INFLUENCE OF GEOMETRY AND PLACEMENT CONFIGURATION ON SIDE FORCES IN COMPRESSION SPRINGS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Rahul Deshmukh

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Arun Prakash, Co-Chair

Lyles School of Civil Engineering

Dr. Jitesh Panchal, Co-Chair

School of Mechanical Engineering

Dr. Marisol Koslowski

School of Mechanical Engineering

Approved by:

Dr. Jay P. Gore

Head of the School Graduate Program

To a lifetime of learning and experimenting ....

and to my family and friends

ACKNOWLEDGMENTS

First, I would like to thank Professor Arun Prakash for being an incredible advisor to me and for teaching me mechanics. I am indebted to him for his guidance and encouragement through tough times. He has helped me not only shape my career and grow professionally, but has also had an indelible effect on my personality.

I would also like to thank the distinguished members of my examination committee, Professor Jitesh Panchal and Professor Marisol Koslowski for their time and effort in going through my work and for their valuable suggestions.

I am grateful for the generous support I have received from Indiana Next Generation Manufacturing Competitiveness Center - IN-MaC through industrial projects throughout my graduate studies at Purdue. The projects truly served the dual purpose of providing the support and presenting an opportunity to work on challenging and exciting projects.

I would specially like to thank MW Industries, Inc. for presenting with the problem which formed the bulk of work presented here. I thank Mike Kime, John Conn and Richard Thoden for their friendly co-operation and for sharing their testing facilities throughout the project.

I would also like to thank Professor Ayman Habib for connecting me with Agronomy Center for Research and Education (ACRE) which helped in obtaining 3D scanning equipment. I thank Jason Adams for loaning the equipment for extended duration without any hassles.

I cannot thank my family enough for their constant love and support. I thank my parents, Keshav Deshmukh and Suman Deshmukh, who always put the good of their children ahead of their own and provided us with the best education and morals. Love for my little sister, Prerna, for taking care of mom and dad while I am here. I would also like to thank my cousin sister, Manisha Deshmukh, for her constant

support which helped me in getting accustomed to a new country and for providing me the comfort and joy of a family away from home.

Finally, I would thank all my colleagues and close friends, especially Hugo Raul Esquivel Otreo, Mriganabh Boruah, Harsh Bohra, Tarutal Ghosh Mondal, Sukirt Thakur, Rashika Madan, Sumit Gupta and Dashpinder Singh for being part of this wonderful journey.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

SYMBOLS

u	Displacement
μ	Coefficient of friction
d	Wire diameter
D	Retainer diameter
d_p	Plate diameter
t	Plate thickness
H	Spring height
E	Young's modulus
K	Stiffness of spring-dashpot element
K_a	Axial spring stiffness
K_c	Radial spring stiffness in compression
K_t	Radial spring stiffness in tension

ABBREVIATIONS

3-D	Three Dimensional
CAD	Computer Aided Design
FEM	Finite Element Model
FEA	Finite Element Analysis
dofs/dof	Degree of freedom
BC	Boundary Condition
C3D8	Continuum 3D 8-noded element
C3D8I	Continuum 3D 8-noded element with Incompatible modes
B32	3D quadratic Timoshenko Beam element
UTM	Universal Testing Machine
DOE	Design of Experiments
LASSO	Least Absolute Shrinkage and Selection Operator

ABSTRACT

Deshmukh, Rahul M.S., Purdue University, December 2019. Influence of Geometry and Placement Configuration on Side Forces in Compression Springs. Major Professor: Arun Prakash, School of Civil Engineering.

A leading cause of premature failure and excessive wear and tear in mechanical components that rely on compression springs for their operation is the development of unwanted side forces when the spring is compressed. These side forces are usually around 10% - 20% of the magnitude of the axial load and point in different directions in the plane perpendicular to the axis of the spring. The magnitude and direction of the resultant of side forces varies very non-linearly and unpredictably even though the axial force behavior of the spring is very consistent and predictable. Since these side forces have to be resisted by the housing components that hold the spring in place, it is difficult to design these components for optimal operation.

The hypothesis of this study is that side forces are highly sensitive to small changes in spring geometry and its placement configuration in the housing. Several experiments are conducted to measure the axial and side forces in barrel springs and two different types of finite element models are developed and calibrated to model the spring behavior. Spring geometry and placement are parameterized using several control variables and an approach based on design of experiments is used to identify the critical parameters that control the behavior of side-forces. The models resulted in deeper insight into the development of side forces as the spring is progressively loaded and how its contact interactions with the housing lead to changes in the side force. It was found that side-forces are indeed sensitive to variations in spring geometry and placement. These sensitivities are quantified to enable designers to and manufacturers of such springs to gain more control of side force variations between different spring specimens.

1. INTRODUCTION

Helical compression springs are one of the most widely used mechanical parts in industry with widespread applications. While the most prevalent form of compression springs is a straight cylindrical spring made from a round wire, many other forms are produced. Helical compression spring come in different shapes such as cylindrical, conical, barrel, hourglass as shown in Fig 1.1 with or without variable pitch. These different shapes lend different characteristics to the helical springs such as reduction of solid height (height of spring at maximum possible compression such that coils come into contact - no additional elastic deflection is possible), higher buckling strength and to produce linear or non-linear load deflection characteristics. This work would mainly be focusing on barrel springs which are coiled from a round wire.

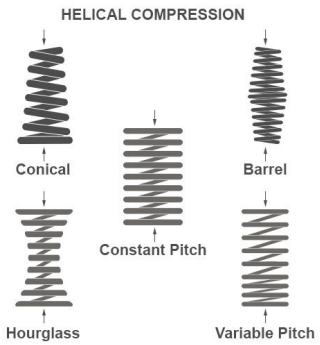


Figure 1.1. Different types of helical spring (source [1]).

