 2.

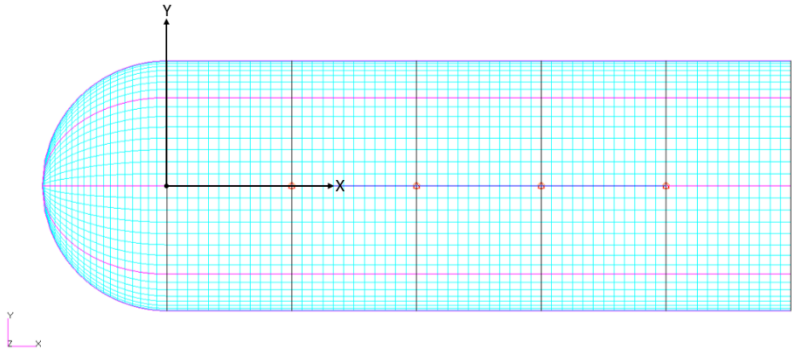

Figure 2: Origin and coordinate system of model

3.3.2 Dimensions and Definitions

The model dimensions and variable definitions are shown in Figure 3. A single element on the barrel section has a length, edgeelem. In the grid points code, x, y, and z locations are

determined for each node on a ring. "Rings" are the y-z cross-sections of the model and are referred to throughout the paper and codes.

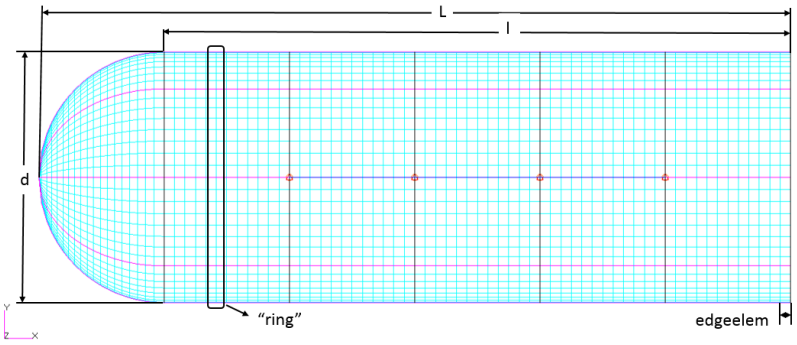

Figure 3: Model dimensions and definitions.

3.3.3 Nose

3.3.3.1 Method

The nose tip is only one grid point; therefore, it is generated separately from the other nose grid points. For the other nose grid points, the x, y, and z locations of each grid point were calculated in two steps. The first step finds the x-location of the ring. Since the nose is rounded, the rings cannot be evenly spaced as this will result in a poor mesh and a pointed nose tip. An equation was developed to ensure that there was a higher density of rings near the nose tip.

The second step finds the y and z locations of each node. Knowing the x-location of the ring and recognizing that the nose is a hemisphere, the corresponding radius of the ring on the y-z plane can be calculated. Each ring has a set number of grid points specified by the user and imported as the variable "ringelem". With the new radius of the ring on the y-z plane, the y and z locations of each node are generated with a total number of equally spaced nodes equal to the user input, "ringelem."

The x, y, and z values are stored in an array with the corresponding node ID number and the Nastran card name, "GRID." Each node must have its own unique node ID number.

3.3.3.2 Equations

3.3.3.2.1 Nose Tip Bias and x-locations

The total number of rings in the nose is calculated in Equation 3 by dividing the radius of the nose by the "edgeelem" variable to get the number of equally spaced rings in the nose and is stored in the variable "nmax". But, since the nose needs to be biased, this value is multiplied by a factor of 1.5. This is essentially taking the density of rings in the body and multiplying it by the factor of 1.5 to get a density of rings in the nose 1.5 times that of the body. This number is rounded to the nearest whole number of rings. Relating the number of rings in the nose to the number of rings in the barrel ensures that the mesh changes together. If the user decides to refine the mesh, the mesh will refine overall. There is a limit in refining the mesh and the user should verify the appearance of very fine meshes to be sure that there are no anomalies. Our experience with the refined meshes is that the nose tip bias shows a large space between the nose tip and the first ring of nodes. This is described later in the report and is recommended as a future

improvement. Then, the x-locations of each ring were mapped onto the range [0, 1] using Equation 4. A new x value is then calculated using Equation 5. This value is a scaling factor for the x-location in the coordinate system of the vehicle. Finally, the x-location in the coordinate system of the Mid-L/D model can be determined using Equation 6.

Equation 3: Calculating number of rings in nose.

$$n_{max} = \frac{1.5 * r}{edgeelem} \tag{3}$$

Equation 4: Mapping x on the range [0, 1]

$$x_i = \frac{i}{n_{max} - 1} \tag{4}$$

Equation 5: New x_i value.

$$x_i = \frac{e^{-kx_i} - 1}{e^{-k} - 1} \tag{5}$$

Equation 6: x-location of each nose grid point.

$$x = a + (b - a) * x_i \tag{6}$$

This final equation generates "n_{max}" values of x for the different ring locations in the nose. The x-locations are calculated starting at the nose tip and moving towards the end of the vehicle. The spacing between each ring decreases with decreasing distance to the nose tip.

The constants k, a, and b in the preceding equations were chosen using a trial and error method. First, a relationship was found for the k, a, and b values for one specific diameter. Then, k, a, and b values were found for different diameters. Through experimentation, it was found that k had the greatest effect on the spacing between rings on the nose section. A relationship was then developed that related k to the radius. The experimental data obtained was fit to a curve to find the best values for k, a, and b with changing dimensions, specifically diameter. The final equation had an r squared value of 0.9999, indicating that the equation was accurate for the range of diameters tested (2 to 20 m). The relationships describing each of the constants are as follows:

Equation 7: Nose bias constants.

$$\begin{aligned} k &= 0.3643 \log r - 0.0666 \\ a &= -1 * edgeelem \\ b &= 0.1r + 1.5 \end{aligned} \tag{7}$$

3.3.3.2.2 y and z-locations

First, an angle, φ is defined using Equation 8. This angle represents the angle between a vector formed from the origin to the x-location of the ring and a vector from the origin to the y_{max} location of the corresponding ring as shown in Figure 4. With this angle, the radius of a new ring can be determined using Equation 9.

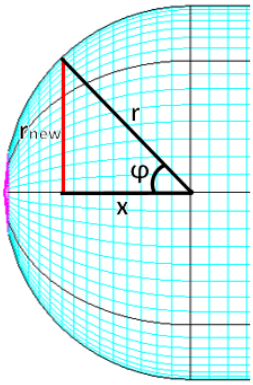

Figure 4: Definitions of angle, φ, and r_{new}.

Equation 8: Calculating angle, φ, of each ring.

$$\varphi = \cos^{-1}\left(\frac{|x|}{r}\right) \tag{8}$$

Equation 9: Calculating the radius, r_{new}, of the corresponding ring.

$$r_{new} = r\sin\varphi \tag{9}$$

The number of nodes, "ringelem", are equally spaced along a ring of radius r_{new}. This spacing is determined through a Python function, linspace. This function equally spaces a number of points, ringelem, on the range 0 to 2π as shown in Figure 5. This eliminates error found by manually calculating a dθ value by which to offset each node. The y and z values can now be calculated using Equation 10 and Equation 11, which use an angle of θ determined by the aforementioned Python function.

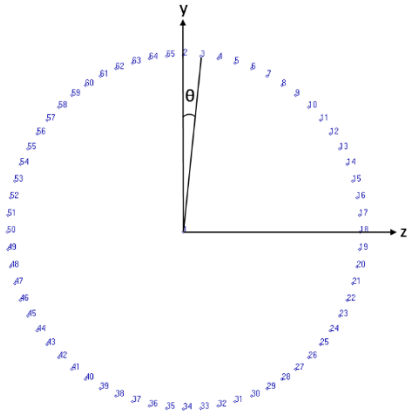

Figure 5: Cross section view of the first "ring" in the model.

Equation 10: y-location of grid point

$$y = r_{new} \cos\theta \tag{10}$$

Equation 11: z-location of grid point

$$z = r_{new} \sin\theta \tag{11}$$

3.3.3.2.3 Normal Vectors

Normal vectors for each grid point are simultaneously calculated in the code and stored in an array for later use when applying aerodynamic pressure loads to the body. The normal vectors at each node are calculated in Equation 12. The angles used in these equations are the same angles defined in Figure 4 and Figure 5.

Equation 12: Calculating normal vectors at each grid point.

$$\begin{aligned} nx &= -\cos\varphi \\ ny &= \cos\theta \sin\varphi \\ nz &= \sin\theta \sin\varphi \end{aligned} \tag{12}$$

3.3.3.3 Code Implementation

The grid points code begins by fixing the type of the variable that was imported. The dimensions and "edgeelems" are converted to floating point numbers. "Ringelem" is converted to an integer. The "length" and "edgeelems" variables are manipulated as seen in Table 2. Length, L, which is the total length of the vehicle, is used to calculate a length, l, of only the body of the vehicle. "Edgeelems" is the number of nodes along the length of the body. This is used to calculate the distance between nodes along the body, "edgeelem".

Next, two arrays for storing the grid points and normal vectors are initialized. A string, "text," is also initialized for use when writing information to a file. The grid points file is created and named with a unique name that was passed into the script as a variable, "filename".

The nose tip grid point is the first node to be specified as it is simply the negative radius of the vehicle, <-r, 0, 0>. The grid point array is stored in the nodes array and the normal vector array is stored in the normal array.

Now, the code begins determining the grid points for the rest of the nose. First, the angles, θ, as shown in Figure 5 are calculated with the Python linspace function. Using the nose bias function developed in 3.3.3.2.1, the code iterates through the number of rings, n_{max}, and determines the x-location of the ring. Once an x-location is determined, the value enters another loop that iterates through the angles, θ, to determine the corresponding y and z-locations for an entire ring of a number of nodes, "ringelem". In this same loop, the normal vector of each grid point is also determined. Once all the grid point locations and node ID's for a ring are specified, the x-location of the next ring is calculated and the process begins again.

There is a safety measure to be sure that an x-location greater than the radius is never stored. If this happens, the number of times this occurs is stored in the variable, "passes". This safety measure also insures that the user can input diameters outside the range that was specifically tested when determining the constants in Equation 7.

After determining the grid points for each ring in the nose, the final node ID is stored in the variable, "nodeID1". This is used as a starting number for the next set of grid points since each node must have a unique ID.

3.3.4 Body

3.3.4.1 Method

The grid points for the body of the vehicle are assigned following the final ring of the nose section. The first ring in the body occurs at a location of x=0 and the final ring occurs when x=l. The method for determining the x, y, and z values of each component is very similar to the method described for the nose. First, the x-location is found. Then, the y and z-locations of each grid point on a ring with a number of nodes, "ringelem," are found. The normal vectors are calculated simultaneously.

3.3.4.2 Equations

The body uses the same equations for the y and z-locations as used for the nose (Equation 10 and Equation 11). The normal vector y and z-components are also calculated the same as the nose (Equation 12). The x component of the normal vector is 0 because the plane of the ring is normal to the x-axis.

3.3.4.3 Code Implementation

The linspace function evenly distributes a number of nodes, "edgeelems", equally along the length of the barrel. The code iterates through each x-location specified. Then, the x-location enters the second loop, which iterates through angle, θ (see Figure 5). This loop calculates the y and z-locations as well as the y and z components of the normal vector. Unique node ID's are assigned to each grid point by adding one to the node ID after each iteration. The grid points and normal vectors arrays are stored in their subsequent arrays as done for the nose. The last node of the body of the vehicle is stored as "nodeID2".

3.3.5 Reference Nodes

Reference nodes are required when creating bar elements. The reference nodes are used to create a vector between a reference node and a grid point on the bar. This defines the orientation of each bar element. In the case of the Mid-L/D parametric model, axial symmetry can be utilized. Thus, the reference nodes are a set of nodes through the center of the cylindrical body. Each node has an x-value of the corresponding ring for which it is in the center. The y and z-locations are 0 for the reference nodes to be in the center. The reference nodes are created from the origin to the length of the cylindrical section.

3.3.5.1 Code Implementation

The code uses a for-statement and the linspace built-in Python function to iterate along the length, l, of the body with equally spaced "edgeelems" at specified x-locations. The x, y, and z-locations and unique node ID are stored in the nodes array.

3.3.6 Writing to a File

With the Grid Points file open, the text string can be appended to the document. The text string is edited so that it contains all the data previously stored in an array in the correct format for a Nastran BDF file. To do this, the code increments through each entry in a subarray, adds commas between entries, and stores it as a formatted subarray string. Keep in mind that each piece of information stored was a subarray and each subarray was stored inside another array. Also, note that element in this context refers to an item in a python array. It is used elsewhere in the paper to refer to the lowest discrete unit in FEA. The setup is pictured in Figure 6. Once all the array information is converted to a string and formatted properly, the text string is ready to be appended to the document.

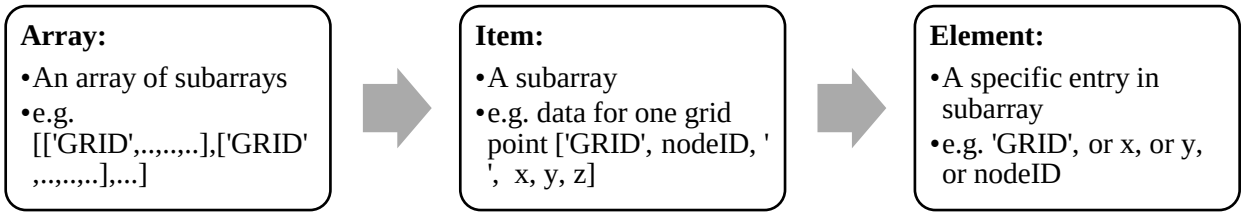

Figure 6: General format for storing data for writing to file.

3.3.6.1 Returned Information

There is important information determined in this script that is necessary for other functions to use. This important information is returned so that it can be called in another function. This information is defined in Table 5 below.

Table 5: Nodes function outputs.

Value	Type	Definition
Special Nodes	Array	List of the three important nodes in the model: 1. nodeID1 is the node ID number of the last node in the nose section. 2. nodeID2 is the node ID number of the last node in the body. 3. nodeID3 is the node ID number of the last reference node.
nodes	Array	Array of arrays. Contains the grid point location and node ID for each node.
normals	Array	Array of arrays. Contains the normal vector components of a normal vector at each node: [[node ID, x-normal, y-normal, z-normal],[node ID 2,...],…]
nmax	Integer	Number of rings in the nose section.
passes	Integer	When calculating the x-location of the nose rings, if the x-location is past the nose tip the value is neglected. This variable counts how many times this occurs.

3.4 Elements

The Elements.py script generates the mesh for the model using the existing grid points and a series of inputs (Table 6). Bar elements and composite shell elements are created in this file. In addition, bars are divided into groups and assigned their own properties. Shells are also divided into sections and provided a unique property.

Table 6: Elements function inputs.

Input	Type	Description
Dimensions	Array	Contains user inputs of model dimensions.
Mesh	Array	Contains user inputs for ringelem and edgeelems.
nmax	Integer	Number of rings in the nose section.
passes	Integer	Error count. See Table 5
filename	String	Title for the elements text file. It is called "elementsname" in the MidLOverD.py script.
SpecialNodes	Array	Lists important node ID's. See Table 5
Structure	Array	Contains user inputs for NumOfLongerons, NumOfSections, NumOfRings.
PayloadMass	Array	Contains user input for payload mass.
ShellMatl	Array	Contains shell information. See Table 2
LongMatl	Array	Contains longeron information. See Table 2
FrameMatl	Array	Contains frames information. See Table 2
TPS	Array	TPS density. See Table 2

3.4.1 Materials

In the baseline finite element models (FEM), the shell elements with a honeycomb sandwich are composed of the two materials shown in Table 7. The bar elements for the frames and longerons are assigned one material. Note that the user has the ability to specify different materials for each of these elements (e.g. a different material for the shells, frames, and longerons).

