Formula UBC Torsional Stiffness Goal

Scott Fjordbotten

November 13, 2016

Contents

List of Figures

List of Tables

1 Project Description

The torsional stiffness of the FUBC chassis is an important design parameter as it determines how the vehicle will handle and the degree to which the suspension can be tuned to improve vehicle dynamics. There are a variety of rules of thumb and suggested values for torsional chassis stiffness available in literature on the topic of chassis design and vehicle dynamics, however, it is desirable to be able to complete our own analysis to either validate these rules of thumb or create our own torsional stiffness goal.

The purpose of this research project will be to identify the roll torsional chassis stiffness plays in vehicle dynamics, write a MATLAB script to allow analysis of the effects of chassis stiffness given a set of vehicle parameters, and ultimately provide a means for setting justified goals for chassis torsional stiffness during the design phase of subsequent FUBC cars.

1.1 Previous Consideration of Torsional Stiffness During Chassis Design

The importance of torsional stiffness and the analysis of torsional stiffness during design has been considered to a limited extent in the past. As far as I can tell, torsional stiffness goals have been set based on approximate values found in papers released by other Formula Student teams, and comparison to previous FUBC designs that have been deemed to be adequately stiff "by feel" while driving and tuning suspension.

The torsional stiffness of chassis design revisions are determined through FEA throughout the design process. In 2016, a torsional stiffness jig for physical testing was constructed. However, due to inadequate knowledge transfer from team members who had worked on the design of the jig over the past 3 or 4 years and inadequate design review, the jig did not perform as desired. Physical testing data was taken in 2016, but the test setup was far from ideal and the accuracy of the data was questionable; fortunately, the physical test data matched the FEA data reasonably well. In the future, a new torsional stiffness jig should be designed and constructed (adhering to a more formal and structured design and review process) to allow FEA results to be validated via physical testing.

1.2 Background Research and Performance Indicators

When a vehicle turns, the lateral acceleration of the turn coupled with height of the vehicle's mass components above the suspension roll center exert a moment about the lateral axis of the vehicle. This moment is reacted by vertical forces at the tires; the distribution of the reacting forces between the front and rear tires is called the lateral load distribution and dictates how the vehicle handles through the corner.

Ideally, the front and rear roll stiffness of the vehicle's suspension can be tuned to achieve the desired lateral load transfer distribution. In order for suspension roll stiffness to be the vehicle parameter dictating lateral load transfer distribution, the chassis must be adequately stiff in torsion. To demonstrate the importance of chassis stiffness, consider the front and rear suspension to be torsional springs and the chassis to be either a perfectly ridged or perfectly flexible rod connecting the suspension springs.

If Mlat is the moment experienced by the vehicle due to lateral acceleration, the moment exerted by the suspension must equal the moment caused by lateral acceleration for the vehicle to be static equilibrium:

$$M_{lat} = M_{front} + M_{rear}$$

Where M_{front} is the moment exerted by the front torsional spring and M_{rear} is the moment exerted by the rear torsional spring and the moment exerted by each spring is proportional to the stiffness of the springs:

$$M_{front} = K_{front}\theta_{front}$$

$$M_{rear} = K_{rear}\theta_{rear}$$

In the case of a perfectly ridged rod, the angular rotation of each torsional spring must be the same:

$$\theta_{front} = \theta_{rear} - \theta$$

The moment balance becomes:

$$M_{lat} = K_{front}\theta + K_{rear}\theta$$

And the proportion of the moment caused by lateral acceleration reacted by the front and rear of the vehicle is determined by the suspension stiffness as desired.

In the case of the perfectly flexible rod, the moment exerted by each torsional spring must be the same:

$$M_{front} = M_{rear} = M$$

or

$$K_{front}\theta_{front} = K_{rear}\theta_{rear}$$

The moment balance becomes:

$$M_{lat} = 2 \times M$$

And the proportion of the moment caused by lateral acceleration reacted by the front and rear of the vehicle does not change as suspension stiffness is varied.

In reality, a chassis is neither perfectly ridged nor perfectly flexible. The stiffness of the chassis is a function of the material used, geometry, and the size of the vehicle. For a given material, increasing the torsional stiffness of the chassis requires more efficient material use or more material (i.e. more mass); increasing the stiffness usually means increasing the mass of the chassis. As a low vehicle weight is desirable, it is important to determine what chassis stiffness is “stiff enough” so that vehicle mass is not increased unnecessarily.

While researching torsional stiffness goals, three different recommendations were discovered:

1. Deakin et al. [2] suggest that about 80% of the difference in front to rear roll stiffness must result in a difference in front to rear lateral load transfer.
2. Milliken and Milliken’s Race Car Vehicle Dynamics [4] states that the chassis stiffness can be approximately designed to be X times the total suspension roll stiffness, or X times the difference between front and rear suspension stiffness. X is said to be somewhere in the range of 3 - 5 times.
3. In Fundamentals of Automobile Body Structure Design [3], Malen suggests yet another rule of thumb. He states that suspension is designed assuming a ridged chassis and that to make this assumption valid, chassis stiffness should be approximately 10 times the total suspension stiffness.

Due to the conflicting suggestions, this project will determine which of the suggestions are feasible for an FSEA car and determine if one of the feasible suggestions will be followed or if another should be developed.

2 Feasibility

To evaluate stiffness feasibility, a MATLAB package was developed to solve forces in minimally stable, statically determinate 3D truss structures in which all members experience only tensile or compressive forces. A macro was also written to allow such truss structures to be extracted from 3D sketches in SolidWorks. These scripts can be found in their entirety in Appendix A.

The initial intension of the package was to analyze a simplified version of the FUBC chassis, however it was soon determined that the chassis could not be analyzed in this way. Instead, the chassis was simplified to a square rectangular prism with length equal to the vehicle's wheelbase and square side length ranging from the width of the front bulkhead to the designed track width for the FUBC 2017 car (Figure 1). The tubes in the prism were considered to be 1" OD (most tubes used in the 2016 FUBC care were this diameter) 0.057" thick tubes. This thickness is the length weighted average thickness of the tubes used in the 2016 FUBC car.

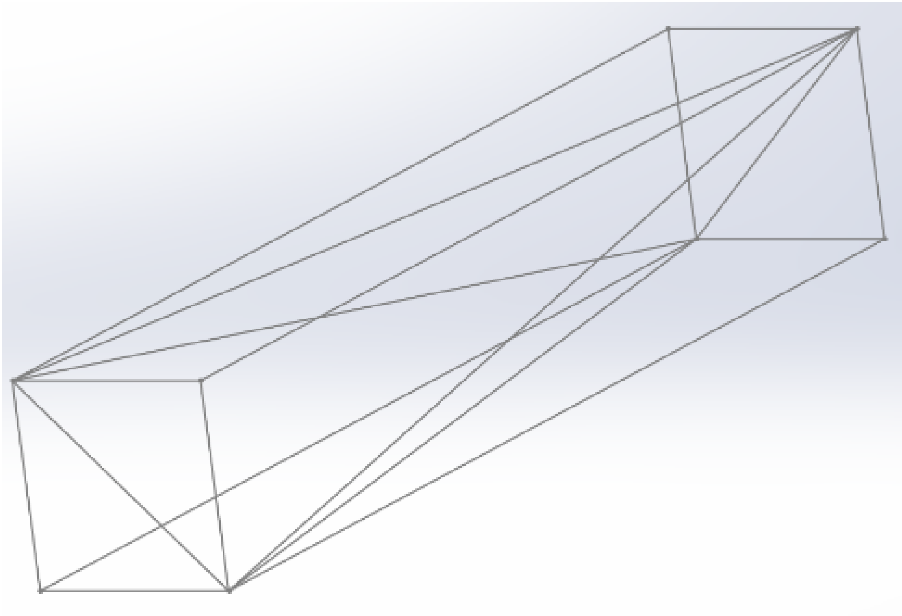

Figure 1: Solidworks sketch of simplified chassis model as a square rectangular prism

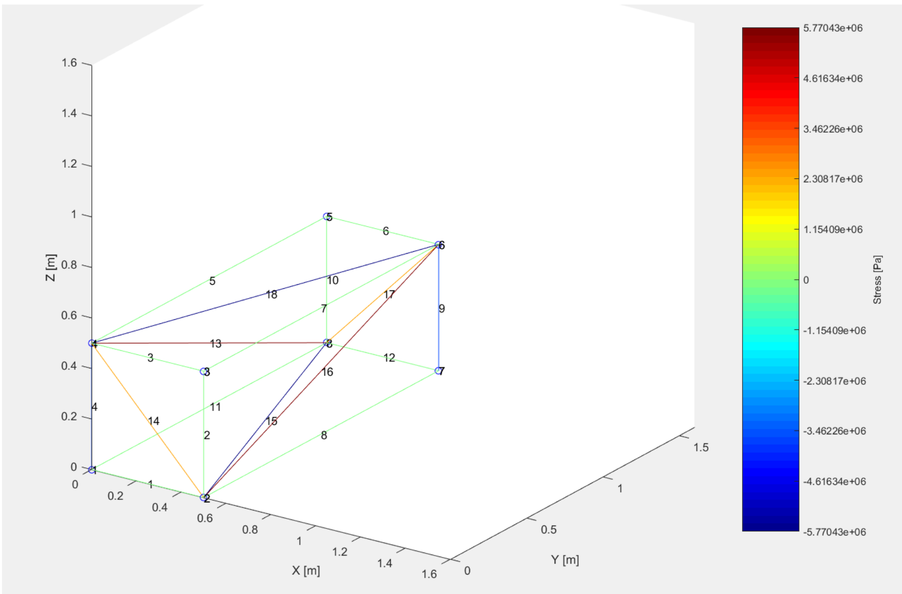

Figure 2: Simplified chassis model as a square rectangular prism imported into MATLAB

A torque was applied to one end of the structure, while the other was fixed and energy methods were used to determine the angle of rotation of the structure:

$$\frac{1}{2}T\theta = \Sigma(tubestrainenergy) = \Sigma\frac{\sigma_i \epsilon_i}{2}A_i L_i = \Sigma\frac{\sigma_i^2}{2E_i}A_i L_i \tag{2.1}$$

$$\theta = \frac{2}{T}\Sigma\frac{\sigma_i^2}{E_i}A_iL_i \tag{2.2}$$

Where T is the applied torque, subscript i denotes the ith tube, σ is the stress a tube, ϵ is the strain in a tube, E is the young's modulus for a tube, A is the cross sectional area of the tube, and L is the length of a tube.