Barrel springs belong to the family of helical springs and are used in applications requiring small solid height and increased lateral stability. These springs are composed of two conical springs such that the spring is widest in the middle section and the ends are have smaller diameters. They can be designed in such a way that when loaded the coils can nest partially or completely into the adjacent coil resulting in a

solid height sometimes as small as the coil diameter itself. This phenomena is also termed as the telescopic property of conical springs.

1.1 Application

The barrel spring that is investigated in this work is used in pneumatic brakes for emergency braking of heavy commercial vehicles such as trucks, buses and trailers. These brakes rely on spring force for application of brakes. A cut-out section of typical air-brake chamber is shown in Fig 1.2. While the vehicle is moving the air pressure in the pneumatic chamber keeps the spring in a compressed position with the help of a flexible diaphragm such that the spring is compressed to nearly its solid height. When brakes are applied the air in the pneumatic chamber is bled and the spring is allowed to expand which transmits the force to braking pads with the help of a push-rod. A typical requirement of air brakes is that the chamber needs to be small enough to fit in tight spaces within the vehicles, therefore they require springs which have nesting coils and have small solid heights. Thus barrel springs are used for this application.

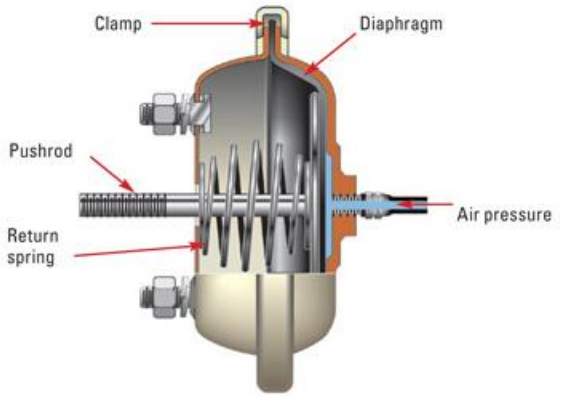


Figure 1.2. Air brakes (source [2]). Annotations indicating the location of spring, diaphragm and pushrod. Pressurized air (on right) enters through a nozzle and loads the spring using diaphragm.

1.2 Manufacturing Process

A typical coiled spring is manufactured in four steps namely, coiling, heat treatment, pre-setting and powder coating. The first step of the process is the coiling, in this process a 'green' spring is produced from the raw material. The raw material in our case is a round wire of diameter 13.5 mm and made up of ASTM A1000 Grade A-Chrome Silicon which has a modulus of elasticity of 207 GPa, minimum tensile strength in the range 1590-2100 MPa. This wire is fed into a coiler where it takes its initial spiral shape with the help of a forming tool as shown in Fig 1.3(a). The 'green spring' is then sent for a two-stage heat treatment process to increase its strength and ductility. During the first stage of heat treatment, the spring is heated to high temperature and is then immediately quenched in oil to increase its strength, after quenching the spring becomes brittle and is therefore sent for tempering in stage two to increase its ductility. After the heat-treatment spring is sent for the process of pre-setting where it is compressed to the design height. The difference in spring heights before and after the pre-setting process can be seen in Fig 1.3(b). The spring is then sent for powder coating process to increase its hardness. Finally the spring is sent for a paint job and the final product is shown in Fig 1.3(c).

(a) Coiling (b) Before & after pre-setting (c) Final product

Figure 1.3. Snapshots of actual spring during different phases of manufacturing process.

It should be noted that during the entire process of manufacturing the spring wire undergoes cold-work and heat treatment which would change the material crystalline structure and would change the material properties of the final product. Therefore the material properties of the raw material are not a reliable estimate of the finished product and we would estimate them with the help of experimental measurements and computer simulations.

1.3 Mechanical Response

During loading of the spring, as the spring is compressed axially it develops lateral/side-forces along with axial forces. These side-forces are generated due to inherent asymmetry of the spring geometry, eccentricity of loading and frictional & normal contact between the spring and loading plates. The side-force for barrel spring is observed to exhibit non-linear response with respect to the loading of the spring. Furthermore, the side-force is sensitive to minor changes in geometry and placement configurations of the spring. Presence of such a non-linear side-force decreases the longevity of the spring and the housing components because of unanticipated wear and tear. Therefore it becomes necessary to characterize the side-forces so as to increase the life of the spring. However, methods available for conventional designing of helical springs do not estimate the side-forces.

1.4 Objective

The objective of this study is to evaluate side forces developed in barrel springs when subjected to axial loads , identify the cause of these forces and characterize & quantify the sensitivity of side forces to geometric and placement-configuration parameters of barrel spring. In this process we will be building several high fidelity finite element models of the spring by calibrating them using experimental measurements.

Such a study will be useful for spring manufacturers in controlling variations in side-forces by distinguishing the sensitivity of different types of variations and therefore design a process of having tighter bounds for critical parameters.

1.5 Existing Literature

Design of helical compression springs has been a topic of interest for researchers since the past century and is still popular now due to its wide-spread application. The existing literature is abundant in works related to helical springs of different shapes, cross-sections, orientations with respect to different applications such as static loading, dynamic loading, free and forced vibrations and buckling. Works such as [3–9] have contributed in analytical/numerical estimation of the load-deflection behavior of the spring under static loading using concepts of Castigliano's theorem, classical spring design and beam theory, verifying their findings with the help of experimental measurements or through finite elements.