Table 7: Material Properties for honeycomb sandwich

Material	Material Notes	Density (kg/m^3)	Elastic Modulus (MPa)	Ultimate Allowable Stress (MPa)	Nastran Representation
IM7/977-3	Carbon fiber composite plates	1578.61	57950.00	514.75	MAT1 (MAT8 in the future)
5052 Aluminum Honeycomb	Honeycomb core with cell size of 3.175 mm	49.57	517.12	1.48	MAT1

The top and bottom plates of the honeycomb sandwich are currently carbon fiber composite plates of IM7/977-3. This material has a density of 1578.61 kg/m^3. The elastic modulus of IM7/977-3 is 57.95 GPa and the ultimate allowable stress is 514.75 MPa. In the current model, the composite is modeled as a quasi-isotropic material using a MAT1 card. Technically, only one layer of composite exists with this set up. In the future, it would be beneficial to specify the layering of the material. This would be done with a MAT8 card specifying the properties of the orthotropic material since one layer has different properties in each direction. Then, in the PCOMP card, this MAT8 card can be applied. At least four layers should be listed in this card at the 0, ±45, and 90 degree directions.

The core of the sandwich is currently 5052 aluminum honeycomb with a cell size of 0.125 in. (3.175 mm). The elastic modulus is 517.12 MPa and the ultimate allowable stress is 1483 kPa. Other core options that should be tested or considered for this model are 5052 aluminum honeycombs of 0.25 in. and 0.1875 in. cell sizes. When the baseline model was analyzed, it was determined that the honeycomb with a cell size of 0.125 inches had the lowest optimized mass. However, there may be different results for differently sized models. Again, this material can be modified in the "UserInputs" excel spreadsheet.

The excel spreadsheet, "UserInputs," calls for shell, longeron, and frame materials. The longerons and frames are also modeled as IM7/977-3. Originally, they were modeled as aluminum but a composite material significantly reduced mass. The shell section allows two materials to be stated: a core material and a composite material. The user must input a density, Young's Modulus, Shear Modulus, the stress allowable, and the upper and lower thickness limits. The bars take only one material. A MAT1 card models the bar materials which assumes the material is quasi-isotropic. Understanding this assumption, any material can be applied to the bars.

3.4.2 Properties

The baseline model divides the body of the vehicle into six shell sections, each of which have a different property. This allows each to be optimized separately in the design optimization for thickness. The user has the ability to alter the number of shell sections in the model. The shell properties are entered in the PCOMP card and treated as a composite.

The baseline model has six frame supports equally spaced along the length of the body of the Mid-L/D model. The first frame is assigned its own property since it sees only small loading. The last frame is also assigned its own property since it sees the highest loading in the launch case. The remaining frames are the payload carrying frames and they share another property. Again, the user has the ability to modify the number of frames. However, for the purpose of design optimization, the frames are always assigned properties in these groups in order to better optimize the vehicle for mass. The frame properties are assigned using the PBARL card, which also allows specification of a cross section.

The baseline model has eight longerons running the length of the vehicle from the nose tip to the end. These all share the same property as this model is a simplistic representation of the Mars Entry concept. The longeron properties are assigned in Nastran using the PBARL card. As of now, the longerons are all the same in order to reduce design complexity, machining costs, and avoid limiting the structure further. In the future, if a control system is in place to properly orient the vehicle for entry, then only a few longerons would carry the highest load. Modifying the model for this scenario would reduce the mass. Another modification that may be useful in the future would be to divide the longerons by shell section. Breaking each longeron into six or more pieces will allow for further reduction of mass. Currently, the entire longeron is optimized for the highest stress case, which has localized stresses at the very base of the vehicle.

Although this may be a powerful addition to the user inputs document, the current form of the code does not give the user the ability to change cross sections. Instead, this cross section is set to 'CHAN2' in the Elements code. This cross-section was selected initially as it was easier to account for in the design optimization and offered mass saving properties when compared to a box or rectangular cross section. This cross-section had fewer design variables and simpler

geometry constraints. The code would need to be much more robust to account for any other cross-sectional shapes and these constraints would need to be cross-section dependent.

3.4.3 Shell Elements

3.4.3.1 Triangle Nose Tip Elements

3.4.3.1.1 Theory

Since the nose converges to a single point, CQUAD shell elements cannot be used to mesh the grid points. Therefore, the first set of elements needs to be CTRIA elements, which only require three points to define the mesh surface. The CTRIA3 grid card is formatted according to the Nastran Quick Reference Guides [2].

3.4.3.1.2 Equations

The following figures show the relationships used to determine the node ID's for each node in a CTRIA3 element. The normal case is shown in Figure 7, while the exception is shown in Figure 8. Since the elements are defined clockwise about the origin, the last element must be treated differently. This is because the node numberings do not follow the same scheme. The first and last nodes are used to define the final element. Note that "tria" is a variable that is assigned a value through a loop.

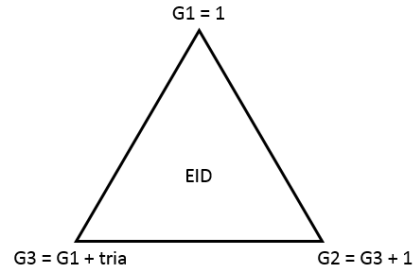

Figure 7: CTRIA3 Element definitions.

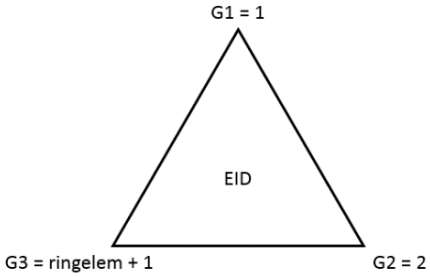

Figure 8: CTRIA3 Element definitions for exception.

3.4.3.1.3 Code Implementation

The code works by iterating through a range of node ID's from one to a number, "ringelem" plus one. The value of the iteration is assigned to the variable "tria". The built-in range function in Python does not include the final number; therefore, one is added to "ringelem". These values represent the first node and all the nodes in the first ring.

This for loop determines the three nodes that create a CTRIA3 element. One node is always the nose tip node, which has a node ID of 1. The second and third nodes are the nodes determined by the loop ("tria"), and the node directly after it.

There is a conditional if, else statement to account for the final node in the iteration. Because of how the nodes code is implemented, the first and last nodes will always be right beside each other. This pair of nodes is the only occurrence where the nodeID's do not have a difference of 1. Therefore, the "if" statement is included to specify the third node when the iteration gets to its last value, "ringelem", and does not follow the normal pattern.

3.4.3.2 Quad Body Elements

CQUAD4 elements are used to define the shell elements elsewhere on the model. This card specifies four points to develop rectangular elements over the surface of the body. The CQUAD4 grid card is formatted according to the Nastran Quick Reference Guides [2].

3.4.3.2.1 Theory

The remaining nodes are used to create CQUAD elements and assigns each a property based on its location on the body since the vehicle is separated into 6 shell sections (or a number specified by the user) with different properties or PID numbers. Each element is also supplied a unique element ID.

3.4.3.2.2 Equations

The grid points are specified in relation to G1, with G1 being the first node of the element assigned by the preceding for-loops. For every element but the last element in each ring, the grid points are calculated according to the relationships shown in Figure 9.

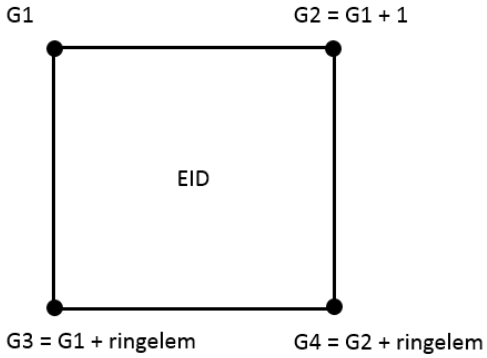

Figure 9: CQUAD4 Element definitions.

For the exception, the grid points are calculated according to Figure 10. Since the elements are defined in a clockwise manner, there comes a point where the first defined element

in the ring meets the last defined element. The element nodal definitions must be altered to account for this disruption in the numbering sequence.

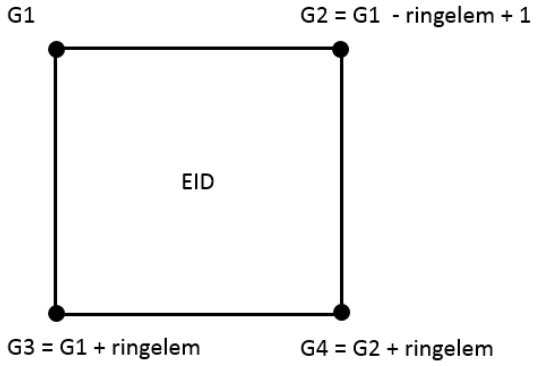

Figure 10: CQUAD4 Element definitions for exception.

The property ID numbers are calculated based on the number of rings of elements that have been created since each new iteration through the loop represents a new ring. A counter was used to determine this number. If this count is less than the number of rings in the nose, it assigns the element a PID of 30000. For the remaining sections, an equation was developed to determine the property ID.

Equation 13: Shell Section PID Assignment

$$PID = 30000 + int\left(math.ceil\left(\frac{count - nmax}{jump}\right)\right) \tag{13}$$

where,

$$jump = edgeelems/(NumOfSections - 1)$$

"Jump" represents the number of rings in each shell section. It is calculated by dividing edgeelems into the number of shell sections minus one. The reason the one is subtracted from the number of shell sections is to calculate the number of rings in the remaining shell sections. The dome section has already been accounted for, so there is one less section available into which to divide the barrel.

The "count" is a counter that keeps track of the number of iterations or rings. The count begins with the first ring in the nose section. Therefore, to determine the number of rings in the cylindrical section, the number of rings in the nose section, "nmax," must be subtracted from the total "count". This value is then divided by the constant, "jump," which returns a decimal value. This decimal value is the fraction of rings in a section. The value is rounded up using the Python function math.ceil. This assigns each fractional value less than 1 a value of 1 and adds it to the base number, 30000. For the baseline model, which has six sections, the PID numbers range from 30000 to 30005.

3.4.3.2.3 Code Implementation

The first loop in the code is a for-loop with the range function. It is used to determine the node ID of the first node on each ring. Each iteration adds on the number of nodes in a ring

(ringelem) to the previous node ID value. Upon determining the starting point, the code enters a second for-loop with a range function to determine each node ID in that corresponding ring. The else statement specifies the grid ID's for that element. The corresponding PID is determined based on its location in the model. Upon reaching the last node of the ring, the code enters the if-statement in order for the grid points to be assigned correctly. This entire process repeats until an element is created for each group of four nodes on the model.

3.4.4 Bar Elements

3.4.4.1 Longerons

3.4.4.1.1 Theory

The longerons are a necessary addition to the model as they carry significant loads, which allows for the shells to remain thin. The longerons are all assigned the same property but may be divided in the future as explained in Section 3.4.2.

The longerons were developed in two pieces. First, the longerons in the nose section were created, followed by the longerons in the barrel section of the model. To define the longerons, each CBAR element must have two node ID's for the attachment points and a reference node ID to orient the beam properly.

Because of the ordering of the nodes, it is simplest to develop the longerons by ring. Therefore, the element ID's of the bar elements increase radially rather than axially down the body of the aeroshell.

3.4.4.1.2 Nose Section

3.4.4.1.2.1 Relationships and Equations

One important relationship to note is the relationship between the CBAR elements in the nose section with the reference node. Because the dome is spherical, only one reference point is needed. This is the grid point located at the center of the sphere. This relationship is depicted in Figure 11 and Figure 12. All of the bars show a direction vector pointing to the center of the hemisphere. The two points needed to define the bar itself, are located on different rings. Therefore, the difference in the node ID's is the variable, "ringelem". Equation 14 is used to find GA and GB.

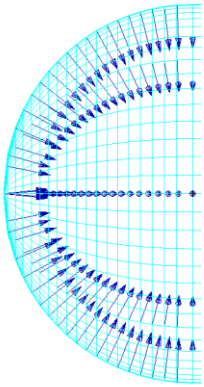

Figure 11: Side view of bar orientation vectors in nose.

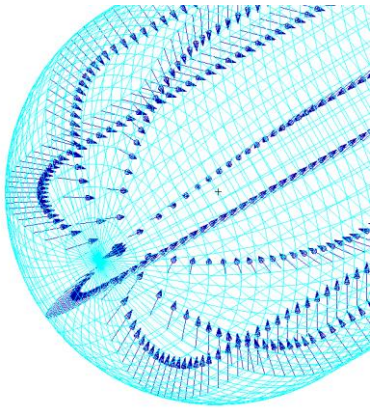

Figure 12: Isometric view of bar orientation vectors in nose.

Equation 14: CBAR nodes relationship.

$$GA = node + ring * ringelem$$
$$GB = node + (ring + 1) * ringelem \quad (14)$$

3.4.4.1.2.2 Code Implementation

This section of the code begins with a for-loop that determines the second node, GB, of each longeron in the nose section. The very first node, GA, does not need to be calculated. It is the same for all bar elements because the longerons all converge to the nose tip which has a node ID of 1. Thus, the second node that is calculated is also the first node present on a ring. The second node ID can range from two (the first node of the first ring) to "ringelem" (the last node of the first ring).

Then, the reference node for all the CBAR elements is defined. For the bars in the nose, this point also remains constant as described in 3.4.4.1.2.1. After creating the first CBAR element in a longeron, the code enters a nested for-loop. This loop iterates through the number of rings present in the nose section of the model and uses this to calculate the new GA and GB values of the next CBAR element (see Equation 14).

Each CBAR element is stored in an array that is appended to the elements array. Each element is assigned a property ID of 20000 and a unique element ID. Unique element ID's are acquired by adding onto the previous ID value.

3.4.4.1.3 Body Section

3.4.4.1.3.1 Relationships and Equations

The longerons in the barrel section of the model need a different set of reference nodes. As the CBAR elements are generated along the length of the barrel, the reference node must shift to be in line with the nodes. This is illustrated in Figure 13 and Figure 14. The relationship between GA and GB is the same as described in Section 3.4.4.1.2.2 and Equation 14.

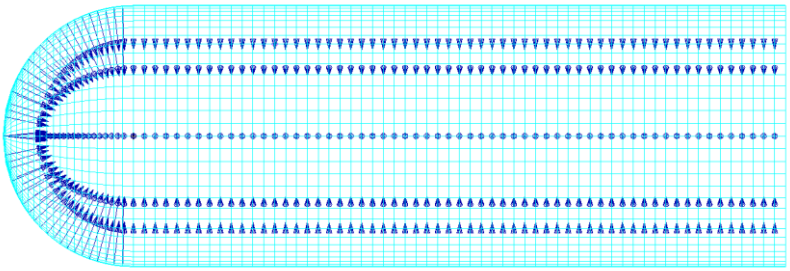

Figure 13: Side view of longeron orientation vectors.

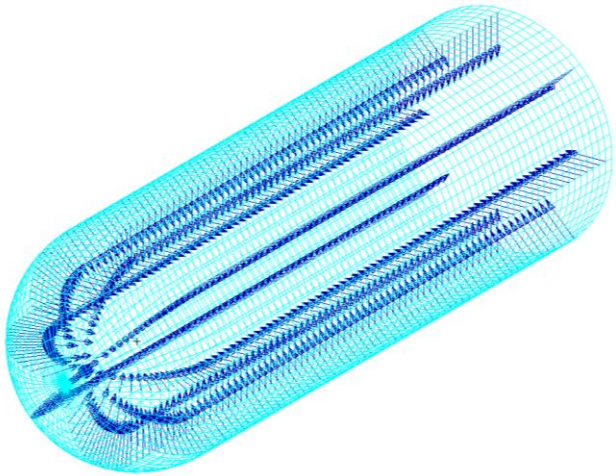

Figure 14: Isometric view of longeron orientation vectors.