Now, stiffness can calculated as:

$$K = \frac{T}{\theta} = \Sigma\frac{\sigma_i^2}{E_i}A_iL_i \tag{2.3}$$

Analysis was run and fourth order polynomial fit to the data was created to determine the approximate dimensions required to meet each suggested chassis stiffness. The suggestions were then evaluated based on the spatial constraints of the car. The preliminary 2017 FUBC vehicle parameters used are summarized in Table 1, and the results from analysis are summarized in Table 2. Due to the requirement of lateral load transfer distribution for the Deakin et al. suggestion, it was not considered at this stage.

Vehilce Property	Value [Unit]
Wheelbase	1.54 [m]
Front Track Width	1.22 [m]
Rear Track Width	1.19 [m]
Bulkhead Width	0.33 [m]
Widest Section (Main roll hoop)	0.67 [m]

Table 1: Projected 2017 FUBC vehicle parameters

Chassis Stiffness Suggestion	Stiffness Target [Nm/deg]	Square Side Length [m]
3X Roll Stiffness	2340	0.727
5X Roll Stiffness	3900	0.85
10X Roll Stiffness	7800	1.07

Table 2: Feasibility analysis results

The results immediately ruled out the Malen suggestion of 10X the roll stiffness as this is projected to require chassis dimensions nearly as large as the projected track width; this is clearly not feasible and will not be considered further. Milliken and Milliken's suggestion of 3-5 times the roll stiffness seems more reasonable; although the predicted chassis dimensions are larger than our projected design, it is important to acknowledge that there will be some error in this overly simple model of the chassis. Thus, the 3-5 times roll stiffness suggestion will be given further consideration in the next section.

3 Model and Script Development

In order to justify the rules of thumb for chassis stiffness, or to create a new way to set torsional stiffness goals at FUBC, the simple model described in section 1.2 to show the importance of torsional chassis stiffness was built upon. This model is based upon the model used by Eurenius et al. [1] and Deakin et al. [2] and will be described in detail in section 3.1.

3.1 The Model

Deakin et al. [2] developed a simple model for calculating the static forces present in the chassis under steady state conditions. This model considers the racing car to consist of two point masses, m_f and m_r for the front and rear respectively, connected by a torsional spring, K_{ch}, and suspension at each end of the vehicle represented by a roll stiffness, K_{rollf} and K_{rollr}, Figure 3.

This simple model was built upon by Eurenius et al. [1]; the vehicle mass is divided further into a sprung component, m_{sprung}, located at the vehicles center of mass and two unsprung components, $m_{frontUnsprung}$ and $m_{rearUnsprung}$, representing the unsprung mass at the front and rear axles, respectively. The unsprung masses are assumed to be at the height of the front and rear wheel centers; further, the unsprung mass distribution is assumed to be symmetric about the longitudinal axis of the vehicle. This model is shown in Figure 4.

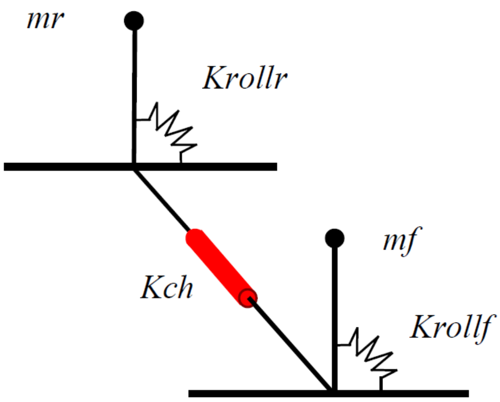

Figure 3: Deakin et al. [2] simplified vehicle model

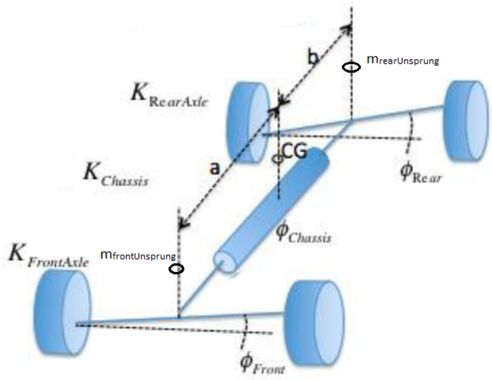

Figure 4: Eurenius et al. [1] simplified vehicle model

Both models assume a uniform chassis stiffness distribution along the length of the chassis. This allows Deakin et al. to divide the vehicle mass between the front and rear point masses based on the geometric location of the center of mass [2]. Similarly, Eurenius et al. divide the moment caused by lateral acceleration of the sprung mass between the front and rear suspension based on geometry [1].

Deakin et al. note that in reality neither the mass nor stiffness distribution of the chassis is uniform, so there will likely be some discrepancies between the simplified model and the actual vehicle. Further, compliances in the suspension, commonly referred to as the installation stiffness, reduce the chassis torsional stiffness as seen at the wheels. Installation stiffness should also be considered as possible cause of discrepancy between the idealized model and the real vehicle.

The model used by Eurenius et al. was adopted as is seems to take into account more of the vehicles mass distribution, which was expected to reduce the error in the idealized model.

Next the moment arms for each mass must be determined to determine the torque experienced by the suspension and chassis during steady state cornering. The method used is again based on the model used by Eurenius et al., in which the front and rear roll centers, wheel centers, and the height of the center of mass are used to calculate the moment arms. Since the roll centers for the front and rear suspension are known, the height of the unsprung masses (wheel centers) above their respective roll center is used as the moment arm for the unsprung masses. The moment arm for the sprung mass is determined by drawing a roll center line between the front and rear roll centers. The height of this line at the longitudinal position of the center of gravity can then be calculated and the moment arm is the difference between the height of the center of mass and the interpolated roll center height at the longitudinal position of the center of mass. This geometry is shown in Figure 5 and the following moment arm equations.

$$x = h - \frac{am + bn}{a + b} \qquad \text{(Sprung mass moment arm)} \tag{3.1}$$

$$\text{Front Unsprung Mass Moment Arm} = r_1 - n \tag{3.2}$$

$$\text{Rear Unsprung Mass Moment Arm} = r_2 - m \tag{3.3}$$

By choosing a lateral acceleration, a_{lat}, the moment exerted by each mass can be calculated:

$$M_{sprung} = a_{lat} m_{sprung} x \tag{3.4}$$

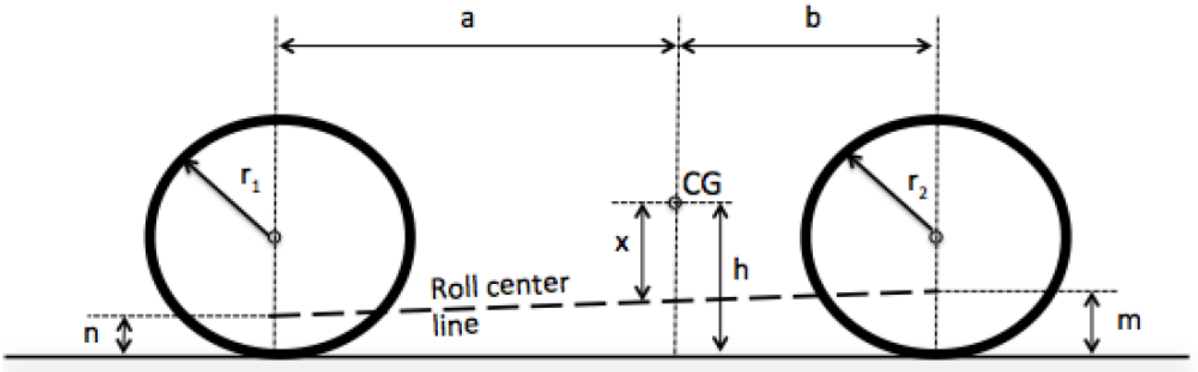

Figure 5: Mass and roll center geometry, from Eurenius et al. [1]

$$M_{FrontUnsprung} = a_{lat} m_{frontUnsprung}(r_1 - n) \tag{3.5}$$

$$M_{RearUnsprung} = a_{lat} m_{rearUnsprung}(r_2 - m) \tag{3.6}$$

The moment due to the sprung mass can then be distributed between the front and rear based on vehicle geometry:

$$M_{frontSprung} = \frac{b}{a+b} M_{sprung} \tag{3.7}$$

$$M_{rearSprung} = \frac{a}{a+b} M_{sprung} \tag{3.8}$$

Now the front and rear suspension springs will want to deflect angularly due to the applied moment and the stiffness of the chassis will apply moments to counteract any difference in suspension angular deflection. Using the resuts from equation 3.4 to 3.8, equating the applied and resultant moments, and requiring the difference in suspension angular deflections to be equal to the chassis angular deflection results in the following system of equations:

$$\begin{bmatrix} K_{front} & 0 & -K_{chassis} \\ 0 & K_{rear} & K_{chassis} \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \phi_{front} \\ \phi_{rear} \\ \phi_{chassis} \end{bmatrix} = \begin{bmatrix} M_{frontUnsprung} + \frac{b}{a+b} M_{sprung} \\ M_{rearUnsprung} + \frac{a}{a+b} M_{sprung} \\ 0 \end{bmatrix} = \begin{bmatrix} M_{front} \\ M_{rear} \\ 0 \end{bmatrix} \tag{3.9}$$

The above matrix equation can either be solve using matrix algebra and MATLAB, or by hand. The expressions for each angle of rotation are:

$$\phi_{front} = \frac{M_{front} K_{front} + K_{chassis}(M_{front} + M_{rear})}{K_{front} K_{rear} + K_{chassis}(K_{front} + K_{rear})} \tag{3.10}$$

$$\phi_{rear} = \frac{M_{rear} K_{front} + K_{chassis}(M_{front} + M_{rear})}{K_{front} K_{rear} + K_{chassis}(K_{front} + K_{rear})} \tag{3.11}$$

$$\phi_{chassis} = \frac{M_{rear} K_{front} - M_{front} K_{rear}}{K_{front} K_{rear} + K_{chassis}(K_{front} + K_{rear})} \tag{3.12}$$

The lateral load transfer, the load that shifts from the inside wheel to the outside wheel, at either the front or rear wheels is defined as:

$$LT = \frac{P_{outside} - P_{inside}}{2} \tag{3.13}$$

Using moment balance at the front axle, the moment caused by the vertical tire forces must equal the moment exerted by the front suspension (Figure 6).