However, when a spring is axially loaded there are lateral/side-forces due to friction which is not modeled in the works discussed above. In existing literature there are several works [10–18] that have investigated into the mechanism of side-forces in Mac Pherson strut suspension. As side forces are a consequence of frictional forces, which are hard to model analytically, majority of the work has utilized finite element modeling and/or experimental testing to study the side forces. In [14, 17, 19], major effort has been in finding the force-centerline for the spring and then minimizing the side-forces by either tilting the spring with respect to the strut axis [11, 17] or by changing the shape of the centerline of the spring [10, 13, 14, 16, 17]. In the context of finite element modeling of springs the work by Shimoseki et al. [20] presents well compiled approaches to model different types of springs using beam, gap and solid elements. However, the discussion is limited to calculation of axial forces.

Several works such as [4, 8, 9, 21–31] have contributed in analytical/numerical estimation of free vibrations and/or critical buckling loads for different types of springs

using concepts of beam theory, wave equation, energy equation and solving using transfer matrix approach or customized finite elements.

Works such as [32, 33] have focused on modeling of fatigue using finite elements and [34–37] have investigated in finding the cause of failure of spring using X-ray diffraction (XRD), scanning electron microscopy (SEM), chemical analysis etc.

Several works have investigated the behavior of helical springs composed of non-conventional materials, Mirzaeifar et al. in [38] used shape memory alloy, Aribas et al. in [8] used composite materials, Michalczyk et al. in [29] used metallic springs covered in elastomeric coatings. These works required material modeling in addition to the geometric modeling.

Apart from the common cylindrical helical springs, several works have modeled different geometric shapes and different cross-sections. Conical springs have been a popular shape of investigation in [3–5, 7, 8, 30, 39] as they exhibit non-linear load-deflection response. Chaudhury et al. in [9] have investigated several prismatic springs of non-circular coil shape and non-prismatic springs of circular coil shape. Kaoua et al. in [40] investigated twin helical spring, which can be hard to model, under tensile loading. Gzal et al. in [41] investigated the effect of elliptical cross-sectional spring wires on the stress distribution of the spring when subjected to axial loads.

Further, several works have focused on carrying out sensitivity analysis and finding the optimal design of helical springs. Paredes et al. in [42] designed an optimization tool for straight cylindrical springs with the help of analytical formulas subject to user defined specifications and then using generalized reduced gradients method for solving the optimization problem. Liu et al. in [16] built a multi-body dynamics finite element model and optimized the side-forces for cylindrical springs by estimating an optimal curvature of the centerline. Ryu et al. in [17] presented an analytical process for designing an optimal S-shaped coil spring which minimizes side-forces using finite element analysis. Taktak et al. in [43] compared four different optimization schemes to minimize the mass/natural frequency along with several constraints using numerical formulation developed in [44]. Choi et al. in [45] carried out sensitivity analysis

for cylindrical springs to minimize side-forces using design of experiments (DOE) based approach for meta-model generation and then using genetic algorithm for optimization. Apart from deterministic formulation of problem, papers such as [18, 46] discuss probabilistic problem formulation for sensitivity analysis & optimization which accounts for uncertainty in material properties and geometric dimension due to manufacturing tolerances. Choi et al. in [18] presents such an example by minimizing side-forces for a pig-tail spring.

From the above discussion pertaining to existing literature on helical compression springs it can be observed that majority of studies on estimation of side-forces have focused on cylindrical helical springs whereas this work is focused on barrel springs. Moreover, the existing literature has focused mainly on variations in ceterline of the spring whereas the current study focuses on more-generic form of geometric variations which is variations in diameter and height of the spring along its length due to permissible manufacturing tolerances.

1.6 Approach Followed

In this study, in order to study the side-forces for barrel springs two different finite element models will be discussed in Chapter 3. The first model, in Section 3.1 , is a continuum model with contacts which is constructed to capture the exact physical behavior of the spring. The second model, in Section 3.2 , is a reduced-order model which uses beam elements and non-linear springs to simulate contacts which reduces the computational cost of simulations and thus used for parametric variations. Thereafter in Chapter 4, the effect of placement and geometric variations on side-force of barrel springs is studied in Section 4.1 and Section 4.2 respectively. To carry out the variations a design of experiments approach for sampling and sensitivity analysis was carried out to identify the critical parameters. Finally, the findings of this study are summarized in Chapter 5.

2. MECHANICAL RESPONSE OF BARREL SPRINGS UNDER COMPRESSION

In this chapter we introduce the reader to the notation used, geometry of the model, experimental measurements and tests carried out for the springs. This discussion will serve as foundation for detailed explanation of the problem and will familiarize the reader with certain keywords which will be used throughout the text.

2.1 Notation

This section is dedicated to introduce explicitly the notation that will be used throughout the text. The discussion that follows this section requires usage of global coordinate system and mathematical symbols. Thus a brief discussion about the notation will be helpful.

A global coordinate system is chosen for the description of the problem such that the origin is located at the center of the bottom face of the bottom plate as shown in Fig 2.1. The global Y axis is chosen in the normal direction pointing from the bottom plate to the top plate as shown in Fig 2.1. The global X axis is chosen such that the starting face of the spring is on the XY plane and its outward normal vector is in the direction of Z axis.