3.4.4.1.3.2 Code Implementation

A for loop is used to iterate through the longerons in the model. For each longeron, a reference node in the center of the body is identified for proper orientation. The starting point for the first bar in each longeron comes from the ending point of the last bar in the longeron from the nose section. From here, the code enters another loop that iterates through each ring along the length of the barrel. GA and GB are assigned under this loop and the information necessary for the CBAR card is appended to the elements array.

3.4.4.2 Frame Rings

3.4.4.2.1 Theory

CBAR frames are a necessary addition to the model as they carry the majority of the loads, which allows the shell thicknesses to be minimized. They also provide the shells support to prevent buckling and the payload mass is directly connected to the frames. By assigning frames their own properties, the bars can be optimized for mass-saving dimensions.

3.4.4.2.2 Equations and Relationships

In order to develop CBAR rings implicitly through the code, a few relationships must be defined and noted. Recall the structure from Figure 1. The first ring is located at the forward end of the barrel and the last ring is located at the aft end of the barrel. One important relationship is the number of rings in the barrel section of the model and how to identify a specific ring. This relationship is shown in Equation 15. The variable, "ring," refers to the frame number.

Equation 15: Identifying rings in barrel.

$$smallring = \frac{edgelems}{NumOfRings - 1} * ring \tag{15}$$

This equation takes the number of elements along the length of the barrel and evenly divides it by the number of frames (rings) minus one. This defines the number of rings between each frame. This is done in this manner in order to reuse the grid points that have already been developed. Therefore, the user must be mindful of the values they choose for "edgeelems" and "NumOfRings". "Edgeelems" should be a number evenly divisible by "NumOfRings-1" as shown in Figure 15 in order to evenly space the frames.

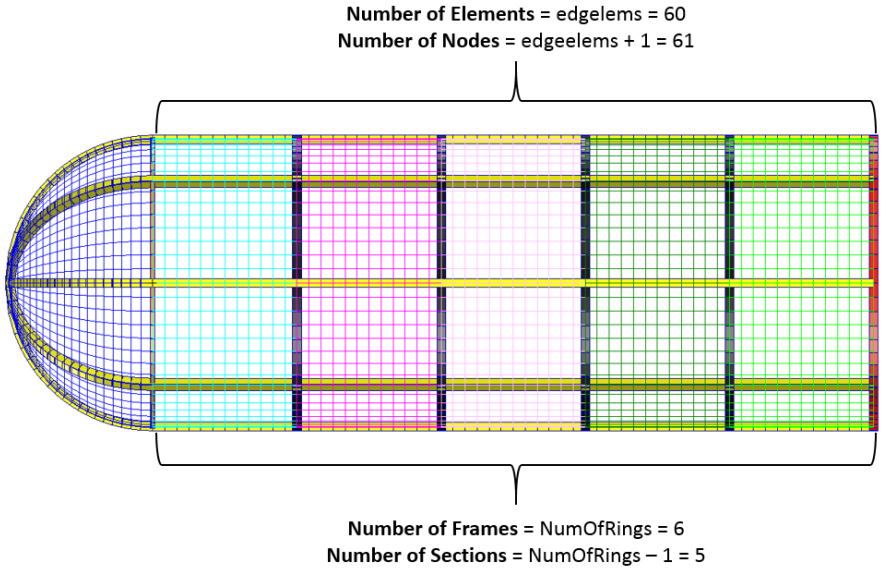

Figure 15: Creating evenly spaced frames.

Each frame must reference a new reference node in order to properly orient the bars in the frame (see Figure 16). To account for the changing reference node with location, Equation 16 was developed. G0 starts at the first reference node which is "nodeID2 + 1". This is the reference node for the first ring. Then, depending on Equation 13, a number is added to G0 so that it refers to the correct ring location. The integer value of "smallring" is taken because Nastran only takes integer inputs for node ID's.

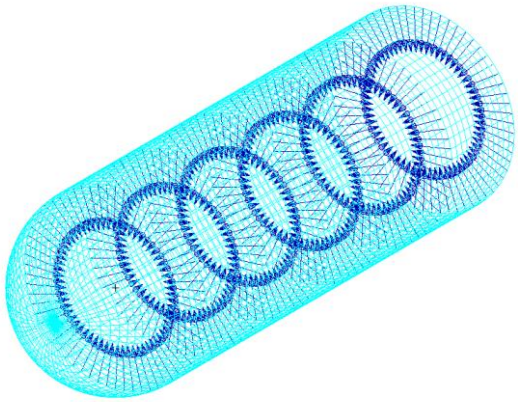

Figure 16: Ring orientation vectors.

Equation 16: Frame reference nodes.

$$G0 = nodeID2 + 1 + int(smallring) \tag{16}$$

The last important relationship is that needed to define the nodes of each bar element. Two nodes, GA and GB, are required to define the bar itself. The first node in each frame is found with Equation 17. This gives the first value of GA, which enters a loop to determine the rest of the values that will define the entire ring. The relationship between GA and GB is pictured in **Error! Reference source not found.** and Figure 18. GA is the first node and GB is the next node in sequence. This is true numerically for the entire frame, except for the very last bar element.

Equation 17: First node ID in a frame.

$$go = int((smallring * ringelem) + nodeID1 + 1) \tag{17}$$

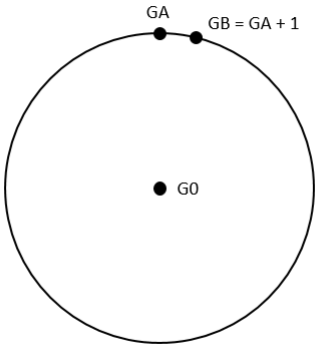

Figure 17: Frame node definitions.

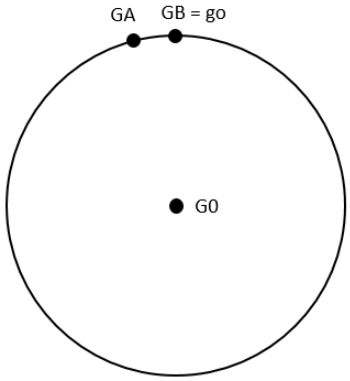

Figure 18: Frame node definitions exception.

3.4.4.2.3 Code Implementation

The script begins defining the CBAR frames with a for-loop that controls the number of frames to be created. This loop iterates through the number of frames specified by the user and passes the value to the variable, "ring". It starts at zero, which assigns the first frame a number of zero. This is also stored in the variable "firstring" and becomes useful when properties are assigned to each frame. For the baseline, the last value passed into "ring" through this loop is five, (NumOfRings – 1), and this represents the number of the last frame which is also stored in the variable "lastring".

The code then enters a second loop that defines the value for node GA. This loop iterates through the number of nodes in a ring, "ringelem". Nested underneath this loop is an if-else statement. When GA fits the condition shown in **Error! Reference source not found.**, it enters he if-statement to define CBAR. When GA fits the condition shown in Figure 18, it enters the else-statement to define CBAR.

Nested underneath the if-else statements is another set of if-else statements. These define the PBARL property ID's for each of the bar elements as described in 3.4.2. The first frame is assigned a PID of 22000. The last frame is assigned a PID of 22002. And, the remaining rings are assigned a PID of 22001.

3.4.5 Payload Mass

3.4.5.1 Representation

The payload mass for the baseline model is 100 Mt. The original setup of this model (Figure 19) treated the payload mass as a point mass (CONM2) and attached it to two frames with rigid body attachments (RBE3). This, however, presented problems due to high localized stresses that highly influenced the optimization of the design. In order to better distribute the mass, the point mass was divided into several point masses placed in the center of the corresponding load-carrying frames as seen in Figure 20. The number of masses into which it is divided is dependent on the user input for the number of frames. All, but the first and last, frames will carry these masses.

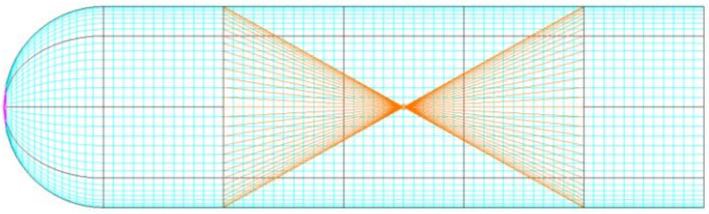

Figure 19: Previous model payload attachments.

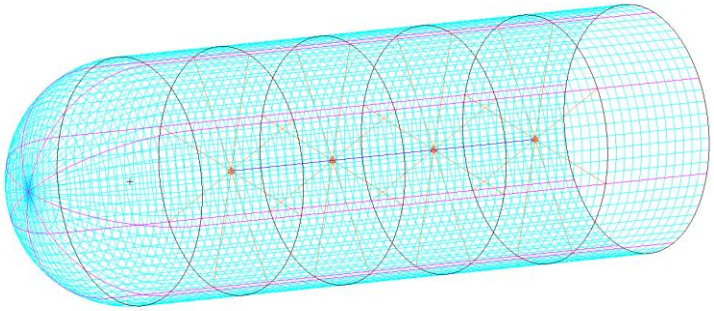

Figure 20: Current model payload attachments.

In order to make this a more realistic model, the CONM2 masses were attached to each other with a CROD. This insures that the masses move together by creating a stiff connection between each of the payload masses. The CROD card was assigned properties in the PROD card. It was given a small cross sectional area of approximately 4 square inches. The torsional constant was determined based on this cross section. The radius was calculated using the equation for the area of a circle and solved with the aforementioned cross sectional area. The radius was then substituted into the equation for the second moment of inertia of a uniform circular cross section and the result is the torsional constant.

3.4.5.2 Connection

The model in Figure 20 used RBE3's to attach the payload to each node on the ring. For the baseline model, this was 128 attachment points. Because of this unreasonable number, the number of attachments was reduced. There are still more attachment points than necessary in the current model, but these attachments can be revised as the design is improved.

For the current model, RBE3's are used to attach each CONM2 to the frame of which it lays in the center. These attachments are made at the junction of the frames and longerons. This distributes the load into the frames rather than the shells. Therefore, for the baseline model, there are 8 attachments for each CONM2 mass.

3.4.5.3 Code Implementation

First, the CONM2 cards are defined. This is done within a for-loop that iterates through the number of rings that will carry a payload mass. The total mass of the payload is divided by the number of rings that will carry it and this mass is written to a CONM2 card with the corresponding node location. Each card is appended to and stored in the elements array.

Next, the RBE3 cards are defined. The code locates the corresponding frame that will carry the load. A nested for loop is used to iterate through the nodes on that frame. The jump size is the number of nodes on a ring divided by the number of longerons. This jump size locates the specific node on the frame that intersects with a longeron. A variable, "cell," is used to properly format the Nastran card because each line in the BDF can have a maximum of eight values. Therefore a new line begins each time "cell" equals eight. There should be a total number of RBE3 cards equal to the number of point masses. Each is appended to the elements array.

Last, the CROD cards are defined. The code exits the first for loop and enters a new for loop that iterates through the number of masses. Based on the mass number, a starting and ending grid point location is selected from the CONM2loc array since the CROD card needs two grid points to be defined. Recall that this array contains the location of each CONM2 mass. This allows the CONM2 masses to be connected directly with this rod.

3.4.6 Writing to File

The elements array is modified much the same way as was described in Section 3.3.6 and Figure 6. Each item in each array in the elements array is concatenated in the correct format for a Nastran BDF file. The same is done for the materials and properties arrays, which contain the MAT1, PCOMP, and PBARL card entries for the shell honeycomb sandwich and bars.

3.4.6.1 Returned Values

The element identification numbers become very important when applying loads to the body. Therefore, these values are returned in an array so that other functions can use the information. Information about the shells, longerons, frames, and masses are important in the design optimization code. This information is also returned here for later use. The returned outputs of the elements function are listed and described in Table 8.

Table 8: Elements function outputs.

Value	Type	Definition
SpecialEIDs	Array	List of the five important nodes in the model: 1. EID1 is the element ID number of the last CTRIA3 element. 2. EID2 is the element ID number of the last CQUAD4 in the body. 3. EID3 is the element ID number of the last CBAR longeron element in the nose. 4. EID4 is the element ID number of the last CBAR longeron element in the body. 5. EID6 is the element ID number of the last CBAR ring element in the body.
Shells	Array	Array of arrays. Contains the shell information necessary for writing the design optimization file. Each element represents the data for one shell property and is formatted in this manner: [PCOMP,PID,Laminate Initial Thickness, Honeycomb Initial Thickness]
Longerons	List	Contains the longeron information necessary for writing the design optimization file. It is a list stating the Nastran card (PBARL), PID, and the initial dimensions for the cross-section of the beam (dim1, dim2, dim3, dim4).
Frames	Array	Array of arrays. Contains the frame information necessary for writing the design optimization file. Each element represents the data for one bar property and is formatted in this manner: [PBARL, PID, frame dim 1, frame dim 2, frame dim 3, frame dim 4].

CONM2loc	Array	Lists the point mass locations (grid point ID's) for each CONM2 in the model.

3.5 Loads and Boundary Conditions

There are several cases for which the model must be tested. From there, the critical load cases can be determined. All the loads are defined in LoadsandBCs.py. This code generates the loads and boundary conditions of the Mars Entry model and the generated information is written to a text file to be read by Nastran.

3.5.1 Hypersonic Theory

The modified Newtonian sine-squared law (Equation 18) can be used to approximate the hypersonic aerodynamics of the vehicle during entry [3].

Equation 18: Modified Newtonian Sine-Squared Law.

$$C_p = C_{p,max}\ sin^2(\theta) \tag{18}$$

This theory assumes that the air particles hit a surface and then bounce off tangential to the surface [3]. The angle at which a particle strikes the surface and the tangential of the surface is θ. In order to calculate the pressure at each node of the model, the unit normal of each point must be calculated. Then, the velocity vector is dotted with the normal vector and the angle, θ, is solved for with the following relationship in Equation 19.

Equation 19: Dot product relationship.

$$\frac{V_\infty \cdot n}{|V_\infty|} = cos\,\varphi = sin\left(\pi/_2 - \varphi\right) = \ sin\,\theta \tag{19}$$

For an angle greater than or equal to 90 degrees, there will be no resulting pressure load. Therefore, it can be assumed that nodes in this "shadow" have no pressure load applied. This is depicted in Figure 21. In this model, $C_{p,max}$ can be approximated as 2.0. The previous equations are manipulated to get it in the proper form for the code, which directly solves for a pressure differential as shown in Equation 20. In this equation, q is the dynamic pressure.

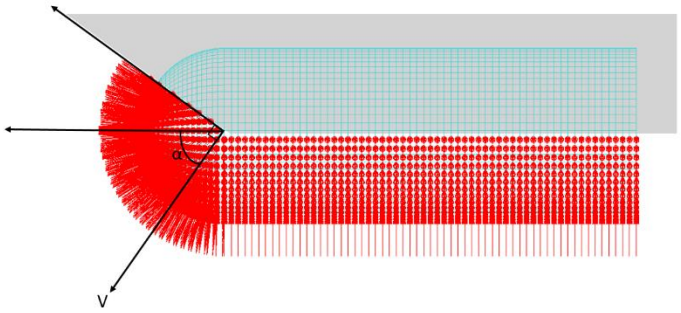

Figure 21: "Shadow" assumed in hypersonic theory.