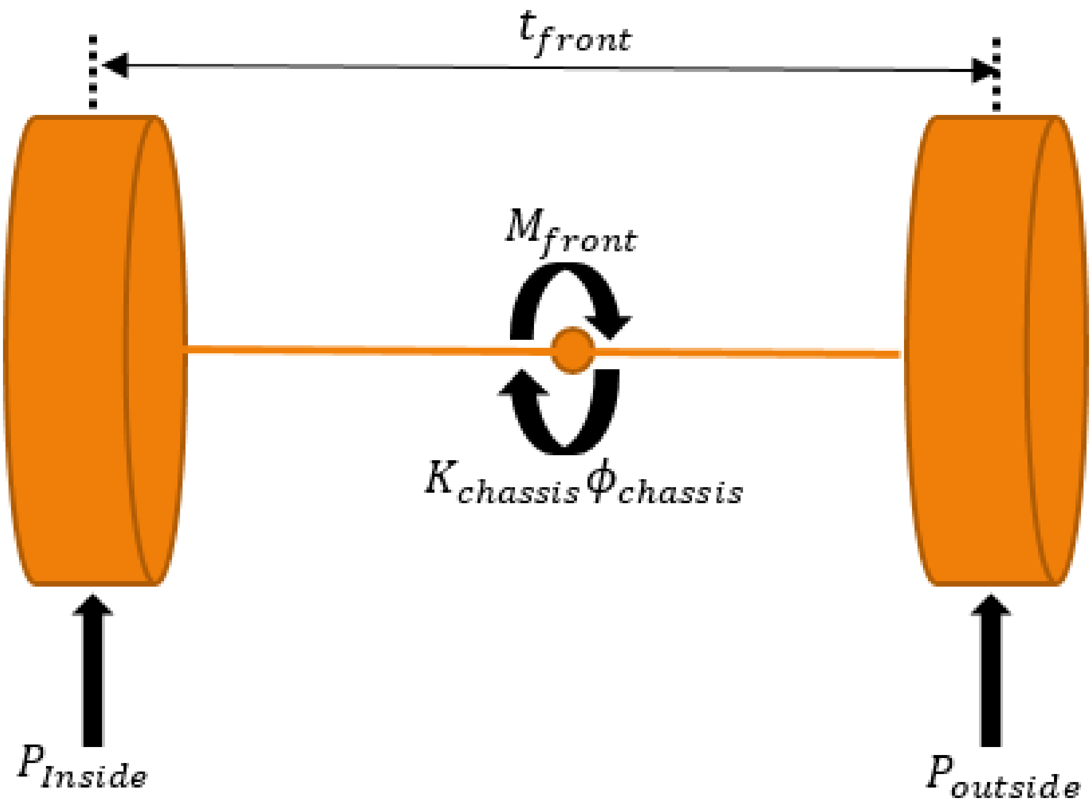

Figure 6: Front axle moment balance. Adapted from Eurenius et al. [1]

$$\frac{P_{outside} - P_{inside}}{2} t_{front} = LT_{front} t_{front} = K_{front}\phi_{front} = M_{front} + K_{chassis}\phi_{chassis}$$

Where t_{front} is the front track width. Solving for the load transfer at the front wheels:

$$LT_{front} = \frac{M_{front} + K_{chassis}\phi_{chassis}}{t_{front}} \tag{3.14}$$

Similarly for the rear wheels:

$$LT_{rear} = \frac{M_{rear} - K_{chassis}\phi_{chassis}}{t_{rear}} \tag{3.15}$$

Now the load transfer distribution, reported as the percentage of the load transfer that occurs at the front or rear wheels, can be calculated using equations 3.14 to 3.15:

$$LLTD_{\%front} = \frac{LT_{front}}{LT_{front} + LT_{rear}} \tag{3.16}$$

$$LLTD_{\%rear} = \frac{LT_{rear}}{LT_{front} + LT_{rear}} = 1 - LLTD_{\%front} \tag{3.17}$$

To determine chassis stiffness performance as described in Deakin et al., the relationship between LLTD and roll stiffness distribution (RSD) is considered. As described in section 1.2, an ideal rigid chassis allows the LLTD to be proportional to the RSD, this means that if LLTD is a function of RSD, the relationship will be approximately linear. To this end, LLTD should be plotted vs RSD and the slope in the region of interest (the near linear portion of the plot where $30\% \leq RSD \leq 70\%$). This slope may be used as a metric to evaluate chassis performance and has a maximum value of 1, corresponding to a ridged chassis.

$$Index = \frac{\partial LLTD_{front}}{\partial RSD_{front}} \tag{3.18}$$

3.2 The Script

The model described in section 3.1 was used to create a MATLAB script that, given vehicle parameters, could calculate the lateral toad transfer and proportion of roll stiffness distribution that translates into lateral load transfer distribution for a range of weight distributions, suspension roll stiffness distributions, and chassis stiffness's.

The script was broken into two parts:

1. A function calc_lltd_per_rsd.m: given vehicle properties and ranges of chassis stiffness and roll stiffness distribution to consider, calculates the index presented at the end of section 3.1 and both the front and rear load transferred for a range of roll stiffness distributions and chassis stiffness's.
2. A script LLTD.m: contains vehicle parameters and organizes the calling of calc_lltd_per_rsd.m and the plotting of its results.

The remainder of this section will explain how these two scripts function in more detail, what data was used to generate the plots presented in section 4, and how to use the scripts. The purpose of this section is to aid in understanding of the script for future use and development.

3.2.1 calc_lltd_per_rsd.m

This function takes the sprung mass, front and rear unsprung masses, center of gravity height (h in Figure 5), center of gravity distance from rear axle (b in Figure 5), front and rear wheel radii (r_1 and r_2 in Figure 5), front and rear roll center heights (m and n in Figure 5), front and rear track widths (t_{front} and t_{rear} from equations 3.14 and 3.15), wheelbase ($a + b$ from Figure 5), total roll stiffness ($K_{front} + K_{rear}$ from equation 3.9), minimum and maximum front roll stiffness distribution, minimum and maximum chassis stiffness, and lateral acceleration as parameters. The return vaues are a vector with the values of chassis stiffness considered, a vector with the result of equation 3.18 for each chassis stiffness considered, and matrices containing the front and rear load transfer for each chassis stiffness-Roll roll stiffness distribution pair considered.

The first section of the script sets up vectors to iterate through chassis stiffness and roll stiffness distributions as determined by the function input. Next, the dimensions a, b and x from Figure 5 are calculated.

Equations 3.4, 3.5, 3.6, 3.7, and 3.8 are used to calculate the moment experienced at the front and rear axles. Then, for each chassis stiffness and roll stiffness distribution, equations 3.9, 3.14, and 3.15 are used to calculate the load transfer at the front and rear wheels.

Finally, equations 3.16 to 3.18 are used to calculate the index measuring how much of the suspension roll stiffness distribution translates into lateral load transfer distribution. This is done by fitting a linear curve to the data $LLTD_{\%front} = f(RSD\%front)$, where the data is filtered such that only RSD corresponding to the middle half of the specified range is used (typically, the range 10% to 90% was passed to the function, making the range used 30% to 70%).

The function in its entirety can be found in Appendix B.

3.2.2 LLTD.m

This script defines the variables used by calc_lltf_per_rsd.m, defines what mass distributions will be considered, calls calc_lltf_per_rsd.m, and interprets and plots the output from calc_lltf_per_rsd.m.

Each mass distribution (determined my the centre of mass location) is passed, along with the other required vehicle parameters, to calc_lltf_per_rsd.m. The chassis stiffness value where slope of the LLTD vs RSD curve equals transferTarget is determined, and this along with the chassis index vs chassis stiffness is plotted in figure 1.

When the middle weight distribution value is reached, curves of lateral load distribution expressed as the percentage of the load transfer that occurs at the front axle vs the roll stiffness distribution (again expressed as percent front) are plotted. This is done for four chassis stiffness' in figure 2. This shows how LLTD changes with roll stiffness distribution and how this relationship changes with chassis stiffness. Next, curves of lateral load distribution expressed as the percentage of the load transfer that occurs at the front axle vs chassis stiffness are plotted. This is done for three roll stiffness distributions to show how chassis stiffness changes the lateral load distribution for a given roll stiffness distribution.

Last, legends and axis labels are added to the plots for completeness.

The output plots are presented and analyzed in section 4 and the script can be found in its entirety in Appendix B.

4 Conclusion

The scripts developed were run with weight distributions of 40, 50 and 60% rear; roll stiffness distributions ranging from 10-90% front, and chassis stiffness ranging from 1-6000 Nm/degree. Slope targets of 0.8, 0.85 and 0.9 were considered.

The first output considered was the LLTD vs RSD plot for a range of chassis stiffness'. This is shown in Figure 7. This plot demonstrates how the relationship between LLTD and RSD becomes increasingly linear with increasing chassis stiffness. The black dashed lines show linear curve fits to the portion of the data corresponding to roll stiffness distribution between 30 and 70% front. This region is defined to be the "region of interest" as reasonable suspension setups will fall within this range.