Throughout the text any vector quantity will be denoted by a italicized small script letter in boldface, all tensor quantities will be denoted with a italicized & capitalized script letter in boldface and finally all scalars will be denoted by italicized symbols in regular font. With this notation the standard basis vectors can be defined by the symbols $\boldsymbol{e}_1, \boldsymbol{e}_2$ & $\boldsymbol{e}_3$ as the unit vectors such that the sub scripts 1,2 & 3 correspond to the directions along the positive X,Y & Z axes respectively. The standard basis vectors helps in defining the position vector, $\boldsymbol{x}$, of any point in the space. Thus

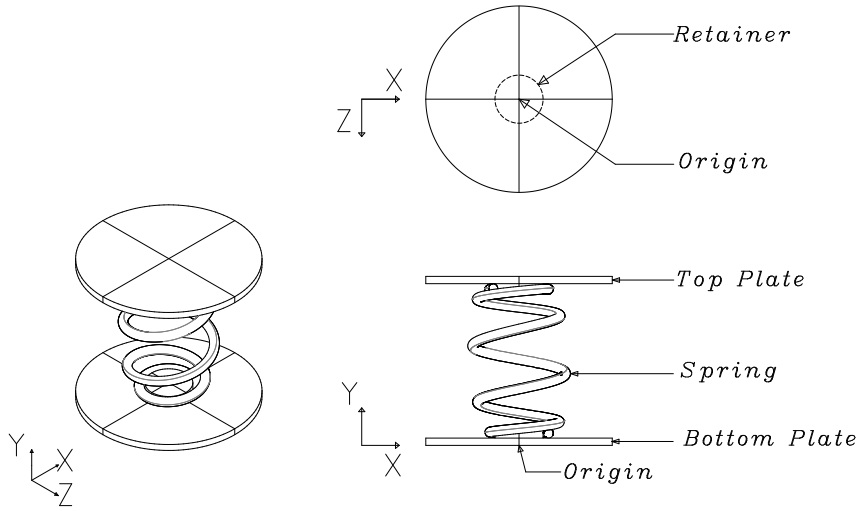


Figure 2.1. Global coordinate system defined with respect to the spring.

position vector for any arbitrary point in space with the coordinate triple (x_1, x_2, x_3) in Cartesian coordinates will be denoted as:

$$\boldsymbol{x} = x_1\boldsymbol{e}_1 + x_2\boldsymbol{e}_2 + x_3\boldsymbol{e}_3 \tag{2.1}$$

An example of a tensor quantity with the above notation can be the identity tensor which will be given by:

$$\boldsymbol{I} = \sum_{i=1}^{3}\sum_{j=1}^{3} \delta_{ij}\boldsymbol{e}_i \otimes \boldsymbol{e}_j \tag{2.2}$$

Where δ_{ij} is the Kronecker delta given by:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{2.3}$$

Another example of a tensor can be the projection tensor, $\mathbf{P}$, which projects a point,$\boldsymbol{x}$, onto the XZ plane to give the projection, $\boldsymbol{y}$ defined as follows:

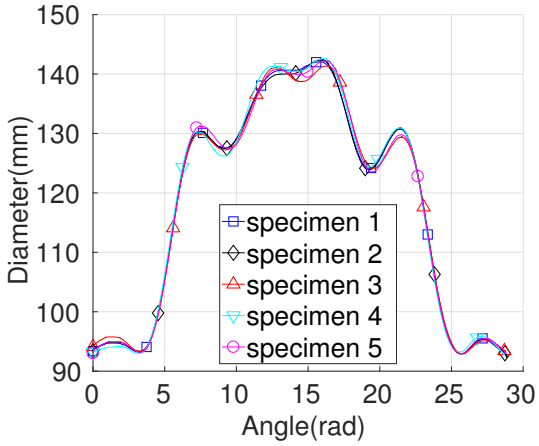
$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{I} - \boldsymbol{e}_2 \otimes \boldsymbol{e}_2 \\ \boldsymbol{y} &= \boldsymbol{P}\boldsymbol{x} \end{aligned} \tag{2.4}$$

In the discussion, the spring is constructed with the help of a center line. The center line of the spring is defined as a collection of N number of points denoted as $\{\boldsymbol{x}_i\}_{i=1}^{N}$. These definitions and notation will now be useful in proceeding with the discussion.

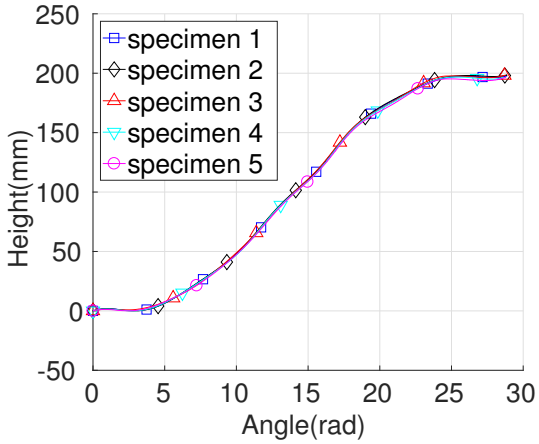
2.2 Spring Geometry and Measurements

This work focuses on the investigation of side forces in barrel spring with variable diameter and variable pitch. The spring is coiled using a wire of cross-sectional diameter of $d = 13.5$ mm. The coordinates of center line, $\{\boldsymbol{x}_i\}_{i=1}^{N}$, of the spring follow a particular profile as shown in Fig 2.2, we may refer to the center line of the spring as wire in the following text. The spring has nearly 4.75 number of coils and a solid height less than 1.5". One such spring is depicted in Fig 2.3.

Manual modeling of springs which follows profile shown in Fig 2.2 can be quite challenging in any computer aided designing(CAD) software. Therefore, the option of python scripting in `ABAQUS` [47] to generate these models was used. When generating these models using a script a wire feature is first created in `ABAQUS` [47] by connecting multiple points along the center line, $\{\boldsymbol{x}_i\}_{i=1}^{N}$, of the spring and then a volume is swept using this path.