Equation 20: Pressure equation.

$$\Delta P = 2q\,cos^2\,\varphi \tag{20}$$

With this model for pressure, the pressure loading can be defined for entry and approximated for launch. To approximate the aerodynamics at lift off, the maximum dynamic pressure is scaled down 90%.

The normal vectors are calculated within the Grid Points code and the data are saved to an array, "normals", that is passed to the Loads and Boundary conditions code for use.

3.5.1.1 Code Implementation

There is a pressure function within the LoadsandBCs.py script. The function is called "PressureLoad". It takes eight inputs (see Table 9) to develop a pressure distribution on the aeroshell according to the modified Newtonian method explored in Section 3.5.1.

Table 9: Pressure function inputs.

Input	Definition
Ringelem	Number of nodes in a ring.
nodeID2	Node ID of the last barrel section node.
SID	Set Identification Number of the load case.
q	Dynamic Pressure value obtained from UserInputs.csv
alpha	Angle of attack/entry obtained from UserInputs.csv
beta	Angle of sideslip obtained from UserInputs.csv
normals	Array of the normals for each node on the model. See Table 5.

The script begins by initializing the array "Pressures" to store the pressure loading data. It then converts the alpha and beta inputs from degrees to radians. Next, the velocity vector is determined based on these angles. The code then enters a loop that iterates through each array in the normals array. Taking the information from these arrays, a normal vector at each node is generated. This normal vector is dotted with the velocity vector to determine an angle, "theta", between the two vectors. This angle is used to decide which nodes lie in the shadow depicted in Figure 21.

If a value lies in the shadow, it enters the if-statement and is assigned a pressure magnitude of zero. The if-statement contradicts the conclusions of hypersonic theory. This theory develops that angles greater than or equal to 90 degrees have zero pressure. However, the code finds angles less than or equal to 90 and assigns those a magnitude of zero. This is because of the direction (sign) of the vectors in the model. Finding angles greater than or equal to 90 degrees would actually return the wrong (opposite) nodes.

For the nodes that are not in this shadow, the code enters the else statement and a pressure value is calculated using Equation 20. The magnitude of the pressure vector and the node at which it occurs is stored in the Pressures array.

With the pressures at each node being known, values can be entered into the PLOAD4 Nastran card to determine the pressures on each element. The PLOAD4 card needs a load set ID, element ID, and the pressures at each node defining the corresponding element. The code generates these PLOAD4 cards in two steps.

The first step assigns the pressure loads to the CTRIA3 shell elements. Recall that only three nodes define these elements. Thus, only three pressure values corresponding to these three

nodes must be found. A for-loop iterates through the first ring of nodes in the model. Recognizing that the triangular elements share one point, the nose tip, P1 can be assigned a constant value. The pressure values are assigned by calling an element in a subarray of the Pressures array. For example, Pressures[0][1] calls the second element in the first subarray. The subarray refers to the pressure data for a specific node. In this case, the first subarray has information for node ID 1. The element refers to the piece of information in that subarray. The first element in each subarray is the node ID number and the second element is the pressure magnitude. The subarray numbers are found through a relationship based on the one developed in Figure 7 and Figure 8. The second and third nodes of the CTRIA3 elements are always right beside each other. Because of this, there is the exception case where the last node of a ring is right beside the first node. When this happens, the code satisfies the if-statement and assigns a different P2. For the normal cases, the code enters the else-statement to assign P2. The PLOAD4 cards are appended to an array titled "loads".

The second step assigns the pressure loads to the CQUAD4 shell elements. These elements are defined by four nodes. Again, an if-else statement is employed to account for the cases explained in the first step. The pressure values are again defined by calling an element in a subarray of the Pressures array. The subarray is determined using the same nodal relationship depicted in Figure 9 and **Error! Reference source not found.**. Then, the PLOAD4 cards are ppended to the loads array.

3.5.2 Launch Case

3.5.2.1 Boundary Conditions

The aeroshell structure would fly on the top of a rocket and essentially replace the payload bay. Realistically, there will be only a few points of attachment to the launch vehicle itself. To properly model launch, the bottom end of the vehicle is constrained from translating and rotating in all directions. This corresponds to the 1, 2, 3, 4, 5, and 6 directions in Nastran. This constraint is directly applied to the nodes on the last ring that coincide with the longerons.

In addition, the reference nodes running through the center of the model must also be constrained with the exception of the grid points referring to the CONM2 point masses. Constraining the reference nodes in all directions prevents singularities in the model from causing errors in the analysis. The vehicle constraints are depicted in Figure 22.

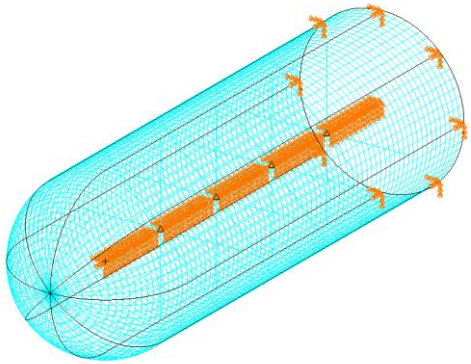

Figure 22: Launch case boundary conditions.

3.5.2.2 G-Loads.

The G loads for launch can be entered into a spreadsheet, following the format depicted in Figure 23. This document should be saved as a comma separated file (.csv). The filepath is a user input and this tells Python were to read the G loads from.

	A	B	C
1	Lateral	Axial	
2	0.5	6	
3	0.5	4.5	
4	2	2.3	
5	2	0	
6	1.5	-0.2	
7	1	-1	
8	0.5	-1	
9	0.5	-2	
10			
11			
12			

Figure 23: G Loads Template.

G loads such as the ones listed can be obtained from manuals and handbooks specific to the launch vehicle. Initially, this table of values was used. These numbers were taken from the Delta IV Heavy manual. However, these loads are high overestimations of the loads that the aeroshell will actually experience on the Space Launch System (SLS), the predicted launch vehicle. These loading conditions made it too difficult to design and optimize the Mid L/D structure and thus, were not used in performing analysis on the final model.

The final model takes a separate excel spreadsheet that was created for the SLS launch loads. Having only one line of data in the spreadsheet, the Python code only generates one launch case. The baseline model takes the only SLS estimates currently available, which are an axial load of 4.0 g's and a lateral load of 0.75 g's. The SLS launch loads case is shown in Figure 24.

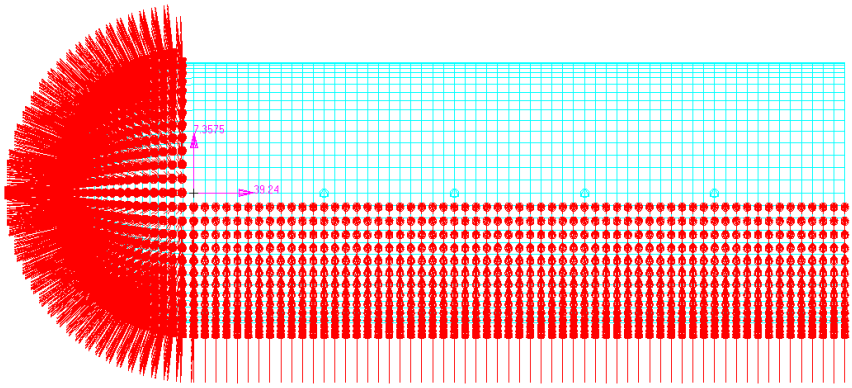

Figure 24: Launch Loads.

3.5.2.3 Aerodynamic Pressure Loads

It is necessary to include aerodynamic loading because the aeroshell will act as a fairing and be exposed to the atmosphere. The aeroshell structure will not be housed inside of a payload bay. The theory described in Section 3.5.1 is used to develop the aerodynamic pressure loading on the vehicle at launch. Alpha is deemed to be a very small value (0 – 10 degrees). However, at launch, this load is not very high in magnitude as the vehicle is still at low speeds. Thus, hypersonic theory is only used to distribute the loading over the aeroshell and the magnitude is approximated by scaling the maximum dynamic pressure down ninety percent. This scaling is built into the code. If the need arises to change this factor, it should be changed within the script. This factor would not accurately model the aerodynamic loads throughout the entire flight of the vehicle. Although this factor is built in, the dynamic pressure is not. Dynamic pressure remains a user input in the UserInputs.csv document. The pressure loading is depicted in Figure 24 and Figure 25.

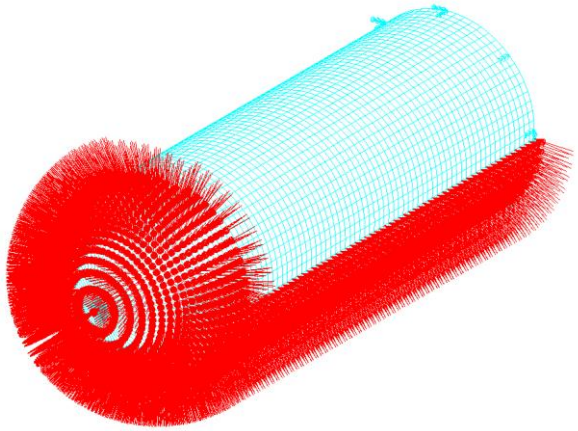

Figure 25: Isometric view of launch loads.

3.5.3 Entry Case

3.5.3.1 Inertial Relief

For this load case, boundary conditions could not be applied to the model since it is a free-falling body. Therefore, inertial relief was used to achieve better results without constraining a free end of the entry vehicle. Nastran uses inertial relief to constrain an unconstrained body. This is typically used for aircraft, satellites, and other air or space vehicles with no physical constraints. This free-free structure is used in the static analysis.

3.5.3.2 Entry Loads

The only loads applied to the entry case are aerodynamic loads as shown in Figure 26. This loading profile was developed using hypersonic theory as detailed in Section 3.5.1. The user has the ability to input values for alpha, beta, and dynamic pressure into the UserInputs.csv spreadsheet. The baseline model uses a dynamic pressure of 5000 Pa and an entry angle of 55 degrees.

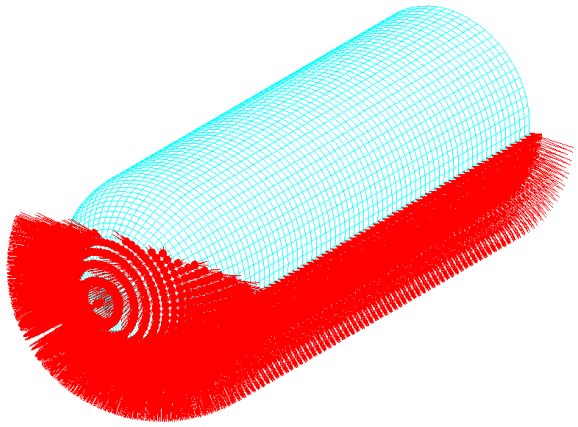

Figure 26: Entry Loads.

3.5.4 Code Implementation

The LoadsandBCs.py Python script is used to generate the loads and boundary conditions for the Entry and Launch cases. As outlined in Section 3.5.1.1, the first part of the script defines a function called "PressureLoad" used to develop the aerodynamic loading.

The second part of the script defines the "LoadsandBCs" function. This function takes nine inputs, which are listed and described in Table 10.

Table 10: LoadsandBCs function inputs.

Input	Type	Definition
Mesh	Array	Contains user inputs for ringelem and edgeelems.
SpecialNodes	Array	Lists important node ID's (see Table 5).
Structure	Array	Contains user inputs for NumOfLongerons, NumOfSections, NumOfRings.
PayloadMass	Array	Contains user input for payload mass.
Press	Array	Contains user inputs for dynamic pressure, alpha, and beta for the launch and entry cases.
normals	Array	Lists normals for each node (see Table 5).
filename	String	Title for the loads and boundary conditions text file. It is called "loadsname" in the MidLOverD.py script.
LaunchLoadsFile	Array	User input for the launch loads file location.
CONM2loc	Array	Stores the locations of the CONM2 point masses.

The code first takes the values it needs from the array inputs and then corrects them to the proper form. A file with the name, "filename," is created for writing the loads and boundary condition cards. Following the same format as the previous scripts, an array titled "loads" is initialized for storing the loads and boundary condition cards and data. A text string is also initialized and will be used in writing the text to a file as explained in Section 3.3.6.

The script then sets the boundary conditions for the entry and launch cases. For entry, there are no single point constraints (SPC) as it is an inertial relief run and only two cards are defined.

For launch, a series of boundary conditions must be applied as were explained previously and shown in Figure 22. A for-loop is used to determine the node ID's that are constrained on the last ring. The loop ranges from the first node of the last ring to the last node of the last ring. The

jump size is calculated by dividing the number of nodes in a ring by the number of longerons. A list of these nodes is then appended to the SPC1 card entry and the SID is saved to the SIDs array. Next, the reference nodes are constrained. This is done with a for-loop, and a nested if-else-statement. The first loop iterates through a number equal to the length of the CONM2loc array. This number refers to a frame. Depending on the number of the frame, the code defines the beginning and end nodes to be listed for a SPC constraint. The frames with a CONM2 mass in the center are routed through the else-statement. These frames define beginning and end nodes that do not include the reference node directly in the center of that ring. This is because this node is also used to define a CONM2 mass. The if- and else-if-statements define the beginning and end nodes for the first and last rings, which do not contain a CONM2 mass. Each list of nodes is written to a SPC1 card with a unique SID. They are also appended to the SIDs array.

Because there are multiple constraint sets for the launch case, an SPCADD card must be used to combine all the constraints into one set. This is done by looping through the SIDs array and listing all the launch SID's in an SPCADD card. The SID of this card is then saved for later use in the case control section.

After determining the boundary conditions, the pressures are found for the entry and launch cases. The elements in the "Press" array correspond to the dynamic pressure, alpha, and beta for each case. These strings are taken, stored in the correct variable, and converted to floats. The pressure function is then called with all eight inputs and the PLOAD4 cards for both the entry and launch cases are defined.

Next, g loads are assigned to the launch case. Recall that the entry case is an inertial relief run and so it does not take any g loads. The launch g loads file is opened to be read line by line. The first line is skipped since it is the title line and does not contain any data. The remaining lines in the file are separated by a comma and stored in an array titled "LaunchGs". The elements of this array represent the lateral and axial g loads. The elements are converted from strings to floating point numbers and then stored in the variables, "lateral" and "axial". A GRAV card is created for each of the lateral and axial loads and each has a unique SID.

Finally, the LOAD cards are defined in order to allow combinations of load sets to appear in each case. The entry case only lists the pressure load SID to this card. The ID of the LOAD card is saved to the "EntryLoadCards" array. This array will be passed to case control so that it can call the proper load set for the entry case.

For launch, the SID's for the pressure, lateral g load, and axial g load are listed to the LOAD card. Then, the unique ID of the LOAD card is saved to the LaunchLoadCards array. Storing all the IDs in this array allows the user to create multiple launch subcases.

The loads array undergoes the same write to file method discussed previously. Each element in each subarray in the array is concatenated and stored in the text string.

3.5.4.1 Returned Information

The loads and boundary conditions script returns valuable information for the case control script. These outputs are listed and described in Table 11.

Table 11: LoadsandBCs function outputs.

Input	Type	Definition
SID_SPCADD	Array	SID corresponding to the launch case boundary conditions.
LaunchLoadCards	Array	Lists the SID's of the Launch Cases.

EntryLoadCards	Array	Lists the SID's of the Entry Cases.

4 Analysis (Nastran Solutions)

Three Nastran solutions are used for analysis of the Mid L/D structure. Thus, three different BDF's are developed through the code. A linear static analysis can be performed with Solution 101. A vibrations analysis can be performed with Solution 111. And, most importantly, a design optimization can be performed with Solution 200. Here lies the true power of the parametric model. Having the nodes, elements, loads, and boundary conditions being generated through a code that takes user input values for several variables gives the user the ability to optimize several designs in a short amount of time.