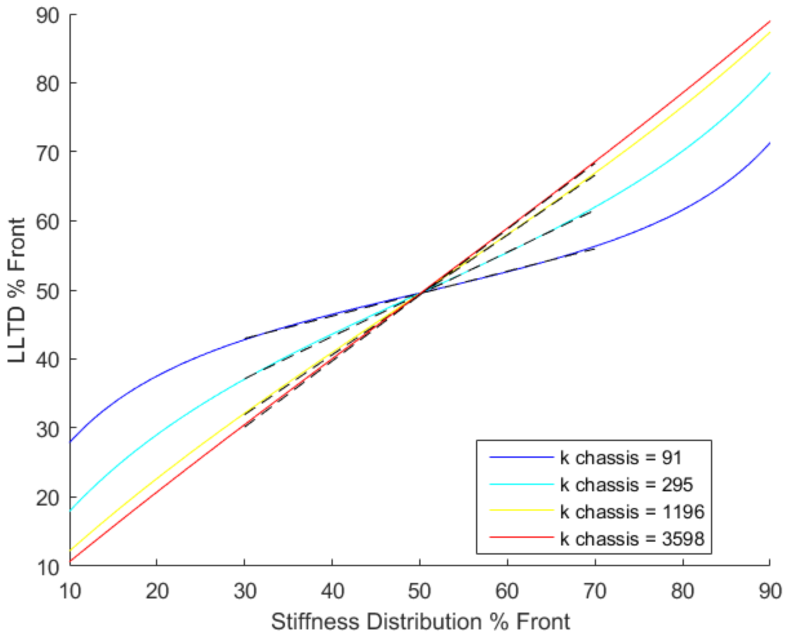

Figure 7: LLSD % front plotted versus RSD % Front for a range of chassis stiffness'

The slope of the linear fit in Figure 7 increases with increasing chassis stiffness. These slopes were used to generate the plot in Figure 8. To determine a chassis stiffness goal and to evaluate the 80% translation between RSD and LLTD suggested by Deakin et al., lines corresponding to slopes of 0.8, 0.85 and 0.9, and Milliken and Milliken's chassis stiffness suggestion of 3-5 times the suspension roll stiffness were also plotted.

To determine a stiffness goal, the desired suspension roll stiffness tuning range must be determined. The desired roll stiffness distribution is determined by the vehicles static mass distribution. From OptimumG's technical papers [5], the baseline roll stiffness distribution should be front biased by the static weight distribution plus 5% ($RSD_{\%front}$ =weight distribution$_{\%front} + 5$). Suspension is typically designed assuming a rigid chassis, so we can consider this to be equivalent to a desired lateral load transfer distribution. Historically, the static weight distribution of the Formula UBC cars has been in the range of 48-52% front. Based on this historic weight distribution variation and assuming suspension is designed for the midpoint 50-50 weight distribution, the lateral load transfer distribution should be able to be tuned by 5% (±2% from the design distribution) to account for deviation from even weight distribution between the front and rear of the vehicle. In the past, the Formula UBC suspension roll stiffness has been capable of tuning exactly 5%. For the 2017 car, an improvement in the amount of tuning possible was desired; the suspension team believes that the 2017 setup will allow roll stiffness tuning of 6%. In order for the effective tuning capabilities to reach at least the 5% goal, the chassis must be stiff enough to allow 83% of suspension tuning to translate into lateral load transfer distribution tuning.

This minimum slope of 0.83 is slightly higher than Deakin et al.'s suggestion of 0.8 and substantially less than Milliken and Milliken's suggestion (which corresponds to a slope between 0.93 and 0.96). Figure 8 shows that increasing chassis stiffness exhibits diminishing returns of $\frac{\partial LLTD}{\partial RSD}$. Increasing the proportion of roll stiffness distribution that results in lateral load transfer distribution from 0.8 to 0.85 requires a chassis stiffness increase of about 300 Nm/degree, while increasing it another 5% to 0.9 requires an additional 600 Nm/degree of chassis stiffness. Based on the determined minimum slope of 0.83 and the observed diminishing returns of increased chassis stiffness, a chassis should be designed to achieve a stiffness resulting in at least 85% of the suspension roll stiffness distribution correlating to lateral load transfer distribution.

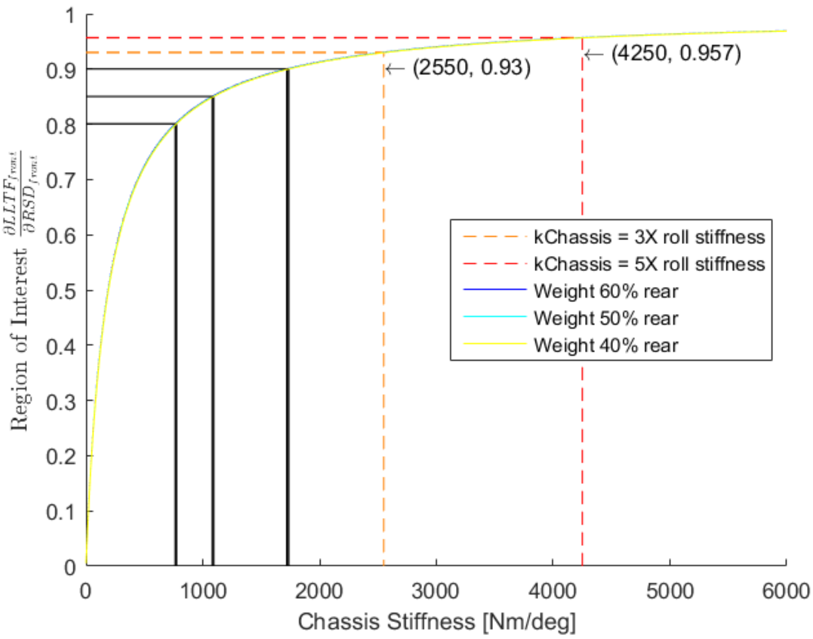

Figure 8: $\frac{\partial LLTD_{\%front}}{\partial RSD_{\%front}}$ plotted versus chassis stiffness'

It is also interesting to observe that the static weight distribution has very little effect on the relationship between chassis stiffness and the ability of lateral load distribution to be effected by suspension roll stiffness distribution. Figure 8 shows that over a 20% range of weight distributions, the curves of $\frac{\partial LLTD}{\partial RSD}$ are indistinguishable. The takeaway here is that, although absolute static weight distribution must be considered for suspension design, the deviation in weight distribution from the predicted value is much more important than the absolute weight distribution when setting a chassis torsional stiffness goal. Therefor, accurate prediction of the vehicle's static weight distribution is important both to allow suspension design optimization and to ensure that the designed chassis stiffness allows suspension tuning to be effective.

4.1 Application of Results

The results of this project were applied to the predicted 2017 Formula UBC car and compared to the predicted and measured stiffness of the 2016 car. The predicted vehicle parameters for the 2017 car are listed in Table 3. The 2016 car stiffness predicted by FEA was 1058 Nm/degree and the stiffness obtained through physical testing was 1039 Nm/degree.

Vehicle Property	Value [unit]
CG Height	0.31122 [m]
Wheel Radius	10 [in]
Front Roll Centre Height	1.906 [in]
Rear Roll Centre Height	2.234 [in]
Front Track Width	48 [in]
Rear Track Width	47 [in]
Wheel Base	60.5 [in]
Total Suspension Roll Stiffness	850 [Nm/degree]
Sprung Mass (including driver)	244 [kg]
Front Unsprung Mass	8 [kg]
Rear Unsprung Mass	7.5 [kg]
Static Weight Distribution	50 [% front]

Table 3: 2017 FUBC predicted vehicle parameters

The figures shown earlier in the document were generated using the values listed in Table 3. Using the method for setting a torsional stiffness goal developed through this project, the target stiffness for the 2017 chassis should be about 1100 Nm/degree. Comparing this value to the 2016 car's stiffness, this goal seems reasonable. Although the goal is slightly higher than last year's stiffness, it should be attainable; further, it validates the team's past "by feel" determination that the chassis stiffness was adequate. If the stiffness of the 2016 car was duplicated with the 2017 car's parameters, the model suggests that about 84.5% of roll stiffness distribution adjustments would translate into lateral load transfer distribution. This value is still above the determined minimum of 83.3%.

References

[1] C. Andersson Eurenius, N. Danielsson, A. Khokar, E. Krane, M. Olofsson and J. Wass. *Analysis of Composite Chassis*. Chalmers University of Technology, Göteborg, Sweden, 2013.

[2] A. Deakin, D. Crolla, J. P. Ramirez and R. Hanley. *The Effect of Chassis Stiffness on Race Car Handling Balance*. SAE Technical Paper, 2013.

[3] D. E. Malen. *Fundamentals of Automobile Body Structure Design*. SAE International, 2011.

[4] W. F. Milliken and D. L. Milliken. *Race Car Vehicle Dynamics*. SAE International, 1994.

[5] Matt Giaraffa. *Tech Tip: Springs & Dampers, Part Two: Attack of the Units* OptimumG. Retrieved November 13, 2016, from http://www.optimumg.com/technical/technical-papers.

Appendices

A Truss FEA Scripts

The truss FEA package created has two main sections:

1. Extracting 3D-sketch information from SolidWorks and parsing this data
2. Analyzing the structure in MATLAB.

The first section consists of a SolidWorks macro (genSketchInfoForMatlab.swp) that extracts the sketch information and a MATLAB function (parse_SWoutput.m) that converts the SolidWorks output file into the input format for the second section. After these have been run, the output node and tube files must be manually filled to specify fixtures, loads, and material properties. Once this is complete, the data is ready for analysis in the second section.

The second section consists of three classes (SpaceNode.m, SpaceTube.m and SpaceFrame.m). These classes represent nodes, tubes and the total space frame. The classes are extensively commented and contain documentation for each method.