(a) Diameter profile



(b) Height profile

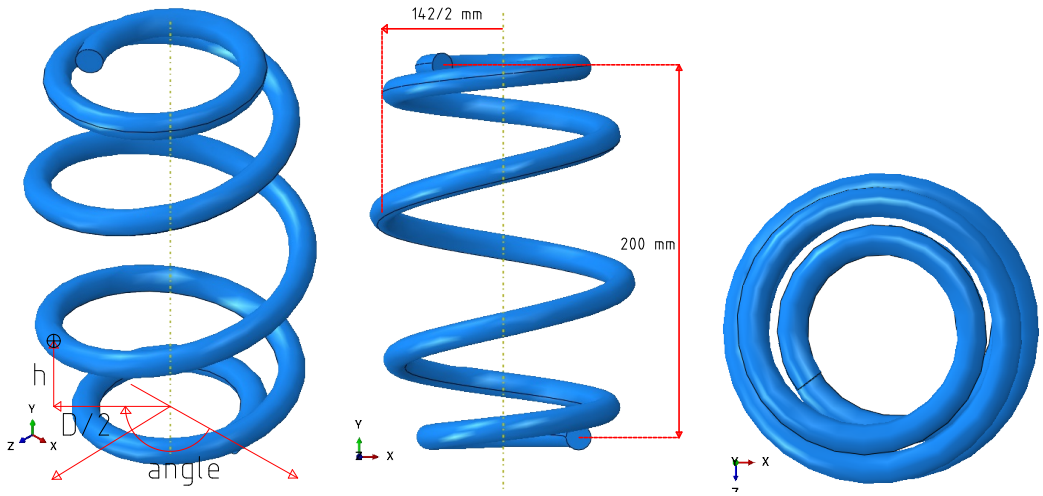
Figure 2.2. Spring Profiles for five different specimens measured using profiler. *(a)* Diameter profile of point on the center line of spring wrt angle *(b)* Height profile of point on the center line of spring wrt angle.

In order to model the springs as shown in Fig 2.3 we need exact measurements of angle, diameter & height of each point, $\boldsymbol{x}_i$, on the center line of the spring. The angle, diameter and height of any point, $\boldsymbol{x}_i$, is defined as follows:

$$\text{height:} \quad h_i = \boldsymbol{x}_i \cdot \boldsymbol{e}_2 - \frac{d}{2} - t \tag{2.5}$$

$$\text{diameter:} \quad d_i = 2 * \|\boldsymbol{P}\boldsymbol{x}_i\|_2 \tag{2.6}$$

$$\text{angle:} \quad \theta_i = \arctan\left(\frac{\boldsymbol{x}_i \cdot \boldsymbol{e}_3}{\boldsymbol{x}_i \cdot \boldsymbol{e}_1}\right) \tag{2.7}$$



(a) Isometric View (b) Front View (c) Top View

Figure 2.3. Spring Model with general dimensions *(a)* Isometric view of spring with annotations indicating height, diameter and angle value for arbitrary point on center line of spring *(b)* Front view of spring with annotations depicting general dimensions *(c)* Top view of the spring showing variable diameter of the spring.

Where t is the thickness of the bottom plate as shown in Fig 2.1 and $\boldsymbol{P}$ is the projection tensor from Equation (2.4)

The profiles shown in Fig 2.2 are measured profiles for five different samples. These profiles were measured using a profiler equipment as shown in Fig 2.4. The profiler works by taking measurements of a moving probe with the help of three rotary encoders which measure the linear (along the height), radial and angular displacement of the probe. While taking measurement of a spring, the spring is first manually fitted between two cones as shown in Fig 2.2. The spring is fitted in such a way that the bottom part of the spring is always at the starting end of the probe. Then with the help of a motor the spring is rotated about the central axis of the cones. As the spring rotates the probe also moves along the outer surface of the spring always maintaining contact and simultaneously the values of the encoders are logged using a data acquisition system. Finally the profile of the spring gets written out a

spreadsheet. The measurements are logged at fine angular resolution such that for our spring of 4.75 turns we have up-to 3000 recordings on average, i.e. a resolution of 0.6 degrees. It should be noted that the profile obtained by the profiler will not be able to capture the very ends of the spring simply because the probe does gets obstructed by the spring coils, therefore our measurements will start from a shifted position than the true start of the spring and will end earlier than the true end of the spring. We assume that such an artifact will not affect the spring response.

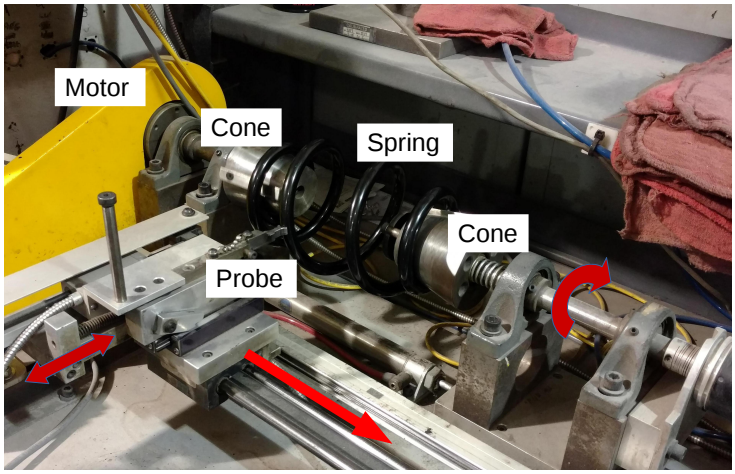


Figure 2.4. Profiler with annotations indicating the location of the cones, spring, motor and probe. Arrows indicating the motion of probe along the spring as the motor rotates the spring.