The Case Control script is responsible for writing the executive control section of the BDF files. It organizes the required information for each subcase in the proper format for a Nastran input file. There is a separate case control script for each of the different solutions (SOL100, SOL111, and SOL200).

For SOL111 and SOL200, there is also the need to add more cards to the bulk data section. Therefore, scripts were written that generate the proper BDF for these solutions.

4.1 Linear Static Analysis (SOL 101)

4.1.1 Case Control

There are two main subcases in this case control section. One subcase represents launch loads whereas the other represents entry loads. For all of the subcases, a unique subcase number and title are created. The boundary conditions are specified by SPC ID and the loads are specified by the SID number. The next part of the case control section states the kind of data that is needed from the analysis. This part remains the same for all the subcases and assures that the analysis reports Von Mises stresses, forces, and displacements. Finally, the text is written to the file.

There can be several launch load cases due to the G loads varying during launch, therefore the code iterates through each case to assign each a unique number and title. For all the launch cases, the same displacement boundary conditions apply. The base of the vehicle is restricted from translations in the x, y, and z-directions as well as rotations about the x, y, and z-axes. Each load case also has the same pressure load applied. The only detail that varies between the launch load cases is the magnitude of the G loads in the axial and lateral directions and this is managed by altering the LOAD card number to represent the loading on the vehicle.

There is only one entry case and it only contains aerodynamic loads. This subcase differs from the launch case in that it uses inertial relief rather than boundary conditions to constrain the vehicle. It was determined after test simulations were performed that inertial loading showed more accurate results. Thus, this section does not have an SPC card. Instead, a "PARAM,INREL,-2" card is used to constrain the vehicle for analysis with inertial relief. As with the launch loads, the entry loads are specified with a LOAD card referring to the proper SID.

4.2 Modal Frequency Response (SOL 111)

In this analysis, the dynamic response of the vehicle under vibro-acoustic excitations is examined. These excitations vary with launch vehicle. Because the SLS data are not available, the data for the Delta IV Heavy rocket were used in order to provide a crude estimation of the structures ability to withstand the launch environment. The results of this analysis, however, did consistently show that the vibro-acoustics were not the critical load case and had negligible effect on the model. Thus, this analysis was not performed for each design in the parametric study in order to reduce computation time. This solution can be neglected from all further studies because of its negligible effect.

4.2.1 Case Control

The Vibro-Acoustics case control section takes only three inputs to set up the analysis. As with the static case control section, it states what analysis is to be performed and how, and what information is output. The inputs are listed and described in Table 12.

Table 12: Vibro-Acoustic case control inputs

Input	Type	Definition
CaseNum	Integer	Case number.
SpecialSIDs	Array	All the set ID's pertaining to important parameters developed in the VA_BulkData.py script.
filename	String	Title for the Vibro-Acoustic case control text file. The variable is called "VACCname" in the MidLOverD.py script.

Only one test element is selected because the analysis would take much longer to run if all the elements were selected. Analysis can be re-run to test different elements at different locations on the body of the vehicle by changing the test element number in the script itself.

The case control script defines how the results are output in a graphical format and the details of the plot are stored in the Plotting array. The power spectral densities are plotted on the .f06 file to provide a quick overview of the vehicles response to the vibro-acoustic environment.

The case control section also determines important information regarding the eigenvalue extraction method and random analysis. At the end, the information is written to the case control text file. There are no returned values in this script.

4.2.2 Bulk Data

The bulk data script is titled VA_BulkData.py. It takes in several inputs as explained in Table 13. The code begins by initializing all the arrays and storing all the appropriate variables. Then, it constrains all the reference nodes through the center of the vehicle, except those referring to the locations of the point masses, and appends this information to the file. The rest of the code defines various elements relating to the frequency analysis such as the range of frequency values in interest, damping values, power spectral density factors, loads and loading conditions, and more. All the necessary information is written to a text file and important values are returned for use in the case control section of the Vibro-Acoustics BDF. The outputs are listed in Table 14.

Table 13: Vibro-Acoustics bulk data script inputs.

Input	Type	Definition
CaseNum	Integer	Case number.
Mesh	Array	Contains user inputs for ringelem and edgeelems.
SpecialNodes	Array	Lists important node ID's (see Table 5).
SpecialEIDs	Array	List of the important nodes in the model.
filename	String	Title for the Vibro-Acoustic bulk data text file. The variable is called "VABDname" in the MidLOverD.py script.
LaunchFiles	Array	Contains filepaths for launch files. For this script, it specifically uses the second element in the array which contains the filepath for the vibro-acoustics data file.
CONM2loc	Array	Nodes locations of the payload point masses.
Structure	Array	User inputs relating to the structure of the vehicle.

Table 14: Vibro-Acoustics bulk data outputs.

Input	Type	Description
SpecialSIDs	List	Important set ID's for use in the Case Control script: TID_TABDMP1, SID_EIGRL, SID_RANDPS, SID_FREQs, SID_SPCADD, SID_DLOAD

4.3 Design Optimization (SOL 200)

This analysis is the most important for the parametric study. The previous solutions were developed so that the user has the ability to check the results and examine the load cases more closely. However, the real power of this code is in the optimization. This Nastran solution takes a model and the loading conditions, and optimizes the structure for mass. It outputs material thicknesses and masses for the beams and shell elements. These numerical results can offer direct comparison of models. It also gives the user powerful insight as to how the model responds overall. The model is optimized for statics, buckling, and modes. The important Nastran cards used in developing the optimization run are listed in Table 15.

4.3.1 Nastran Cards

Table 15: Design optimization cards.

Card	Definition	Necessary Information
DCONSTR	Defines design constraints.	Lower and upper bounds.
DEQTN	Define an equation used to relate variables in design sensitivity analysis.	Equation in proper format.
DESVAR	Defines a design variable for optimization.	A name, initial value, and lower and upper bounds.
DLINK	Relates one design variable to another.	Constants and design variable ID's.
DRESP1	Defines a set of structural responses that can either be used as a constraint or objective.	Type of response, property name, and attribute.
DRESP2	Defines equations that are used as constraints or objectives.	Unique ID and equation ID.
DVPREL1	Relates a model property to a design variable.	Name of property, PID, property entry name, and design variable ID.
EIGRL	Defines data needed to perform real eigenvalue analysis.	Number of roots desired.
DOPTPRM	Defines the parameters used in the design optimization.	Name of parameter and value.

4.3.2 Case Control

The Case Control code takes in number of inputs, most of them being ID numbers for various aspects of the load case. These inputs are listed in Table 16.

Table 16: Design optimization case control inputs.

Input	Type	Description
CaseNum	Integer	Case number.
filename	String	Title for the Design Optimization case control text file. The variable is called "DOCCname" in the MidLOverD.py script.
SPC	Array	SID corresponding to the launch case boundary conditions.
LaunchCard	Array	Lists the SID's of the Launch Cases. See Table 11.
EntryCard	Array	Lists the SID's of the Entry Cases. See Table 11.
EIGRL	Integer	SID of the EIGRL card defining the eigenvalue analysis.
Target	Integer	SID of DRESP1 card specifying the response to be optimized (in this case, weight).
DCIDS	Integer	SID for the linear static analysis.
DCIDB	Integer	SID for the buckling analysis.
DCIDM	Integer	SID for the normal modes analysis.

The case control script begins by initializing two arrays for storing the case control commands and subcase information. Then, a file titled "DO_CaseControl_#.dat" is created and the inputs are all converted to strings.

A series of case control commands are then written to the case control array. “DESOBJ(MIN) =” is used to specify which parameter is to be minimized. In this case, weight is the parameter to be minimized, so this command is set equal to the ID of the weight design variable. “ELSUM(PIDSUM) = ALL” tells Nastran to printout a summary of the masses for each of the PID groupings. This is a useful command as it provides a mass breakdown for each property at each cycle of the design optimization. A sample of this output is pictured in Figure 27.

E L E M E N T P R O P E R T Y S U M M A R Y (BY PROPERTY TYPE / ID)

				STRUCT.MASS	NON-STR.MASS	TOTAL MASS	TM*WTMASS
SUBTOTAL MASS FOR ALL BAR	ELEMENTS	FOR PBARL, ID =	20000	5.2036E+02	0.0000E+00	5.2036E+02	5.2036E+02
SUBTOTAL MASS FOR ALL BAR	ELEMENTS	FOR PBARL, ID =	22000	7.5641E+01	0.0000E+00	7.5641E+01	7.5641E+01
SUBTOTAL MASS FOR ALL BAR	ELEMENTS	FOR PBARL, ID =	22001	1.6666E+03	0.0000E+00	1.6666E+03	1.6666E+03
SUBTOTAL MASS FOR ALL BAR	ELEMENTS	FOR PBARL, ID =	22002	8.3861E+02	0.0000E+00	8.3861E+02	8.3861E+02
SUBTOTAL MASS FOR ALL BAR	ELEMENTS			3.1012E+03	0.0000E+00	3.1012E+03	3.1012E+03
SUBTOTAL MASS FOR ALL CONM2	ELEMENTS			1.0000E+05	0.0000E+00	1.0000E+05	1.0000E+05
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS	FOR PCOMP, ID =	30000	6.5105E+02	1.5369E+03	2.1879E+03	2.1879E+03
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS	FOR PCOMP, ID =	30001	6.5185E+02	1.5388E+03	2.1906E+03	2.1906E+03
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS	FOR PCOMP, ID =	30002	2.4463E+03	1.5388E+03	3.9850E+03	3.9850E+03
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS	FOR PCOMP, ID =	30003	4.6298E+03	1.5388E+03	6.1685E+03	6.1685E+03
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS	FOR PCOMP, ID =	30004	7.0243E+03	1.5388E+03	8.5631E+03	8.5631E+03
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS	FOR PCOMP, ID =	30005	1.0661E+04	1.5388E+03	1.2200E+04	1.2200E+04
SUBTOTAL MASS FOR ALL SHELL	ELEMENTS			2.6064E+04	9.2307E+03	3.5295E+04	3.5295E+04
SUBTOTAL MASS FOR ALL QUAD4	ELEMENTS			2.6051E+04	9.1982E+03	3.5249E+04	3.5249E+04
SUBTOTAL MASS FOR ALL TRIA3	ELEMENTS			1.3740E+01	3.2435E+01	4.6175E+01	4.6175E+01
SUBTOTAL MASS FOR ALL ROD	ELEMENTS	FOR PROD, ID =	10000	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
SUBTOTAL MASS FOR ALL ROD	ELEMENTS			0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
TOTAL MASS FOR ALL SUPPORTED ELEMENT TYPES				1.2917E+05	9.2307E+03	1.3840E+05	1.3840E+05

Figure 27: Mass summary in .f06

The static entry case is setup in the code, however, it is commented out because it is not used in the design optimization. This is because the launch case was the driver and the entry case can be optimized to a mass much lower than that for launch [4].

The launch static subcase is defined for analysis. It is assigned a subcase number and a subtitle for easier user readability. The analysis type is statics. The constraints are specified with the “SPC=” card which takes a set ID number. The loads are specified in a similar manner with the “LOAD=” card being set equal to the ID of the launch load set. The next three commands assure that displacement, force, and stress information is reported for all points and elements. The “DESSUB” command is set equal to the design constraint ID for the buckling analysis. This sets the design constraints for the analysis.

The buckling subcase is defined in the same manner. It is given a unique subcase number and a different subtitle. For this case, the analysis type is buckling (“BUCK”) and only

displacement information is requested. Last, the "DESSUB" command is set equal to the design constraint ID for the modal analysis.

Finally, as with all the previous scripts, all the arrays are converted to strings and printed to the file in the correct Nastran format. This script does not return any values or information.

4.3.3 Bulk Data

The bulk data script develops all the design variables and constraints for the design optimization analysis. The inputs for this file are listed in Table 17.

Table 17: Design optimization bulk data inputs.

Input	Type	Description
Shells	Array	Important information relating to shells such as PID, initial thickness, limits, etc (see Table 8).
ShellMatl	Array	User inputs for shell materials.
Longerons	Array	Important information relating to longerons such as PID, initial dimensions, limits, etc (see Table 8).
LongMatl	Array	User input for longeron material.
Frames	Array	User input for frame material.
Frame Matl	Array	Important information relating to frames such as PID, initial dimensions, limits, etc (see Table 8).
filename	String	Filename for bulk data file "DO_BulkData_#.dat".
CaseNum	Integer	Case number. Used to include the correct nodes, elements, and loads files.

The script begins by creating a file for storing the design optimization bulk data and it is titled "DO_BulkData_#.dat". Then, a series of arrays are initialized for storing design variables, design constraints, relationships, and more. The "PARAM,NSPRT,1" card is saved to an array and is used to output data for each cycle. Next, the "EIGRL" card is used to define the number of modes to be considered in the buckling analysis. The set ID of this card is saved for use in the case control script.

Next, three strings are appended to the bulk data array. These strings will be written to the .BDF file and take advantage of the "Include" function in Nastran. By typing, "Include" and the filename of the file you wish to include, the nodes, elements, and loads files can easily be written to the design optimization file. Because the only thing that varies in the naming of the files is the run number, this is treated as a variable, CaseNum, so that the code includes the correct files.

In order to perform optimization, a target variable must be selected. Thus, weight is defined as the constraint for the response and is saved into the appropriate array.

The next section of coding handles the buckling analysis. The first ten modes are important in buckling analysis so, ten "DRESP1" and "DCONSTR" cards are generated in a for loop. These set the response type and constraint for the buckling analysis and are appended to the buckling array.

The next script section handles the frequency response and modal analysis of the vehicle. A "DRESP1" card defines the response type as being a frequency response and selects the mode at which to look. For the launch case, the first rigid-body mode is the one that should be optimized. Then, a design constraint card defines the upper and lower bounds of the frequency response. An arbitrary high value of 1000 Hz is chosen. The lower bound represents the lower

limit of the allowed natural frequency and was set to 7 Hz. It was noticed that the value of this lower bound has a significant effect on the mass of the vehicle. As the design is improved, the vehicle should be optimized for a higher frequency of 10 Hz instead. This limit comes from the frequencies expected during launch and is the minimum requirement to insure that the vehicle does not fail due to normal modes during launch. The current design optimizes to unreasonable masses with the 10 Hz limit and that is why the lower bound was reduced to 7 Hz.

Next, the shell sections are assigned two design variables each. One is the thickness of the honeycomb core, and the second is the thickness of one composite plate. Recall that since symmetry was invoked in the "PCOMP" card, the top and bottom plates are of equal thickness. A "DVPREL1" card is used to directly relate the specific property in the "PCOMP" card to the design variable. The response type for the thicknesses is "CSTRESS" and the design constraint is the maximum allowable stress for the material. Each card is stored to the proper array.

The longerons are assigned four design variables, which correspond to the four cross-sectional dimensions of the bar. The design constraint for the bars is the maximum allowable stress for the material. For the bars, it is necessary to utilize an equation to properly relate the dimensions of the cross-section as shown in Figure 28. The equation defined in the "DEQATN" card is a ratio of two dimensions. The limits for this ratio are defined in the "DRESP2" card and are shown in Equation 21 and 22. These constraints come from the Nastran Design Optimization User Guide. The geometry constraints exist so that if one dimension is changed, it still remains the same shape. For example, DIM3 cannot be less than DIM2. And, when it is equal to DIM2, the new shape has a rectangular beam cross-section. The buckling constraint is used to maintain a stiff structure and assure that DIM3 or DIM4 do not become excessively large.