A.1 genSketchInfoForMatlab.swp

```
Dim swApp As Object
Sub Main()
Dim swApp As SldWorks.SldWorks
Dim doc As SldWorks.ModelDoc2
Dim part As SldWorks.PartDoc
Dim sm As SldWorks.SelectionMgr
Dim feat As SldWorks.Feature
Dim sketch As SldWorks.sketch
Dim v As Variant
Dim i As Long
Dim sline As SldWorks.SketchLine
Dim sp As SldWorks.SketchPoint
Dim ep As SldWorks.SketchPoint
Dim s As String
Dim NumLines As Long

Set exApp = CreateObject("Excel.Application")
If Not exApp Is Nothing Then
    exApp.Visible = True
    If Not exApp Is Nothing Then
        exApp.Workbooks.Add
        Set sheet = exApp.ActiveSheet
        If Not sheet Is Nothing Then
            sheet.Cells(1, 1).Value = "Line"
            sheet.Cells(1, 2).Value = "Node1 X"
            sheet.Cells(1, 3).Value = "Node1 Y"
            sheet.Cells(1, 4).Value = "Node1 Z"
            sheet.Cells(1, 5).Value = "Node2 X"
            sheet.Cells(1, 6).Value = "Node2 Y"
            sheet.Cells(1, 7).Value = "Node2 Z"
        End If
    End If
End If


Set swApp = GetObject(, "sldworks.application")
If Not swApp Is Nothing Then
    Set doc = swApp.ActiveDoc
    If Not doc Is Nothing Then
        If doc.GetType = swDocPART Then
            Set part = doc
            Set sm = doc.SelectionManager
            If Not part Is Nothing And Not sm Is Nothing Then
```

```
                If sm.GetSelectedObjectType2(1) = swSelSKETCHES Then
                    Set feat = sm.GetSelectedObject4(1)
                    Set sketch = feat.GetSpecificFeature
                    If Not sketch Is Nothing Then
                        NumLines = sketch.GetLineCount2(1)
                        v = sketch.GetLines2(1)
                        For i = 0 To NumLines - 1
                            If Not sheet Is Nothing And Not exApp Is Nothing Then
                                sheet.Cells(2 + i, 1).Value = (i)
                                sheet.Cells(2 + i, 2).Value = (v(12 * i + 6))
                                sheet.Cells(2 + i, 3).Value = (v(12 * i + 7))
                                sheet.Cells(2 + i, 4).Value = (v(12 * i + 8))
                                sheet.Cells(2 + i, 5).Value = (v(12 * i + 9))
                                sheet.Cells(2 + i, 6).Value = (v(12 * i + 10))
                                sheet.Cells(2 + i, 7).Value = (v(12 * i + 11))
                                exApp.Columns.AutoFit
                            End If
                            Next i
                    End If
                End If
            End If
        End If
    End If
End If
End Sub
```

A.2 parse_SWoutput.m

```
function [nodes, tubes] = parse_SWoutput(fileName)
% parses data from solidworks macro to create spreadsheet for defining a
% space frame and its loads

    maxFileLines = 100000; % maximum number of file lines to scan

    file = fopen(fileName, 'r'); % open file
    fgets(file); % skip header line

    % tracks existing node positions
    nodePos = [];
    nodes = [];
    tubes = [];

    % search file, up to max 100000 lines
    for iii = 1 : maxFileLines
        line = fgets(file);
        if line == -1
            % found end of file
            break
        else
            % format of solidworks macro output is
            % [tube id, Node1 X, Node1 Y, Node1 Z, Node2 X, Node2 Y, Node2 Z]
            vars = textscan(line, '%f,%f,%f,%f,%f,%f,%f');
            tubeid = vars(1);
            tubeid = tubeid{1} + 1;
            node1 = vars(2 : 4);
            node1 = [node1{:}];
            node2 = vars(5 : 7);
            node2 = [node2{:}];
            if ~isempty(nodePos)
                ind = (nodePos(:, 1) == node1(1) & nodePos(:, 2) == node1(2) & ...
                    nodePos(:, 3) == node1(3));
                if max(ind) == 0
                    node1ID = length(nodes) + 1;
                    nodes = [nodes, SpaceNode(node1ID , node1, [0, 0, 0])];
                    nodePos = [nodePos; node1];
                else
                    node1ID = find(ind);
                end

```

```
                ind = (nodePos(:, 1) == node2(1) & nodePos(:, 2) == node2(2) & ...
                    nodePos(:, 3) == node2(3));
                if max(ind) == 0
                    node2ID = length(nodes) + 1;
                    nodes = [nodes, SpaceNode(node2ID , node2, [0, 0, 0])];
                    nodePos = [nodePos; node2];
                else
                    node2ID = find(ind);
                end
            else
                node1ID = 1;
                node2ID = 2;
                nodes = [SpaceNode(node1ID , node1, [0, 0, 0]), SpaceNode(node2ID , ...
                    node2, [0, 0, 0])];
                nodePos = [node1; node2];
            end

            tubes = [tubes, SpaceTube(tubeid, nodes(node1ID), nodes(node2ID), -1, -1)];
        end
    end

    fclose(file);
    fileName = strrep(fileName, '.csv', '');
    nodeFile = fopen([fileName, '_nodes.csv'], 'w');
    fprintf(nodeFile, ['ID, X (m), Y (m), Z (m), Load X (N), Load Y (N), Load Z (N), ...
        Fixture X (0/1), Fixture Y (0/1), Fixture Z (0/1)\n']);

    for iii = 1 : length(nodes)
        node = nodes(iii);
        fprintf(nodeFile, '%i, %f, %f, %f, , , , , , ,', node.id, ...
        node.position(1), node.position(2), node.position(3));
        fprintf(nodeFile, '\n');
    end
    fclose(nodeFile);

    tubeFile = fopen([fileName, '_tubes.csv'], 'w');
    fprintf(tubeFile, ['ID, Node1, Node2, diameter (in), thickness (in), E (Pa), ...
        sigma_y (Pa), sigma_u (Pa)\n']);
    for iii = 1 : length(tubes)
        tube = tubes(iii);
        fprintf(tubeFile, '%i, %i, %i, , , , , ,\n', tube.id, tube.node1.id, ...
            tube.node2.id);
    end
    fclose(tubeFile);

    frame = SpaceFrame();
    for iii = 1 : length(nodes)
        frame.addNode(nodes(iii));
    end

    for iii = 1 : length(tubes)
        frame.addTube(tubes(iii));
    end

    frame.plotFrame();
end
```

A.3 SpaceNode.m

```
classdef SpaceNode < handle
    % Represents a node in a 3D space frame
    properties
        id % number to identify the node
        position % [m] the nodes position in cartesian co-ordinate, as a vector
        load % [N] external force on the node as a vector
        fixtures % vector defining whether or not there are reaction forces 0 = free, ...
            1 = force
        reactions % [N] reaction force on the node as a vector
        tubes % list of tubes that end at the node
```

```
    end

    methods
        function obj = SpaceNode(id, position, fixtures)
        % constructs a node with the given ID and position
        % Parameters:
        %   -id (required): the id number for the node
        %   - position (required): [m] vector defining the cartesian
        %           co-ordiates of the node
            obj.id = id;
            obj.position = position;
            obj.load = [0, 0, 0];
            obj.reactions = [0, 0, 0];
            obj.fixtures = fixtures;
            obj.tubes = [];
        end

        function addLoad(obj, load)
            % Adds a load to the node
            % Parameters:
            %   - obj (required): the node to which the load will be added
            %   - load (required): [N] a vector defining the load to add to the
            %           node
            obj.load = obj.load + load;
        end
    end

end
```

A.4 SpaceTube.m

```
classdef SpaceTube < handle
    % Represents a tube in a 3D spaceframe

    properties
        id % number to identify the tube
        node1 % the node that defines one end of the tube
        node2 % the node that defines the second end of the tube
        diameter % [m] outside diameter of the tube
        thickness % [m] wall thickness of the tube
        E % [Pa] young's modulus for the tube material
        sigma_y % [Pa] yeild strength for the tube material
        sigma_u % [Pa] ultimate strength for the tube material
        unitVector % unit vector along the tube, pointing from node1 to node 2
        length % [m] length of the tube
        tubeVector % [m] vector with the magnitude and direction of the tube
        force % [N] the force acting on the tube
        stress % [Pa] the axial stress on the tube
    end

    methods
        function obj = SpaceTube(id, node1, node2, diameter, thickness, varargin)
        % Creates a tube as a straight line between the specified nodes and
        % with the specified geometric and material properties
        % Parameters:
        %   -id (required): a number to identify the node
        %   -node1 (required): the node where the tube starts
        %   -node2 (required): the node where the tube ends
        %   -diameter (required): the diameter of the tube in inches
        %   -thickness (required): the wall thickness of the tube in inches
        %   -E (optional name-value pair): the youngs modulus of the tube
        %           in Pascals. Corresponds to 4130 steel by default
        %   -sigma_y (optional name-value pair): yeild strength of the tube
        %           material in pascals. Corresponds to 4130 steel by default.
        %   -sigma_u (optional name-value pair): ultimate strength of the tube
        %           material in pascals.. Corresponds to 4130 steel by
        %           default.
            p = inputParser;
            default_E = 100e9; % [Pa] TODO update to real value
```

```
            default_Sy = 100e6; % [Pa] TODO update to real value
            default_Su = 100e6; % [Pa] TODO update to real value
            addRequired(p, 'id', @isnumeric);
            addRequired(p, 'node1', @(x) isa(x, 'SpaceNode'));
            addRequired(p, 'node2', @(x) isa(x, 'SpaceNode'));
            addRequired(p, 'diameter', @isnumeric);
            addRequired(p, 'thickness', @isnumeric);
            addParameter(p, 'E', default_E, @isnumeric);
            addParameter(p, 'sigma_y', default_Sy, @isnumeric);
            addParameter(p, 'sigma_u', default_Su, @isnumeric);
            parse(p, id, node1, node2, diameter, thickness, varargin{:});

            % if inputs were valid, initialize tube properties
            obj.id = p.Results.id;
            obj.node1 = p.Results.node1;
            obj.node2 = p.Results.node2;
            obj.diameter = p.Results.diameter * 25.4 / 1000;
            obj.thickness = p.Results.thickness * 25.4 / 1000;
            obj.E = p.Results.E;
            obj.sigma_y = p.Results.sigma_y;
            obj.sigma_u = p.Results.sigma_u;
            obj.tubeVector = obj.node2.position - obj.node1.position;
            obj.length = norm(obj.tubeVector);
            obj.unitVector = obj.tubeVector./obj.length;

            obj.node1.tubes = [obj.node1.tubes, obj];
            obj.node2.tubes = [obj.node2.tubes, obj];
        end

        function calculateStress(obj)
        % calculates the stress in a tube based on its diameter, area,
        % thickness, and force applied
            area = (pi / 4) * (obj.diameter^2 - (obj.diameter - ...
            obj.thickness) ^ 2); % [m^2]
            obj.stress = obj.force / area; % [Pa]
        end
    end
end
```