Another method of obtaining the spring profile that was investigated was using structure from motion technique and using optical scanners. The idea here was to generate a point cloud for the spring and then extract the center line using skeleton extraction techniques available in computer vision literature. When using structure from motion technique with uncalibrated projective cameras the point cloud obtained can be only correct up-to a projective homography which means the point cloud will never have the exact scale as that of the real object and the point cloud might also be suffering from projective distortion. Also, the quality of point cloud highly

depends on the number of images, coverage of view angles of images, amount of change in view angle across different images and illumination of the object, even with perfect conditions it is difficult to get a water tight model of the object. Furthermore, standard computer vision techniques require feature rich objects for extraction of key points. However, the manufactured spring as shown in Fig 1.3(c) is a powder coated and painted spring which is devoid of any feature/pattern. To add features to the spring, it was masked with masking tape and random patterns were scribbled onto the tape as shown in Fig 2.5(a) which helped in obtaining in a comparatively dense 3D reconstruction as shown in Fig 2.5(b) using structure from motion implementation by Wu Changchang et al. [48, 49]. It can be observed that the bottom part of the point cloud shown in Fig 2.5(b) is sparse which would also affect the quality of center line extraction.

(a) Masked spring

(b) 3D reconstruction

Figure 2.5. 3D reconstruction of spring using structure from motion with 120 images.

When using optical scanners one can expect better results than standard structure from motion technique because the optical scanners do not require estimation of

homography and triangulation of points rather it computes the position of a point in space using time of flight of the ray and sensor geometry of the equipment. The 3D reconstruction using optical scanners is correct up-to euclidean homography which means that the point cloud and the object has the same scale and the point cloud will suffer from only rigid body motions, i.e. translations and rotation. The accuracy of 3D reconstruction can be up-to 0.1 mm depending on the sensor accuracy. The quality of point cloud obtained from the scanners depends on the reflectance of the surface, features/patterns available on the surface and size of smallest feature on the object. Two different optical scanners, Sense 3D and Go! Scan 3D, were used to generate the point cloud for spring. It was observed that the quality of point cloud was poor. Also, in some cases the scanner was not able to generate a complete point cloud due to small wire diameter of the spring.

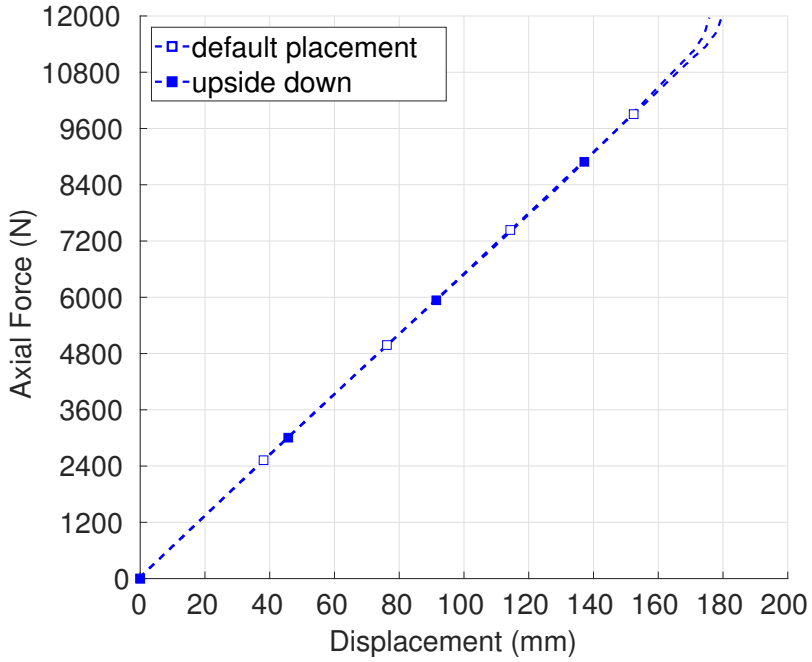
Therefore, due to poor quality of point cloud using structure from motion and using optical scanners, the geometry measured using the profiler was retained for the work presented in the following chapters.