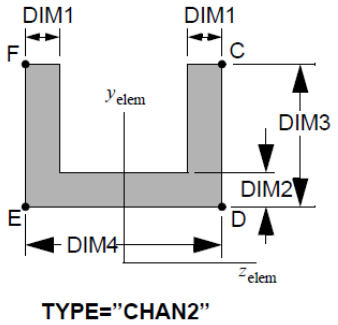

Figure 28: CHAN2 dimensions.

Equation 21: CHAN2 Geometry constraints.

$$Dim4 \geq 2 * Dim1 \rightarrow \frac{Dim1}{Dim4} \leq 0.5$$

$$Dim3 \geq Dim2 \rightarrow \frac{Dim2}{Dim3} \leq 1.0$$

Equation 22: CHAN2 buckling constraints.

$$\frac{Dim2}{Dim4} \leq 0.125, \quad \frac{Dim1}{Dim3} \leq 0.125$$

The frames section is coded the same as the longeron section except that it exists inside of a loop to create unique design variables for each frame. Since it shares the same CHAN2 cross-section, the relationships in Equation 21 and Equation 22 are also used to constrain the dimensions and avoid infeasible cross-sectional sizes.

Finally, the design optimization parameters are set with the "DOPTPRM" card. The optimization is given a maximum number of cycles of 50. This prevents an infeasible run from running infinitely many cycles. Another command tells the optimization to print the design variables after each cycle. This tells the user how the variables were changed after each run. A sample output is shown in Figure 29. The arrays are all converted to strings following the coding structure previously used and the text is written to the Design Optimization bulk data file. The outputs are listed in Table 18.

```
                                          ******************************************
                                          *                                        *
                                          *     D E S I G N   C Y C L E      1     *
                                          *                                        *
                                          ******************************************


                          *****   OPTIMIZATION RESULTS BASED ON THE APPROXIMATE MODEL   *****

                                        -----   DESIGN OBJECTIVE   -----

------------------------------------------------------------------------------------------------------------------------------------------
        INTERNAL                                MINIMIZE
        RESPONSE                RESPONSE          OR       SUPERELEMENT    SUBCASE        INPUT          OUTPUT
          ID        DRESPx        TYPE          MAXIMIZE        ID            ID           VALUE          VALUE
------------------------------------------------------------------------------------------------------------------------------------------
               1     DRESP1     WEIGHT        MINIMIZE            0             0      2.0026E+05      1.6251E+05

                                        -----   DESIGN VARIABLES   -----

------------------------------------------------------------------------------------------------------------------------------------------
      INTERNAL       DESVAR                          LOWER            INPUT            OUTPUT            UPPER
        ID             ID          LABEL             BOUND            VALUE             VALUE            BOUND
------------------------------------------------------------------------------------------------------------------------------------------
          1              1      CORE_0          1.0000E-02       1.0000E-01       5.9338E-02       1.0000E-01
          2              2      LAM_0           1.0000E-03       2.0000E-02       1.2153E-03       2.0000E-02
          3              3      CORE_1          1.0000E-02       1.0000E-01       6.6368E-02       1.0000E-01
          4              4      LAM_1           1.0000E-03       2.0000E-02       3.0528E-03       2.0000E-02
          5              5      CORE_2          1.0000E-02       1.0000E-01       7.2945E-02       1.0000E-01
          6              6      LAM_2           1.0000E-03       2.0000E-02       7.3344E-03       2.0000E-02
          7              7      CORE_3          1.0000E-02       1.0000E-01       7.8625E-02       1.0000E-01
          8              8      LAM_3           1.0000E-03       2.0000E-02       1.2379E-02       2.0000E-02
          9              9      CORE_4          1.0000E-02       1.0000E-01       8.3793E-02       1.0000E-01
         10             10      LAM_4           1.0000E-03       2.0000E-02       1.7910E-02       2.0000E-02
                                                                                  APRIL  22, 2015  MSC Nastran 11/13/14   PAGE    37
```

Figure 29: DOPTPRM, P1, 1, P2, 1 sample output.

Table 18: Design optimization bulk data outputs.

Input	Type	Description
EIGRL	Integer	EIGRL card ID.
DRESP1_target_var	Integer	DRESP1 card ID for target variable, weight.
DCID1	Integer	Static case set ID.
DCID2	Integer	Buckling case set ID.
DCID3	Integer	Modes case set ID.

5 Parametric Model

5.1 MyNastran.py

This script was developed by Jamshid Samareh and was adapted for use in this parametric model. This code is responsible for invoking Nastran and performing the finite element analysis through Nastran. It also cleans up the generated subfolders and deletes unnecessary file types.

5.2 Main Script

All the previously described scripts each serve a specific purpose. But to generate the model in its entirety and to be able to read the user inputs spreadsheet, the MidLOverD.py script must be run. This code compiles all of the functions developed to generate grid points, mesh the model, apply the loads and boundary conditions, and run each separate analysis for an infinite number of runs listed on the spreadsheet as depicted in Figure 31. There should be a main folder where all the scripts and spreadsheets are contained. It should look like Figure 30. With all these documents saved to the same folder, only the MidLOverD.py script needs to be run. When the program is run, a series of case folders are created to store each run. A successful run should have a folder with the contents pictured in Figure 31. Figure 32 provides an overview of the processes performed and files saved in a run of the main script.

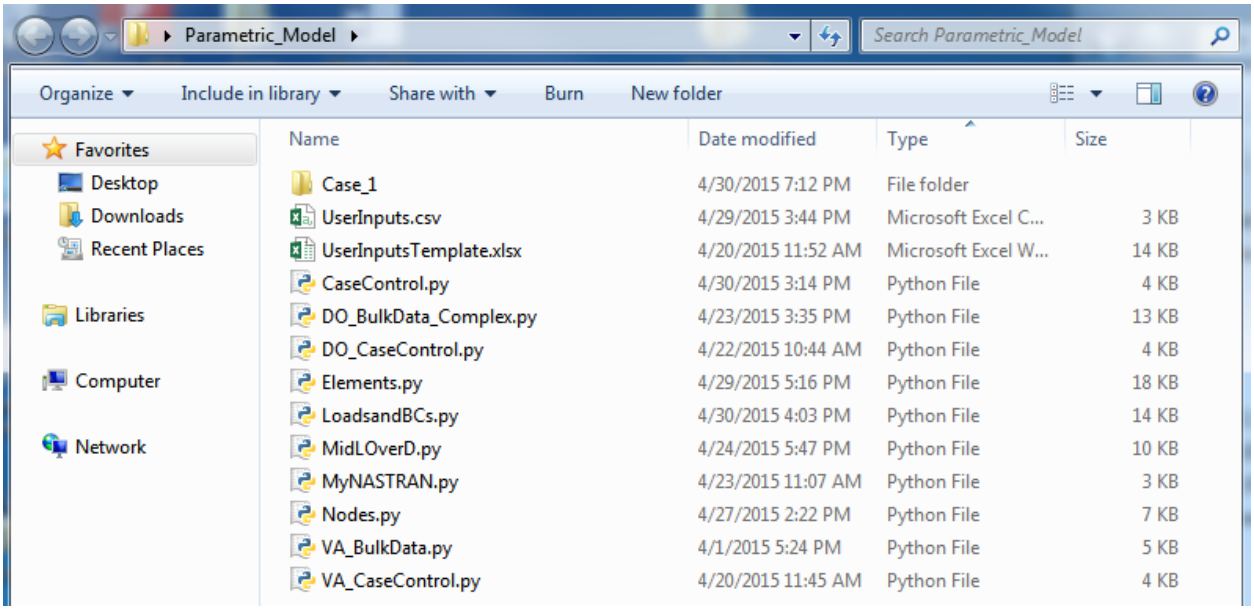

Figure 30: Main folder.

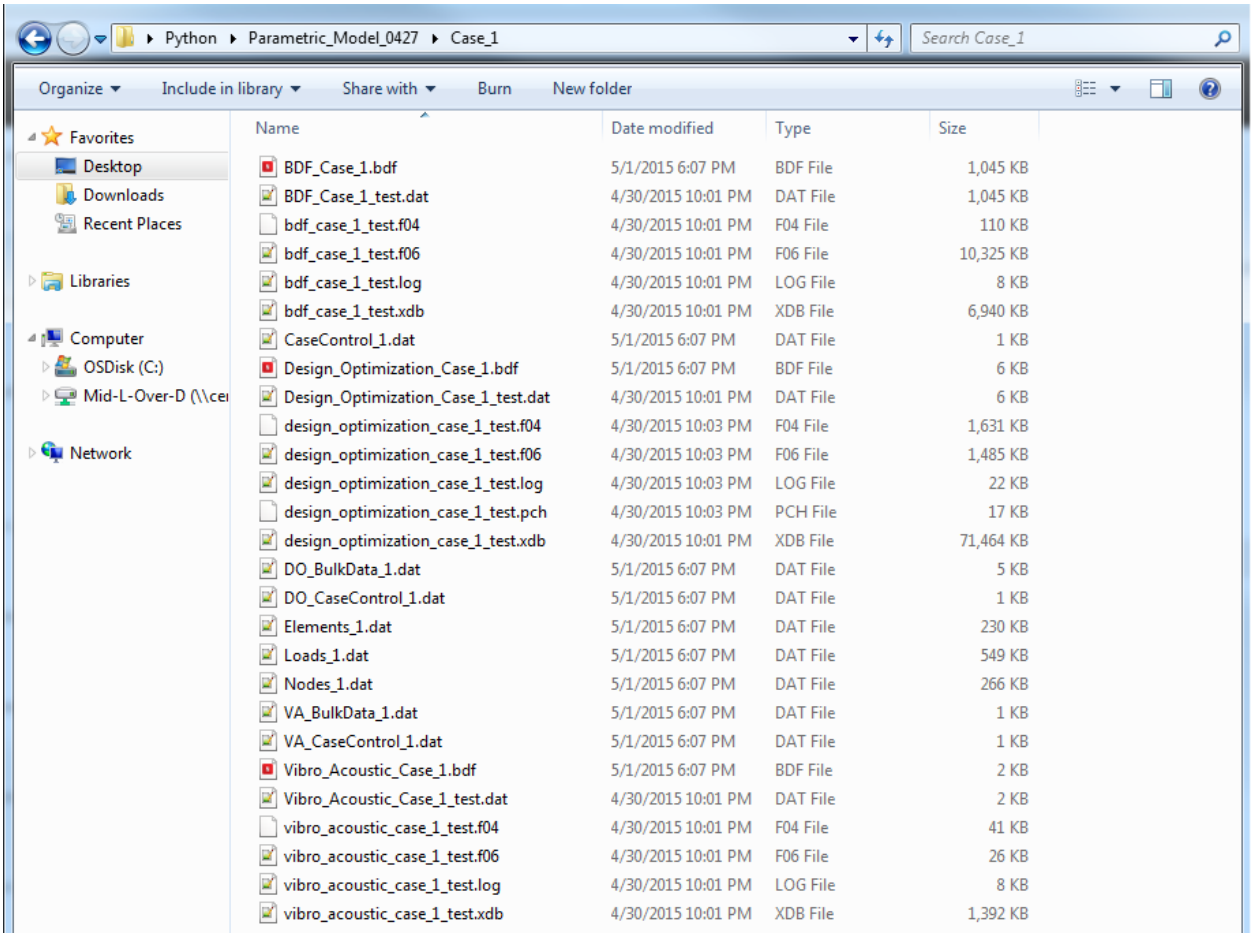

Figure 31: Case folder.

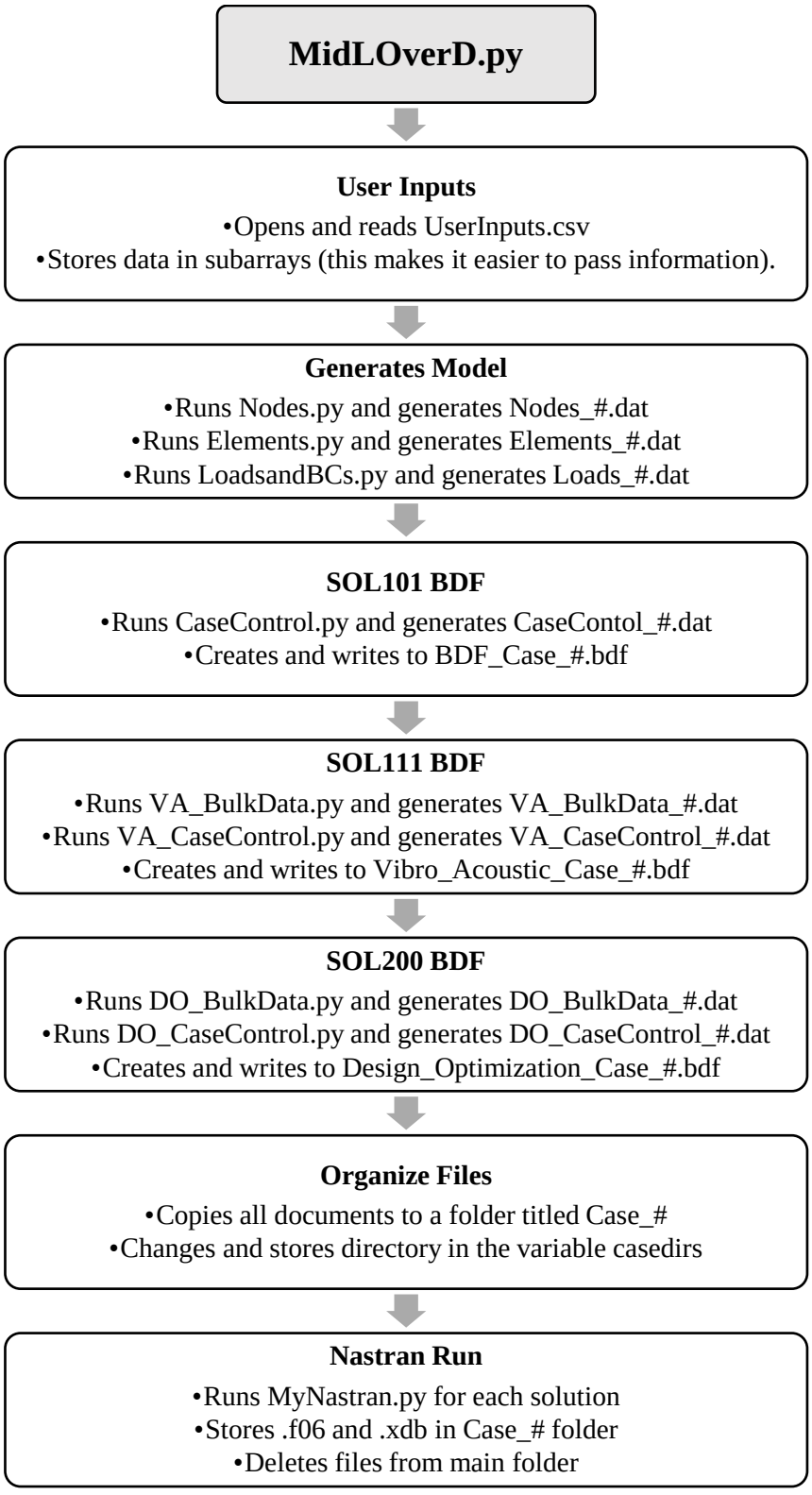

Figure 32: MidLOverD.py structure.

6 Results

This section reports the results of the parametric study. Various user inputs were tested to see their specific effect on the model. These results are plotted against the structural mass of the model and the area density of the model. The structural mass is the mass of the bars and shell elements combined. It neglects the mass of the TPS and the payload. Area density is the average area density of the model. The area density is calculated by dividing the mass of the structure by the surface area of the model.