A.5 SpaceFrame.m

```
% Represents a 3D space frame structure
classdef SpaceFrame < handle

    % Method comments duplicated so they are visible both when functions are
    % collapsed and when viewing class documentation via typing doc SpaceFrame
    % in the MATLAB console

    properties
    nodes % an array of nodes in the space frame
    tubes % an array of tubes in the space frame
    confinedNodes % list of nodes that are not free
    numReactionForces % list corresponding to confinedNodes with the number of reation ...
        forces at each node
    solved % boolean, true if the space frame has been solved
    maxStress % [Pa] max stress in the frame after solving
    minStress % [Pa] in stress in the frame after solving
    end

    methods
        function obj = SpaceFrame()
        % creates a new space frame with no nodes or tubes
            obj.nodes = [];
            obj.tubes = [];
            obj.confinedNodes = [];
            obj.solved = false;
            obj.maxStress = 0;
            obj.minStress = 0;
        end
```

```
        function addNode(obj, node)
        % adds a node to the spaceframe
        % Parameters:
        %   - obj: the space frame object to which the node will be added
        %   - node: the node object to add
            if isa(node, 'SpaceNode')
                obj.nodes = [obj.nodes, node];

                if max(node.fixtures) ~= 0
                    obj.confinedNodes = [obj.confinedNodes, node];
                    obj.numReactionForces = [obj.numReactionForces, nnz(node.fixtures)];
                end
            else
                error('Only nodes can be added as a node')
            end
        end

        function createTube(obj, node1, node2, diameter, thickness)
        % creates a 4130 steel tube between node1 and node2
        % Parameters:
        %   - obj: the spaceframe in which the tube will be created
        %   - node1: the node object where the tube starts
        %   - node2: the node object where the tube ends
        %   - diameter: the tube diameter in inches
        %   - thickness: the tube thickness in inches
            id = length(obj.tubes);
            for iii = 1 : length(obj.tubes)
                tube = obj.tubes(iii);
                if tube.id > id
                    id = tube.id + 1;
                end
            end

            obj.tubes = [obj.tubes, SpaceTube(id, node1, node2, diameter, thickness)];
        end

        function addTube(obj, tube)
        % adds the specified tube to the space frame
        % Parameters:
        %   - obj: the space frame to which the tube will be added
        %   - tube: the spaceTube object to add
            if isa(tube, 'SpaceTube')
                obj.tubes = [obj.tubes, tube];
            else
                error('Only tubes can be added as a tube')
            end
        end

        function plotFrame(obj)
        % plots the space frame in a figure. If the frame is solved, tubes
        % are color coded by stress
            figure;
            hold on
            xlabel('X [m]');
            ylabel('Y [m]')
            zlabel('Z [m]')
            plottedNodeIds = [];

            if obj.solved
                colors = jet(1001);
                colormap jet
                bar = colorbar('eastoutside');
                set(bar, 'TickLabels', linspace(obj.minStress, obj.maxStress, 11))
                ylabel(bar, 'Stress [Pa]');
            end

            nodeX = zeros(1, length(obj.nodes));
            nodeY = zeros(1, length(obj.nodes));
            nodeZ = zeros(1, length(obj.nodes));
            nodeLabels = cell(1, length(obj.nodes));
            nodeInd = 1;

```

```
            for iii = 1 : length(obj.tubes)
                tube = obj.tubes(iii);
                x = zeros(1, 2);
                y = zeros(1, 2);
                z = zeros(1, 2);

                % plot node1 if not plotted
                if ~any(tube.node1.id==plottedNodeIds)
                    plottedNodeIds = [plottedNodeIds, tube.node1.id];
                    nodeX(nodeInd) = tube.node1.position(1);
                    nodeY(nodeInd) = tube.node1.position(2);
                    nodeZ(nodeInd) = tube.node1.position(3);
                    nodeLabels(nodeInd) = {num2str(tube.node1.id)};
                    nodeInd = nodeInd + 1;
                end

                % plot node2 if not plotted
                if ~any(tube.node2.id==plottedNodeIds)
                    plottedNodeIds = [plottedNodeIds, tube.node1.id];
                    nodeX(nodeInd) = tube.node2.position(1);
                    nodeY(nodeInd) = tube.node2.position(2);
                    nodeZ(nodeInd) = tube.node2.position(3);
                    nodeLabels(nodeInd) = {num2str(tube.node2.id)};
                    nodeInd = nodeInd + 1;
                end
                x(1) = tube.node1.position(1);
                y(1) = tube.node1.position(2);
                z(1) = tube.node1.position(3);
                x(2) = tube.node2.position(1);
                y(2) = tube.node2.position(2);
                z(2) = tube.node2.position(3);
                text((x(2) + x(1)) / 2, (y(2) + y(1)) / 2,  (z(2) + z(1)) / 2, ...
                    num2str(tube.id))

                if ~obj.solved
                    plot3(x,y,z, 'b')
                else
                    plot3(x, y, z, 'color', colors(floor(abs((tube.stress - ...
                        obj.minStress) / (obj.maxStress - obj.minStress) * 1000)) + 1, :))
                end
            end

            plot3(nodeX, nodeY, nodeZ, 'bo')
            text(nodeX, nodeY, nodeZ, nodeLabels)
        end

        function [A, x, y] = solveFrame(obj)
        % creates the matrix defining the forces in the spaceframe and
        % solves. System of equations is defined as A * x = y where x is
        % the force on each tube or a reaction force and y is the applied
        % loads. The order of x is <tube forces>, <reaction forces> where
        % the tube forces are in the order they occur in obj.tubes and
        % reaction forces are in the order they occur in obj.confinedNodes.

            % determine number of constraints
            constraints = 0;
            for iii = 1 : length(obj.confinedNodes)
                node =  obj.confinedNodes(iii);
                constraints = constraints + sum(node.fixtures);
            end

            % if there are less than 6 contraints, the frame is
            % underdefined and cannot be solved
            if constraints < 6
                error('Space frame is underconstrained and cannot be solved')
            end

            % check if the model is statically determinate (number of
            % unknowns <= number of equations
            if constraints + length(obj.tubes) > 3 * length(obj.nodes)
                error('Space frame is statically indeterminate and cannot be solved')
            end