2.3 Loading & Testing Procedure

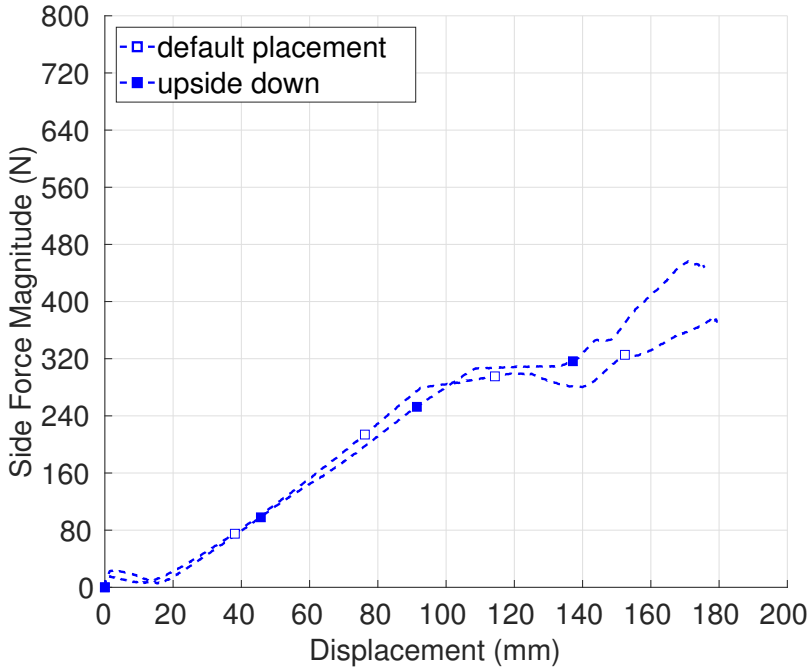
When the spring is manufactured, it is tested for its force response using a universal testing machine (UTM). The spring is placed between two loading plates as shown in Fig 2.1, The bottom plate is a fixed plate and the top plate applies the force, this setup of loading is also known as fixed-ends condition for the spring. There is a circular disc of fixed size screwed onto the center of both the loading plates which helps the user to place the spring properly. This circular disc is also called as retainer and has a diameter of $D = 2.5$" and is d thick disc made up of steel. The UTM is equipped with both axial and lateral load cells which measure the axial and lateral force response of the spring. The application of the spring dictates that the spring needs to be tested up-to a minimum displaced height of 1.5". The force response gets stored in an excel file such that we have the information of height of the top plate,

axial force and lateral forces throughout the loading process of the spring with a fine resolution of nearly 1500 recordings throughout loading of the spring.

When carrying out load-testing of the specimens we have carried out the tests by changing orientation of the spring the details of tests carried out for each specimen will be discussed shortly. First set of tests comprise of tests for each specimen such that the spring is placed in a default placement such that the starting face of the bottom coil of the spring is in the XY plane as shown in Fig 2.1. The second set of tests comprise of tests for each specimen such that the springs rotated about its body center in X direction and is placed upside down in the default placement. The third set of tests comprise of only the specimen 5 such that it is rotated about the Y axis and tested at angles of 0, 45, 135, 225 & 315 degrees. The results for each test for each specimen with legends are shown in Fig 2.6, Fig 2.7, Fig 2.8,Fig 2.9 and Fig 2.10.

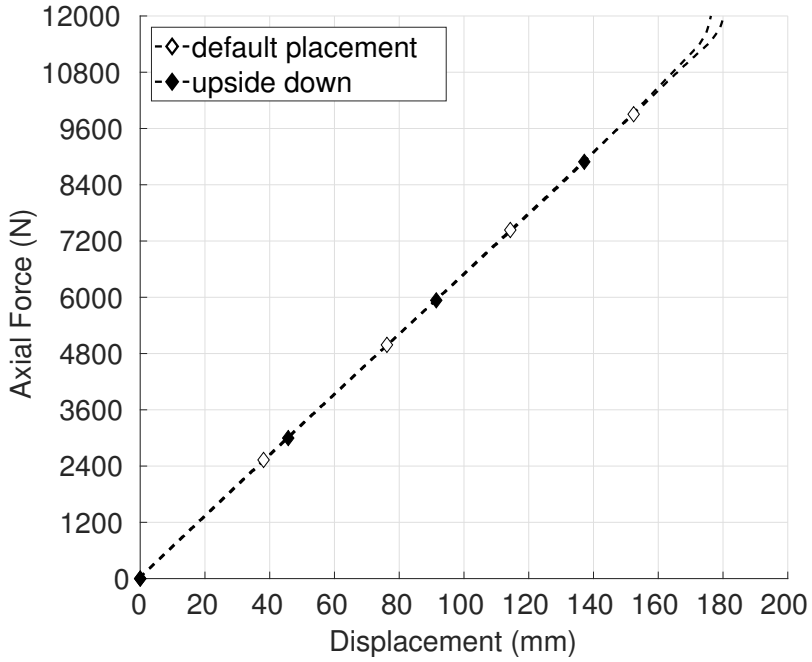


(a) Axial Force

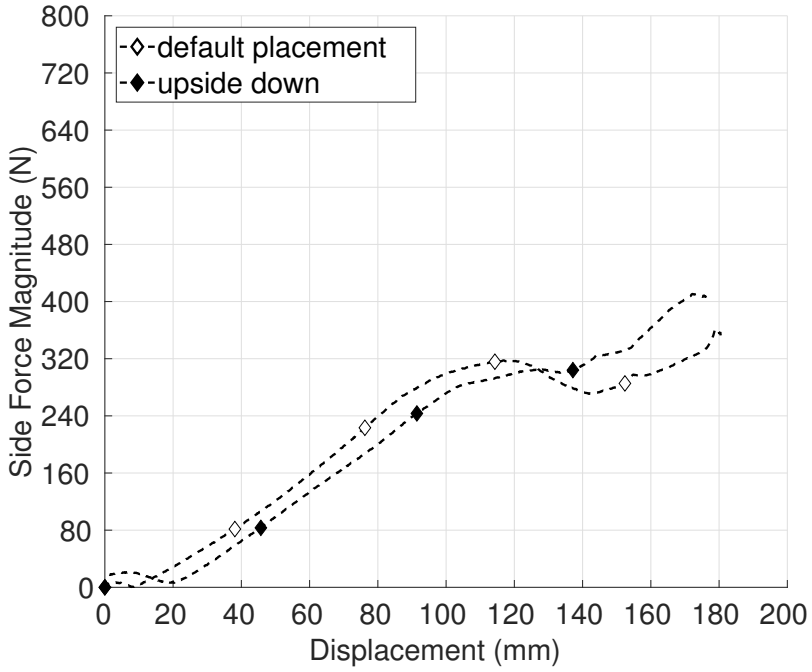


(b) Side Force

Figure 2.6. Measured load response for specimen 1.