6.1 Effect of Diameter

To determine the effect of the diameter on the aeroshell structure, all the inputs were held constant while only diameter changed. This was done for structures of three different lengths: 10m, 20m and 30m. The results of the design optimization are plotted below in Figure 33 and Figure 34.

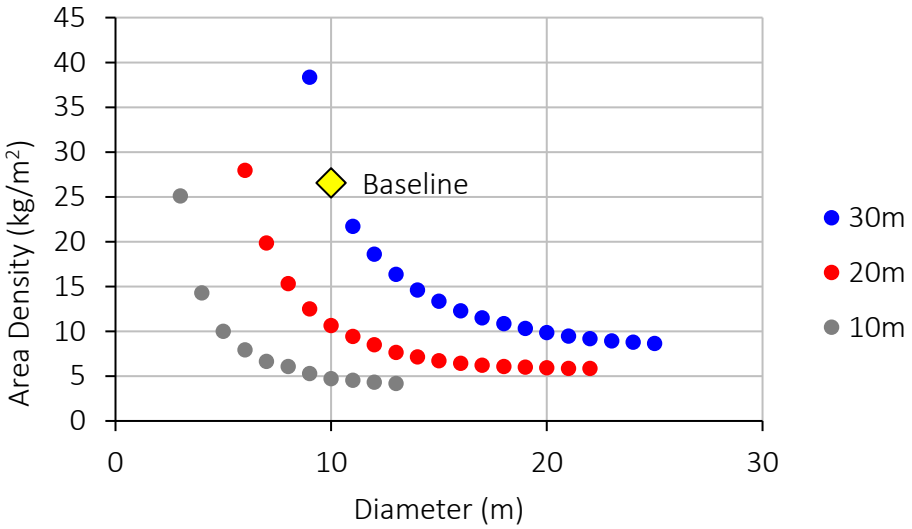

Figure 33: Effect of diameter on area density.

Area density is defined as the mass per unit surface area of the structure. Figure 33 shows that as diameter increases, area density decreases and seems to approach a limit. This would imply that a larger diameter makes for a significantly lighter structure. However, this is not necessarily feasible as the payload bay section of the SLS rocket is about 10 meters in diameter. Making the aeroshell significantly larger would have extreme effects on the aerodynamics of the launch vehicle. These data can also be interpreted to see the effect of rigidity on the model. With the baseline number of longerons and frames, the shells are highly susceptible to buckling failure and resonance at low frequencies. Thus, the optimization increases the rigidity of the structure using the dimensions.

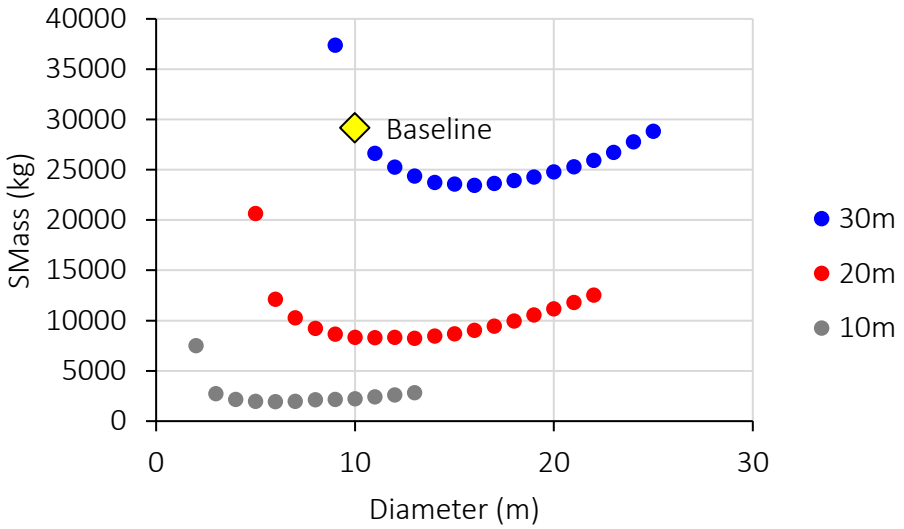

Figure 34: Effect of diameter on mass.

Figure 34 shows that there is a minimum mass for each diameter and length. Looking at the location of the baseline model, it is implied that the model can be reduced in mass by changing the diameter or the length. A length change would have a much larger effect than a diameter change as shown by the separate curves in the graph. Again, this graph shows that structures with a smaller ratio of length to diameter have lower structural masses. This is due to their increased stiffness and resilience to the effect of buckling and vibration modes.

6.2 Effect of Length

To determine the effect of length on the optimization of the structure, three different diameters were tested for various lengths. All the other parameters were held constant and only the length was changed. All of the structures tested used the baseline configuration of six frames, six shell sections, and eight longerons.

Figure 35 shows that as length increases, area density increases. For smaller diameters, area density increases faster with length. This, again, reiterates the effect of stiffening the structure. Overall, the trend is that an increase in length causes an increase in mass. The relationship between the amount of material and the mass of the material is a linear relationship. Both Figure 35 and Figure 36 show exponentially increasing relationships between length and mass or density. This is because as the structure becomes larger and longer, there are higher bending moments. This forces the design optimization to provide additional reinforcement.

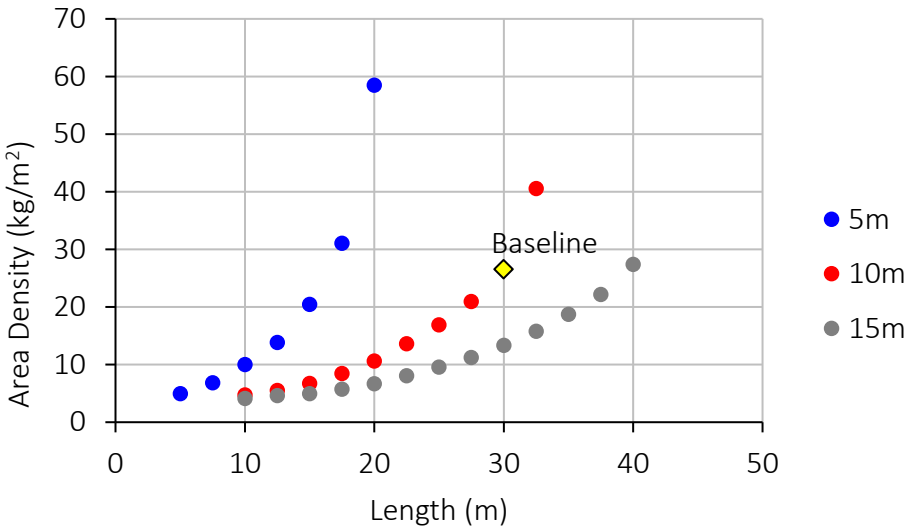

Figure 35: Effect of length on area density.

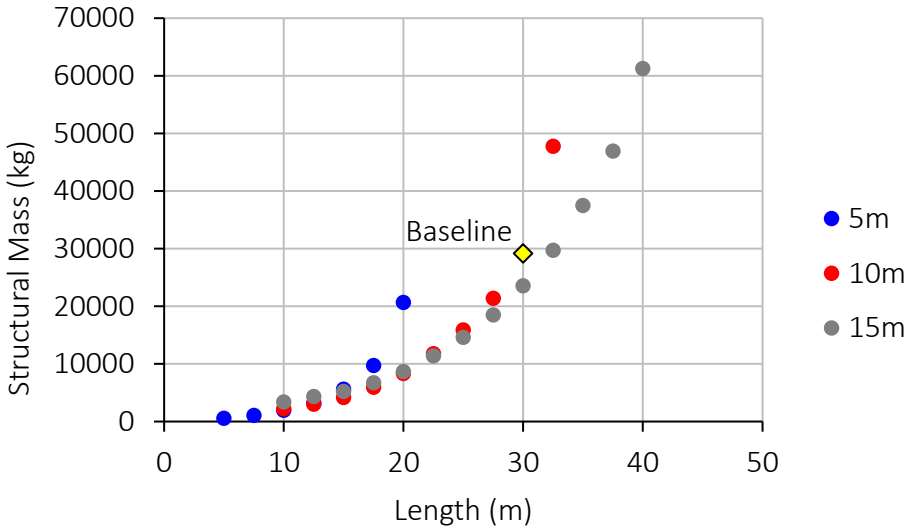

Figure 36: Effect of length on mass.

One interesting thing to note is the 10 to 15 meter length section of Figure 36. In this range, there is a very small difference in the masses of the structure but very large differences in the sizes of the model. This suggests that a 15x15m structure could be used over a 5x15m structure. A significantly larger structure can be used to fit the payload without adding on significantly more mass.

6.3 Effect of Longerons

The effect of the longerons was explored by testing different numbers of longerons on the same model. This was done for the baseline model as well as a 13x20m model. Both models have eight longerons and six shell sections. The results are shown in Figure 37. The plots show that the structural mass decreases with an increasing number of longerons. This is because the

longerons mitigate the effect of buckling and normal modes in the launch loads case, which governs the design. The longerons are located where the boundary conditions are. Thus, with more longerons, the load is better distributed circumferentially. This supports the conclusion that modal requirements are the biggest driver of the model [4].

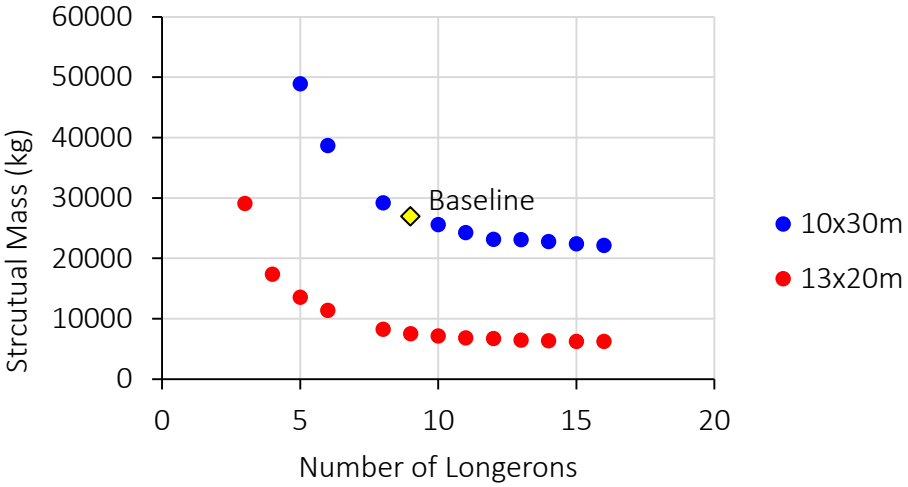

Figure 37: Effect of the number of longerons on structural mass.

6.4 Effect of Frames

The effect of the frames was analyzed by testing different numbers of frames on the same structure. The structure that was used for testing was the 10m diameter by 30m length baseline model. It has eight longerons and six shell sections. As shown in Figure 38, increasing the number of frames results in an increase in mass. This suggests that the frames do not mitigate normal modes or buckling. Therefore, the number of frames can be minimized to the number of frames necessary for attaching the payload. Halving the number of frames in the structure can reduce the baseline model's mass.

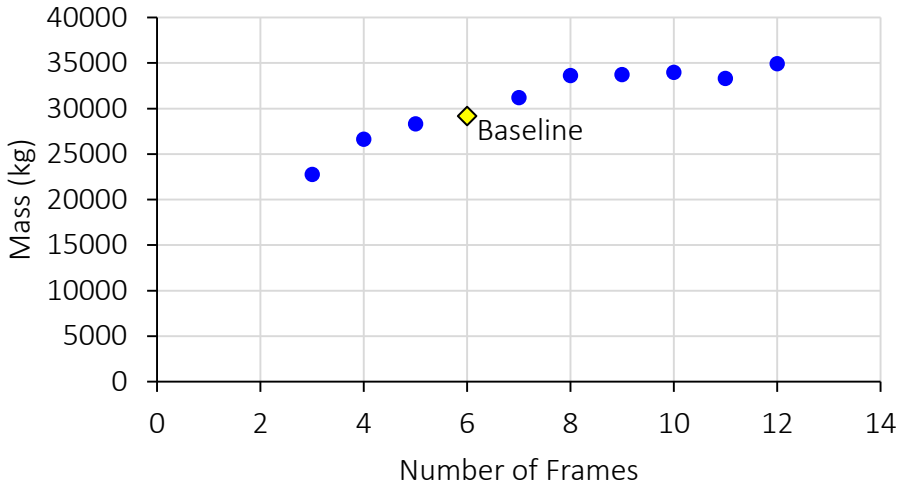

Figure 38: Effect of the number of frames on structural mass.

6.5 Effect of Shell Sections

The effect of the number of shell sections was analyzed by giving the program several different number inputs for the shell sections while keeping the rest of the model constant. This analysis was performed on the baseline model.

In Figure 39, it is evident that six shell sections is the minimum number of shell sections necessary to get a good approximation of the structural mass. Using more than six sections does not make the approximation any more accurate. Therefore, it can be concluded that the baseline model does not need to be changed from its original design with six shell sections. Introducing more shell sections would increase the number of design variables and consequently, the run time for the optimization. At this stage in the analysis, it is best to keep the model as simple as possible without sacrificing the validity of the results. Using only six shell sections does not negatively impact the data.

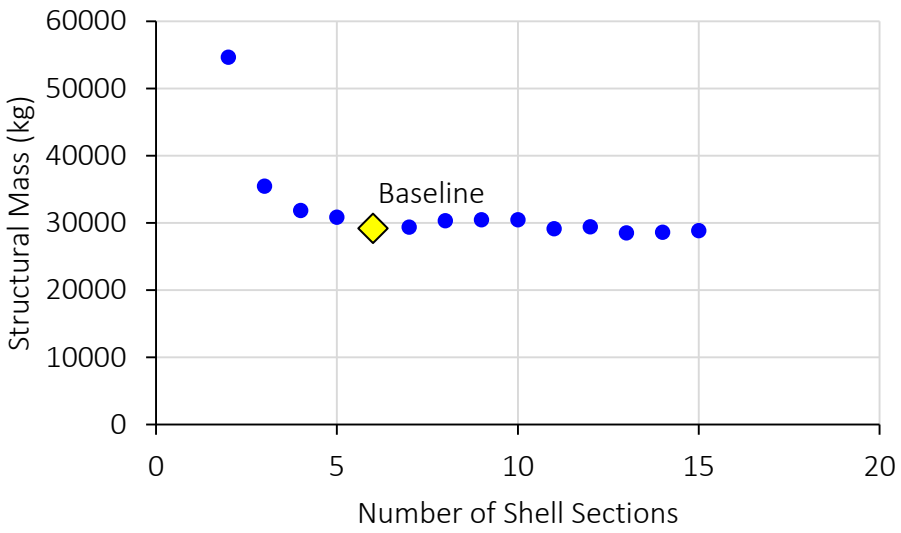

Figure 39: Effect of the number of shell sections on structural mass.

6.6 Effect of the Mesh

To determine the effect of mesh sizing on the optimization results, the two mesh variables in the user inputs document were varied. The current mesh on the baseline model uses 5056 shell elements (not including the bar elements). The values used to declare this mesh were increased by twenty percent for each iteration. The effect of this is plotted in Figure 40. This figure conveys that the mesh sizing does not have an appreciable effect on the structural mass. A different number of elements does not cause the model to optimize differently or to varying final masses. The number of elements has almost no effect on the mass. Therefore, the further analysis can be performed on a model with a coarse mesh. The ability to use a coarse mesh significantly decreases computational time and is a huge advantage to this model.