```

```
        % System is described by
        %   A * x = y
        % where A is a matrix, x is a vector of tube forces and
        % reaction forces, and y is a vector of constants defined by
        % the geometry
        % matrix defining the system
        A = zeros(3 * length(obj.nodes), ...
        constraints + length(obj.tubes));

        y = zeros(3 * length(obj.nodes), 1);
        % the equation currently being created
        equation = 1;
        for iii = 1 : length(obj.nodes)
            node = obj.nodes(iii);
            for jjj = 1 : length(node.tubes)
                tube = node.tubes(jjj);
                direction = tube.unitVector; % default points from node1 to node2
                % ensure vector points away from the node (assume tubes
                % are in tension)
                if node == tube.node1
                    % switch direction so it points from node2 to node1
                    direction = -1 .* direction;
                end

                index = find(tube == obj.tubes);
                % node forces in x sum to zero
                A(equation, index) = direction(1);
                % node forces in y sum to zero
                A(equation + 1, index) = direction(2);
                % node forces in z sum to zero
                A(equation + 2, index) = direction(3);
            end

            index = find(node == obj.confinedNodes, 1);
            % if this node has reaction forces, add them to the matrix
            if ~isempty(index)
                % find the matrix column corresponding to the node's
                % reaction forces
                matIndex = length(obj.tubes) + 1;

                for jjj = 1 : index - 1
                    matIndex = matIndex + obj.numReactionForces(jjj);
                end

                for jjj = 1 : 3
                    % if the node has a reaction force
                    if node.fixtures(jjj) ~= 0
                        % add it to the matrix
                        A(equation + jjj - 1, matIndex) = 1;
                        matIndex = matIndex + 1;
                    end
                end
            end

            % add node loads to the vector y
            y(equation) = node.load(1);
            y(equation + 1) = node.load(2);
            y(equation + 2) = node.load(3);

            equation = equation + 3;
        end

        % solve the system of equations
        solution = rref([A, y]);
        x = solution(:, end);

        for iii = 1 : length(x)
            if x(iii) ~= 0 && nnz(solution(iii, 1 : end - 1)) == 0
                error(['Cannot be solved. The model is unstable or one or more ...
                    members or under bending.'])
            end
        end
```

```
            % assign results to tubes and calculate max/min stress
            max = 0;
            min = 0;
            for iii = 1 : length(obj.tubes)
                tube = obj.tubes(iii);
                tube.force = x(iii);
                tube.calculateStress();
                if tube.stress > max
                    max = tube.stress;
                elseif tube.stress < min
                    min = tube.stress;
                end
            end
            obj.maxStress = max;
            obj.minStress = min;

            % assign reaction forces to nodes
            iii = iii + 1;
            for jjj = 1 : length(obj.confinedNodes)
                node = obj.confinedNodes(jjj);
                node.reactions(node.fixtures == 1) = x(iii : iii + ...
                sum(node.fixtures) - 1);
                iii = iii + sum(node.fixtures);
            end

            obj.solved = true;
            % plot with stress
            obj.plotFrame();
        end

        function K = calcTorsionalStiffness(obj, torque)
        % calculates stiffness using energy methods, sum(strain energy in
        % tube) = work done by torque.
        % Parameters:
        %   - obj: the space frame object
        %   - torque: the torque applied to the model
        % Returs:
        %   - K: stiffness of the space frame
            energy = 0; % [J] strain energy in tubes
            for i = 1 : length(obj.tubes)
                tube = obj.tubes(i);
                % strain energy = 1/(2E) * stress^2 * pi * area * length
                energy = energy + 1 / (2 * tube.E) * (tube.stress) ^ 2 * ...
                pi * ((tube.diameter / 2)^2 - (tube.diameter / 2 - ...
                tube.thickness) ^2) * tube.length;
            end

            % work = strain energy => T * theta / 2 = strain energy
            theta = (2 * energy / torque) * (180 / pi); % [degrees] angular deflection
            K = torque / theta; % [Nm/deg] stiffness
        end
    end

    methods(Static)
        function obj = spaceFrameFromFiles(nodeFile, tubeFile)
        % creates a space frame object from csv files containing node and
        % tube information. csv files should be generated using the
        % following process:
        %   1. Make a 3D sketch in solidworks
        %   2. select the sketch and run the attached solidworks macro to
        %      generate an excel file with raw data
        %   3. Save the excel file as a csv
        %   4. Run the attached matlab function parse_SWoutput.m on the csv
        %      generated by the macro to generate the proper files for this
        %      package
        %   5. Fill in the node load and fixture information, and tube
        %      property information in the generated node and tube csv
        %      files.
        %   6. Pass the populated node and tube csv files to this function
        %      to create a spaceFrame for analysis
        % Parameters:
        %   - nodeFile: The csv containing node information generated by
        %               following the above process
```

```
        %   - tubeFile: the csv containing tube information generated by
        %               following the above process
        % Returns:
        %   - obj: a SpaceFrame object created from the specified files
            maxFileLines = 100000; % maximum number of file lines to scan
            obj = SpaceFrame(); % create empty frame to populate
            nodeFile = fopen(nodeFile); % open file with node info
            fgets(nodeFile); %skip header line

            for iii = 1 : maxFileLines
                line = fgets(nodeFile);
                if line == -1
                    % found end of file
                    break
                else
                    % format of solidworks macro output is
                    % [ID, X, Y, Z, Load X, Load Y, Load Z, Fixture X,
                    %  Fixture Y, Fixture Z, Tubes
                    vars = textscan(line, '%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%s');
                    id = vars{1};
                    position = [vars{2:4}];
                    load = [vars{5:7}];
                    fixtures = [vars{8:10}];
                    node = SpaceNode(id, position, fixtures);
                    node.addLoad(load);
                    obj.addNode(node);
                end
            end

            fclose(nodeFile); % close node file

            tubeFile = fopen(tubeFile); % open file with tube info
            fgets(tubeFile); %skip header line

            for iii = 1 : maxFileLines
                line = fgets(tubeFile);
                if line == -1
                    % found end of file
                    break
                else
                    % format of solidworks macro output is
                    % [ID, Node1, Node2, diameter, thickness, E, sigma_y,
                    % sigma_u]
                    vars = textscan(line, '%f,%f,%f,%f,%f,%f,%f,%f');
                    id = vars{1};
                    node1 = obj.nodes(vars{2});
                    node2 = obj.nodes(vars{3});
                    diameter = vars{4};
                    thickness = vars{5};
                    E = vars{6};
                    sigma_y = vars{7};
                    sigma_u = vars{8};
                    tube = SpaceTube(id, node1, node2, diameter, thickness, ...
                    'E', E, 'sigma_y', sigma_y, 'sigma_u', sigma_u);
                    obj.addTube(tube);
                end
            end

            fclose(tubeFile); % close node file
        end
    end
end
```

B LLTD Scripts

The model described in section 3.1 was used to create a MATLAB script that, given vehicle parameters, could calculate the lateral toad transfer and proportion of roll stiffness distribution that translates into lateral load transfer distribution for a range of weight distributions, suspension roll stiffness distributions, and chassis stiffness's.

The script was broken into two parts:

1. A function calc_lltd_per_rsd.m: given vehicle properties and ranges of chassis stiffness and roll stiffness distribution to consider, calculates the index presented at the end of section 3.1 and both the front and rear load transferred for a range of roll stiffness distributions and chassis stiffness's.
2. A script LLTD.m: contains vehicle parameters and organizes the calling of calc_lltd_per_rsd.m and the plotting of its results.

B.1 calc_lltd_per_rsd.m

```
function [k_chassis, dLLTD_by_dRSD, LLT_front, LLT_rear] =  calc_lltd_per_rsd( ...
    mSprung, mUnsprung_front, mUnsprung_rear, hCG, dist_cg_from_rear, rWheel_front, ...
    rWheel_rear, hRC_front, hRC_rear, trackWidth_front, trackWidth_rear, wheelBase, ...
    kRoll, kRoll_pctFrontMin, kRoll_pctFrontMax, kChassis_min, kChassis_max, latAcc)
% Calculates the lateral load distribution difference per roll
% stiffness distribution difference as a function of chassis stiffness.
% Parameters :
%   - mSprung : [kg] Sprung vehicle mass, including driver.
%   - mUnsprung_front : [kg] unsprung mass of front components
%   - mUnsprung_rear : [kg] unsprung mass of rear components
%   - hCG : [m] height of CG above the ground
%   - dist_cg_from_rear : [m] longitudinal distance from the rear axle to
%                           the CG.
%   - rWheel_front : [m] radius of the front wheels
%   - rWheel_rear : [m] radius of the rear wheels
%   - hRC_front : [m] height of the front roll center above the ground
%   - hRC_rear : [m] height of the rear roll center above the ground
%   - trackWidth_front : [m] track width at the front wheels
%   - trackWidth_rear : [m] track width at the rear wheels
%   - wheelBase : [m] wheel base of the vehicle
%   - kRoll : [Nm/deg] total roll stiffness (sum of front and rear roll
%             stiffness
%   - kRoll_pctFrontMin: [unitless] the minimum fraction of total roll
%                        stiffness to consider as front roll stiffness
%   - kRoll_pctFrontMax: [unitless] the maximum fraction of total roll
%                        stiffness to consider as front roll stiffness
%   - kChassis_min : [Nm/deg] minimum chassis stiffness to consider
%   - kChassis_max : [Nm/deg] maximum chassis stiffness to consider
%   - latAcc : [m/s^2] lateral acceleration to use for calculations
% Returns :
%   - k_chassis : [Nm/deg] vector for range of chassis stiffnesses considered
%   - dLLTD_by_dRSD :   |LLTD_r-LLTD_f| / |RSD_r - RSD_f|, where RSD is
%                       roll stiffness distribution as a percent and LLTD
%                       is lateral load transfer distribution as a percent
%                       averaged over RSD = 10-30% front and 70-90%
%                       front. This ignores the near linear portion of
%                       graph and the portion where |RSD_r - RSD_f| = 0
%   - LLT_front : [N] Load transferred from the front inside wheel to the
%               front outside wheel. Rows vary roll stiffness distribution
%               from 10-90% front and columns vary chassis stiffness from
%               kChassis_min to kChassis_max

%   - LLT_rear : [N] Load transferred from the rear inside wheel to the
%               rear outside wheel. Rows vary roll stiffness distribution
%               from 10-90% front and columns vary chassis stiffness from
%               kChassis_min to kChassis_max

% Based on models from:
%   Chalmers University Paper: ...
    http://publications.lib.chalmers.se/records/fulltext/191830/191830.pdf
%   SAE Paper : The Effect of Chassis Stiffness on Race Car Handling Balance. Deakin ...
    et. al.

kChassis_dataPoints = 1000; % number of points within the chassis stiffness range ...
    provided to use
kRoll_dataPoints = 160; % number of points to use for front and rear roll stiffness

g = 9.81; % [m/s^2] acceleration due to gravity
```

```
k_front = kRoll.*linspace(kRoll_pctFrontMin, kRoll_pctFrontMax,...
kRoll_dataPoints); % front suspension stiffness range [Nm/deg]
k_rear = kRoll.*linspace(1 - kRoll_pctFrontMin, 1 - kRoll_pctFrontMax,...
kRoll_dataPoints); % rear suspension stiffness range [Nm/deg]
k_chassis = linspace(kChassis_min, kChassis_max, kChassis_dataPoints); % range of ...
    chassis stiffnesses to consider [Nm/deg]
b = dist_cg_from_rear; % longitudinal distance from rear axle to CG [m]
a = wheelBase - b; % longitudinal position from front axel to CG [m]
x = hCG-(a*hRC_rear+b*hRC_front)/(wheelBase); % CG's Height over roll centre axis [m]

M_cg = mSprung*latAcc*x; % Momentum on roll axis from sprung mass [Nm]
% Moment from unsprung masses on front and rear axle
MUnsprung_front = latAcc*(rWheel_front-hRC_front)*mUnsprung_front; % [Nm]
MUnsprung_rear = latAcc*(rWheel_rear-hRC_rear)*mUnsprung_rear; % [Nm]

% Total cornering moment about roll axis, assume chassis stiffness is
% uniformly distributed and that the location of the CG will determine
% how much of the CG moment is transfered to the front and rear axle
M_front_total = MUnsprung_front + (b / (a + b)) * M_cg; % [Nm]
M_rear_total = MUnsprung_rear + (a / (a + b)) * M_cg; % [Nm]

LLT_front = zeros(kRoll_dataPoints, kChassis_dataPoints);
LLT_rear = zeros(kRoll_dataPoints, kChassis_dataPoints);
for iii = 1 : length(k_chassis)
    for jjj = 1 : length(k_front)
        %% Equation system
        % matrix relating angular deflection of suspension and chassis to
        % moments and continuity equation
        A = [k_front(jjj) 0 -k_chassis(iii);
            0 k_rear(jjj) k_chassis(iii);
            1 -1 1];
        M_array = [M_front_total; M_rear_total; 0];

        angles_of_rotation = A \ M_array; % angles of rotation (front, rear, chassis) ...
            [deg]
        LLT_front(jjj, iii) = 1 / trackWidth_front * (k_chassis(iii) * ...
            angles_of_rotation(3) + M_front_total); % [N] Front left vertical wheel ...
            force - Front right vertical wheel force
        LLT_rear(jjj, iii) = 1 / trackWidth_rear * (-k_chassis(iii) * ...
            angles_of_rotation(3) + M_rear_total); % [N] Rear left vertical wheel ...
            force - Rear right vertical wheel force
    end
end

frontLoadDist = LLT_front./(LLT_front + LLT_rear);
frontRollStiffnessDist = k_front ./ (k_front + k_rear);

dLLTD_by_dRSD = zeros(1, length(k_chassis));
for iii = 1 : length(k_chassis)
    y = frontLoadDist(kRoll_dataPoints / 4 : 3 * kRoll_dataPoints / 4, iii)';
    x = frontRollStiffnessDist(kRoll_dataPoints / 4 : 3 * kRoll_dataPoints / 4);
    fit = polyfit(x, y, 1);
    dLLTD_by_dRSD(iii) = fit(1);
end
end
```