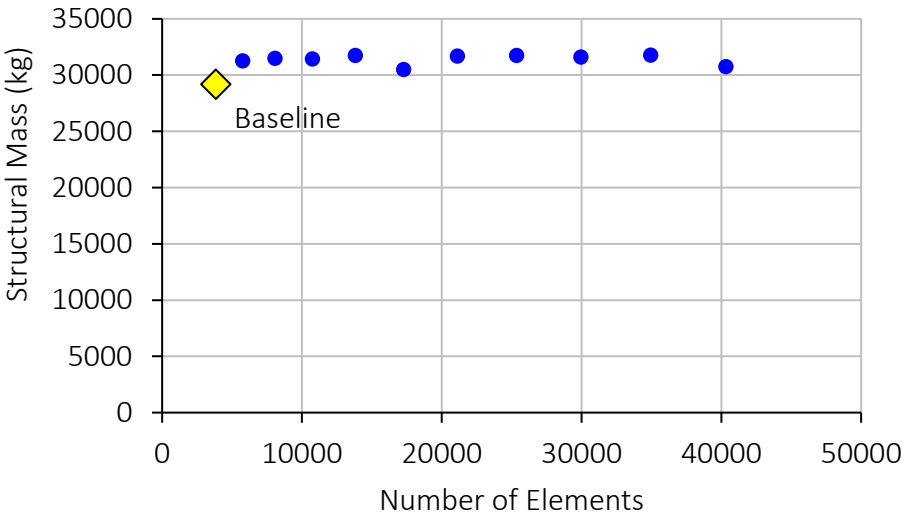

Figure 40: Effect of the mesh sizing on structural mass.

6.7 Volume Relationship

The data already obtained are now presented in a different manner. This time, the volume of the aeroshell is examined against its effect on structural mass and area density. The data are divided into groups to emphasize that length has a large impact on the mass. Each set of colored data points is representative of a different length structure.

It is evident in Figure 41 that there exists a minimum mass for each set of aeroshell lengths. It can also be concluded that there are preferred dimensions. The same volume can have a significantly lower mass if the lengths and diameters are changed. For example, the 13m diameter by 20m length structure has nearly the same volume as the 10m by 30m structure, however, the mass is almost 20 Mt less.

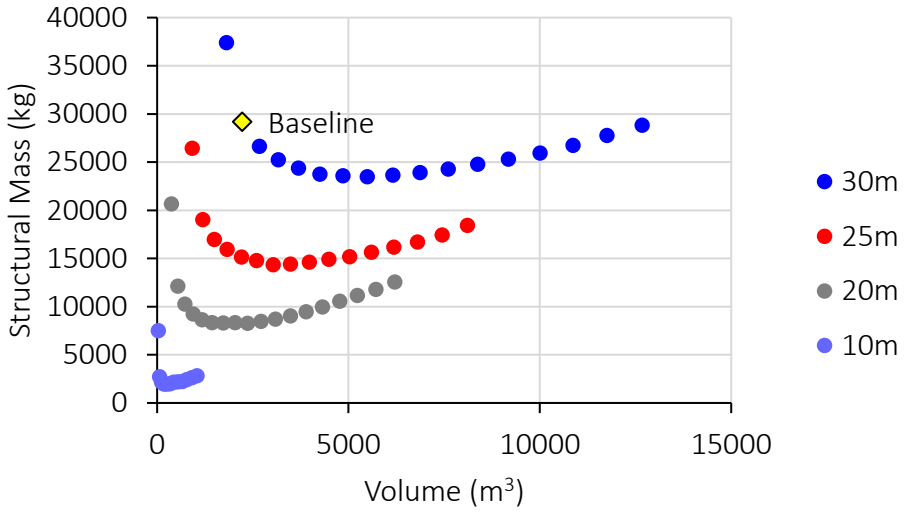

Figure 41: Effect of volume on mass.

In Figure 42, the plot of area density shows that smaller volumes yield much higher masses. This implies that the size does not pose a limit on the structure. However, as inferred from the other data, the structure must be very rigid. Area density remains very low for high volume structures. Length, again, has a large influence. The baseline structure lies on the highest curve and has a relatively high area density. Dimension changes to the baseline would incur significant improvements.

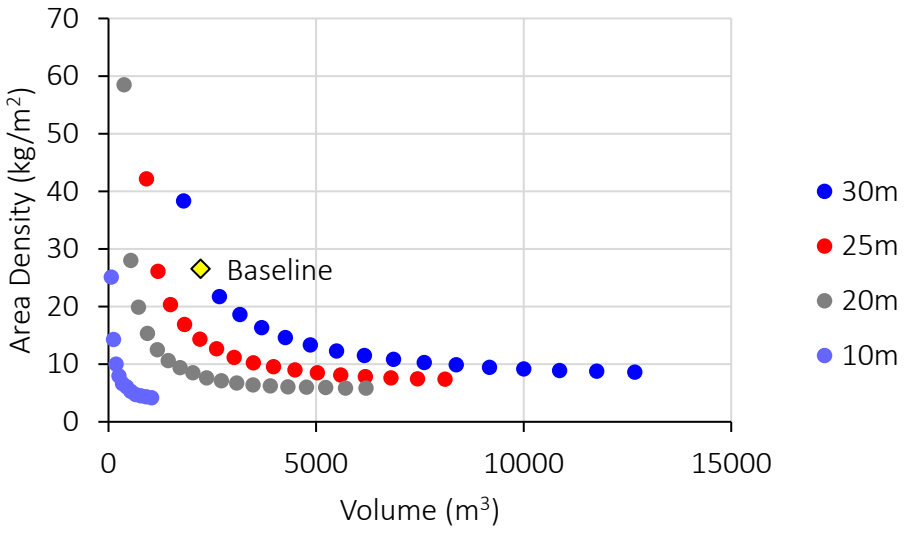

Figure 42: Effect of volume on area density.

6.8 Aspect Ratio

The previous information is presented in yet another manner. Aspect ratio was calculated by dividing the length of the structure by the diameter. The general trend of Figure 43 is that area density increases with increasing aspect ratio, and increasing length. Actual values of existing spacecraft are plotted to show the validity of the numbers. Reasonable area density values can be achieved with a mars entry structure with an aspect ratio of about two. The current baseline model has a very high area density compared to the area densities of actual spacecraft. It should also be noted that around an aspect ratio of one, there are less data points. This is because many designs become infeasible at an aspect ratio of one.

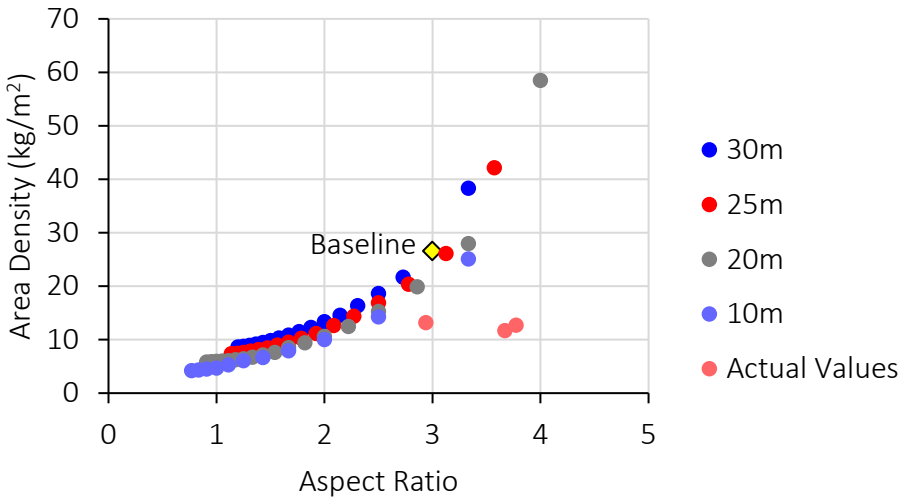

Figure 43: Effect of aspect ratio on area density.

Figure 44 shows that there is an optimal aspect ratio for each length of structures. A 30m length structure is optimized at an aspect ratio of about two. However, aspect ratio itself has a minimal effect on the mass. For example, in the case of the 30m length structure, there is a mass range of only 5000 kg. A much larger mass change can be associated with the length of the structure.

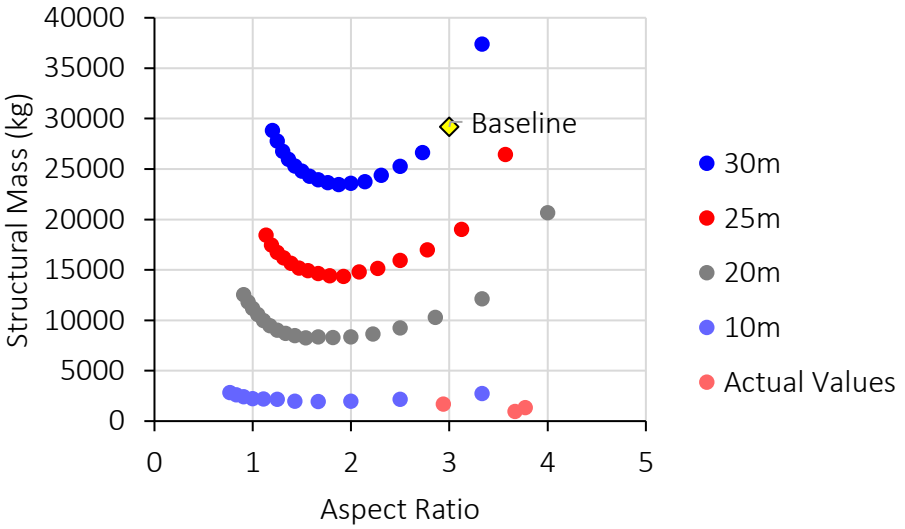

Figure 44: Effect of aspect ratio on mass.

6.9 Conclusion

The results of the parametric study emphasize the large impacts of the sizing and structure of the entry vehicle. Slight modifications to the structure or dimensions of the vehicle can significantly reduce the structural mass. Overall, the structure needs to be stiffened using batons and the length should be minimized. The largest feasible diameter should also be used.

And, as new information is provided for the SLS, modal requirements should be adjusted to better represent the entry vehicle on launch. It may be possible to lower the modal constraints, which in turn would significantly lower the mass as the structure would not need to be as rigid as currently needed.

Most importantly, the parametric study reveals the speed and flexibility of this approach. The data in the plots were obtained in one run of the code. The parameters were entered into the Excel spreadsheet, and the code was run. The entire process was automated and required no supervision as it ran for a day. All the results were organized into easy to understand folders. This process saves time for the engineer. Typical design processes would only test a few design options because the preliminary design and analyses of crude designs are costly and time-consuming endeavors. This code revolutionizes the design process and would avoid prematurely discarding design options. It gives you the capability of exploring all options and reduces the time in developing conceptual designs.

7 Future Recommendations

7.1 Shell Elements

Eventually, the PCOMP card should be implemented. This means that instead of treating the carbon fiber as a black box, it should be entered in the PCOMP card layer by layer. This would better model the material as it would account for shearing between layers. At least four layers should be entered at the 0, ±45, and 90 degree directions. Each layer should have the properties of an orthotropic material since one ply of carbon fiber has different properties in different directions. This change would allow for Nastran to optimize the number of layers in each direction, which should improve the overall optimization results. This modification would increase the number of design variables and so would require additional coding in the design optimization scripts.

7.2 Statics and Vibrations Analysis

Because of the order in which things were completed, the statics and vibrations analysis were set up to take a series of inputs and run the analysis. This means that the BDF's for these analyses are taking the initial dimensions and applying these to the shells and beams. These are the same initial dimensions that the optimization analysis takes and decreases to optimize the design. It would be beneficial to write a code that reads the .f06 file from the design optimization and uses these optimized dimensions in the statics and vibrations analysis rather than the initial dimensions.

7.3 Cross Sections

It would be powerful to allow the user to select the cross section for the bar elements. This was not done due to time constraints, however, it would allow for more extensive analysis and probably even more reduction of mass. The current model is set up for the CHAN2 cross-section. Other cross-sections that should be considered are T and HAT cross-sections. The most challenging part of implementing this change will be the design optimization set up. Different equations and relationships will be needed to properly dimension the cross-section.

7.4 Batons

Through the parametric study, it became evident that failure to meet modal requirements drove an increase in the mass of the design significantly. It may prove useful to add batons to the model to avoid resonance at low frequencies and to avoid buckling. The results showed that increasing the longerons decreased the mass. This is because at some point, the longerons stop acting as a structure and more like an extension or thickening of the shell. Adding batons that cross through the frames and longerons at an angle may lower the frequency response of the vehicle and significantly reduce the mass.

Another option can be to revisit the modal constraints set on the modal analysis. The current analysis uses a lower bound of 7 Hz. This value should be changed if new information becomes available and suggests otherwise. Lowering this value would significantly lower the mass of the vehicle as it would no longer be driven by the dynamic response.

7.5 Nose Tip Bias

The preliminary models have very coarse meshes. When the nose bias function was tested, it was tested for this coarse mesh and slightly finer meshes. Thus, it went unnoticed until much later, but an extremely fine mesh has a poor nose tip bias. Thus, if this model is used to test very fine meshes for more accurate results, the function should be improved so that the nose tip is meshed well. This could just be a simple modification of the constants used in defining the nose tip bias function. Another solution could be to use a cosine function instead of using an exponential function to develop the nose-tip bias.

7.6 Cross-Section Dimensions

Initial dimensions have a very large impact on the result of the design optimization. Thus, it may be beneficial to make the initial values a user input. It was neglected from the current code for the sake of simplicity since adding these initial dimensions to the user inputs file adds a total of 16 more input columns.

Also, depending on the cross section, the upper and lower bounds of each dimension change. For certain cross-sections, like CHAN2, dimensions one and three cannot have lower and upper bounds that overlap. Therefore, a middle bound must be calculated. As of now, the middle bound is an input directly in the code. This should also be made a user input. Or, a function should be used to calculate the middle bound.

7.7 File Clean-up

When the analyses are run, files of very large sizes are created. In order to free up space on the computer, a code should be developed to delete the unnecessary files, especially the large files.

8 References

[1] Dwyer, A., et al., "Entry, Descent and Landing Systems Analysis Study: Phase 1 Report," NASA TM-216720, 2010.

[2] MD Nastran 2006 r1 Quick Reference Guide Vol. 1-3, MSC Software Corporation, 2006.

[3] Anderson Jr., J. D., *Hypersonic and High Temperature Gas Dynamics*, American Institute of Aeronautics and Astronautics, 1989.

[4] Ahmed, S., Lane, B., "Finite Element Modeling and Analysis of Mars Entry Aeroshell Baseline Concept," NASA TM-219374, 2017.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-02-2017	Technical Memorandum	

4. TITLE AND SUBTITLE

Parametric Structural Model for a Mars Entry Concept

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Lane, Brittney M.; Ahmed, Samee W.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

388496.04.01.02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L-20754

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

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA-TM-2017-219375

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified
Subject Category 39
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This paper outlines the process of developing a parametric model for a vehicle that can withstand Earth launch and Mars entry conditions. This model allows the user to change a variety of parameters ranging from dimensions and meshing to materials and atmospheric entry angles to perform finite element analysis on the model for the specified load cases. While this work focuses on an aeroshell for Earth launch aboard the Space Launch System (SLS) and Mars entry, the model can be applied to different vehicles and destinations. This specific project derived from the need to deliver large payloads to Mars efficiently, safely, and cheaply. Doing so requires minimizing the structural mass of the body as much as possible. The code developed for this project allows for dozens of cases to be run with the single click of a button. The end result of the parametric model gives the user a sense of how the body reacts under different loading cases so that it can be optimized for its purpose. The data is reported in this paper and can provide engineers with a good understanding of the model and valuable information for improving the design of the vehicle. In addition, conclusions show that the frequency analysis drives the design and suggestions are made to reduce the significance of normal modes in the design.

15. SUBJECT TERMS

Mars entry concept; Parametric; Space launch system

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	64	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658