B.2 LLTD.m

```
clear all; close all
%% Lateral load transfer distribution for varying torsional stiffness
% Based on models from:
%   Chalmers University Paper: ...
    http://publications.lib.chalmers.se/records/fulltext/191830/191830.pdf
%   SAE Paper : The Effect of Chassis Stiffness on Race Car Handling Balance. Deakin ...
    et. al.

%% Lateral accelleration
latAcc = 2 * 9.81; %[m/s^2]
%% Car input static data [m]
```

```
% Car dimensions
hCG = 0.31122; % CG's height over ground contact line
rWheel_front = 9 * 25.4 / 1000; % Front wheel radius
rWheel_rear = 9 * 25.4 / 1000; % Rear wheel radius
hRC_front = 1.906 * 25.4 / 1000; % Front wheel roll centre height
hRC_rear = 2.234 * 25.4 / 1000; % Rear wheel roll centre height
trackWidth_front = 48 * 25.4 / 1000; % Track width
trackWidth_rear = 47 * 25.4 / 1000; % Track width
wheelBase = 60.5 * 25.4 / 1000; %[m]
rollStiffness =  850; %[Nm/deg]
% Masses
mSprung = 244; % Sprung mass, including driver [kg]
mUnsprung_front = 8; % Front unsprung mass [kg]
mUnsprung_rear = 7.5; % Rear unsprung mass [kg]
%% weight distributions to consider
b = wheelBase.*[0.6, 0.5, 0.4]; % CG's longitudal position from rear axle
% target percent of roll stiffness distribution difference that translates
% into lateral load transfer distribution difference
tranferTarget = [0.8, 0.85, 0.9];
% set up colormap
col=jet(length(b) + 1);

% Generate curves for varrying dLLTD/dRSD vs k_chassis vs weight distribution
figure(1)
hold on

hInd = 1;
h = zeros(1,5);
hh = zeros(1,4);
for iii = 1 : length(b)
    [kChassis, dLLTD_by_dRSD, LLT_front, LLT_rear] = calc_lltd_per_rsd( mSprung, ...
        mUnsprung_front, mUnsprung_rear, hCG, b(iii), rWheel_front, rWheel_rear, ...
        hRC_front, hRC_rear, trackWidth_front, trackWidth_rear, wheelBase, ...
        rollStiffness, 0.1, 0.9, 1, 6000, latAcc);
    % diff_LLTD_per_RSD vs kChassis for this longitudinal position and a
    % line picking out the required chassis stiffness for 90% of
    % rsd to translate to lltd
    figure(1)
    for jjj = 1 : length(tranferTarget)
        ind = dLLTD_by_dRSD - tranferTarget(jjj) > 0;
        ind = find(ind, 1, 'first');
        plot([0, kChassis(ind), kChassis(ind)], [dLLTD_by_dRSD(ind), ...
            dLLTD_by_dRSD(ind), 0], 'k')
    end
    rsX3 = rollStiffness * 3;
    rsX5 = rollStiffness * 5;
    index_rsX3 = interp1(kChassis, dLLTD_by_dRSD, rsX3);
    index_rsX5 = interp1(kChassis, dLLTD_by_dRSD, rsX5);

    if iii == 1
        h(hInd) = plot([0, rsX3, rsX3], [index_rsX3, index_rsX3, 0], '--', 'color', ...
            [1, 0.5, 0]);
        h(hInd + 1) = plot([0, rsX5, rsX5], [index_rsX5, index_rsX5, 0], 'r--');
        hInd = hInd + 2;
        text(rsX3, index_rsX3 - 0.025, ['\leftarrow (', num2str(rsX3), ', ', ...
            num2str(round(index_rsX3, 3)), ')'])
        text(rsX5, index_rsX5 - 0.025, ['\leftarrow (', num2str(rsX5), ', ', ...
            num2str(round(index_rsX5,3)), ')'])
    else
        plot([0, rsX3, rsX3], [index_rsX3, index_rsX3, 0], '--', 'color', [1, 0.5, 0]);
        plot([0, rsX5, rsX5], [index_rsX5, index_rsX5, 0], 'r--');
    end

    h(hInd) = plot(kChassis, dLLTD_by_dRSD, 'color',col(iii,:));
    hInd = hInd + 1;

    if iii == round(length(b) / 2)
        % plot LLTD % front VS RSD % front for a variety of chassis
        % stiffnesses
        figure(2)
        hold on
        lower = length(LLT_front(:, 1)) / 4;
        upper = length(LLT_front(:, 1)) * 3 / 4;
```

```
        F_pctFront = LLT_front ./ (LLT_front + LLT_rear) * 100;
        hh(1) = plot(linspace(10, 90, length(LLT_front(:, 1))), F_pctFront(:, 16), ...
            'color', col(1, :));
        fit = polyfit(linspace(30, 70, length(LLT_front(:, 1)) / 2 + 1), ...
            F_pctFront(lower : upper, 16)', 1);
        plot(linspace(30, 70, length(LLT_front(:, 1))), polyval(fit, linspace(30, 70, ...
            length(LLT_front(:, 1)))), 'k--');

        hh(2) = plot(linspace(10, 90, length(LLT_front(:, 1))), F_pctFront(:, 50), ...
            'color', col(2, :));
        fit = polyfit(linspace(30, 70, length(LLT_front(:, 1)) / 2 + 1), ...
            F_pctFront(lower : upper, 50)', 1);
        plot(linspace(30, 70, length(LLT_front(:, 1))), polyval(fit, linspace(30, 70, ...
            length(LLT_front(:, 1)))), 'k--');

        hh(3) = plot(linspace(10, 90, length(LLT_front(:, 1))), F_pctFront(:, 200), ...
            'color', col(3,:));
        fit = polyfit(linspace(30, 70, length(LLT_front(:, 1)) / 2 + 1), ...
            F_pctFront(lower : upper, 200)', 1);
        plot(linspace(30, 70, length(LLT_front(:, 1))), polyval(fit, linspace(30, 70, ...
            length(LLT_front(:, 1)))), 'k--');

        hh(4) = plot(linspace(10, 90, length(LLT_front(:, 1))), F_pctFront(:, 600), ...
            'color', col(4, :));
        fit = polyfit(linspace(30, 70, length(LLT_front(:, 1)) / 2 + 1), ...
            F_pctFront(lower : upper, 600)', 1);
        plot(linspace(30, 70, length(LLT_front(:, 1))), polyval(fit, linspace(30, 70, ...
            length(LLT_front(:, 1)))), 'k--');

        figure(3)
        hold on
        xDim = size(LLT_front, 1);
        plot(linspace(1, 6000, length(LLT_front(1, :))), F_pctFront(xDim / 4, :), ...
            'color', col(1, :))
        plot(linspace(1, 6000, length(LLT_front(1, :))), F_pctFront(xDim / 2, :), ...
            'color', col(2, :))
        plot(linspace(1, 6000, length(LLT_front(1, :))), F_pctFront(3 * xDim / 4, :), ...
            'color', col(3,:))
    end
end

figure(1)
xlabel('Chassis Stiffness [Nm/deg]')
ylabel('Region of Interest $\frac{\partial LLTF_{front}}{\partial RSD_{front}}$', ...
    'Interpreter', 'Latex')
legend(h, {'kChassis = 3X roll stiffness', 'kChassis = 5X roll stiffness', ['Weight ' ...
    num2str(b(1) / wheelBase * 100) '% rear'], ['Weight ' num2str(b(2) / wheelBase * ...
    100) '% rear'], ['Weight ' num2str(b(3) / wheelBase * 100) '% rear']}, 'Location', ...
    'best')

figure(2)
legend(hh, {['k chassis = ' num2str(round(kChassis(16)))], ['k chassis = ' ...
    num2str(round(kChassis(50)))], ['k chassis = ' num2str(round(kChassis(200)))], ['k ...
    chassis = ' num2str(round(kChassis(600)))]}, 'Location', 'best');
xlabel('Stiffness Distribution % Front')
ylabel('LLTD % Front')

figure(3)
legend({'RSD % Front = 30', 'RSD % Front = 50', 'RSD % Front = 70'}, 'Location', 'best');
xlabel('Chassis Stiffness [Nm/deg]')
ylabel('LLTD % Front')
```