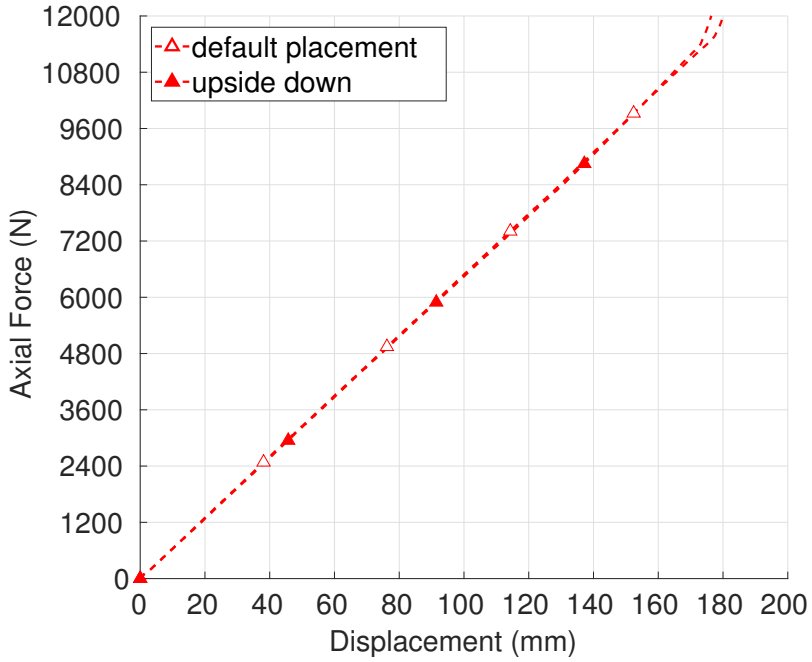


(a) Axial Force

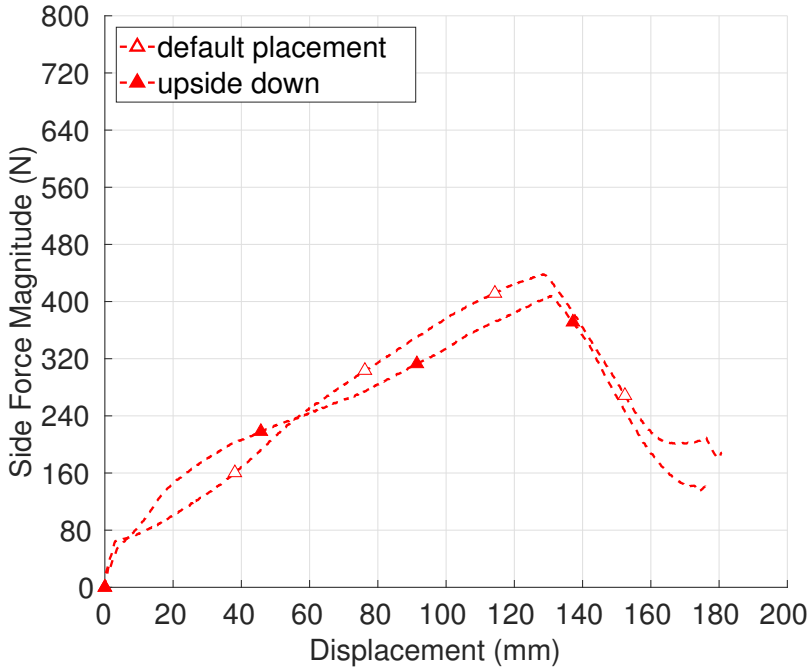


(b) Side Force

Figure 2.7. Measured load response for specimen 2.

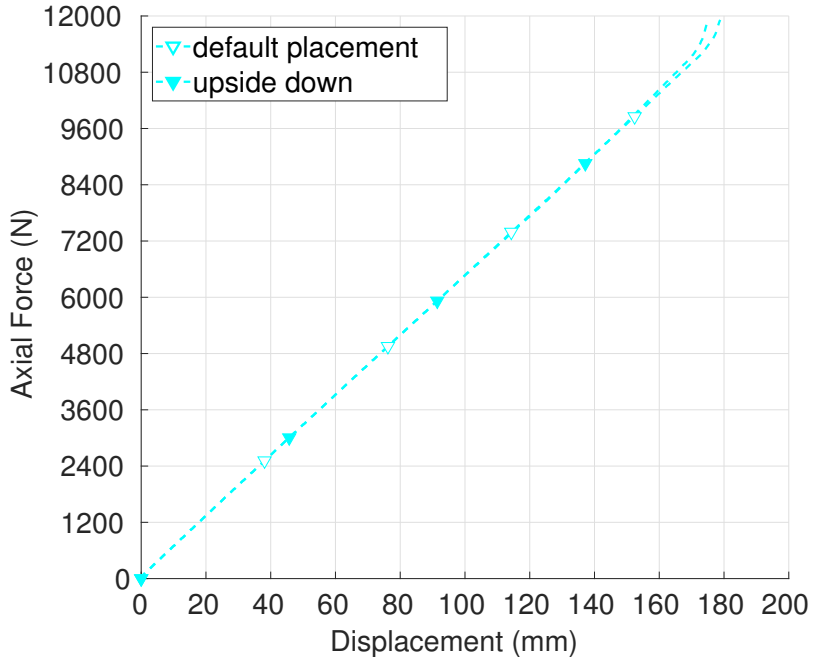


(a) Axial Force

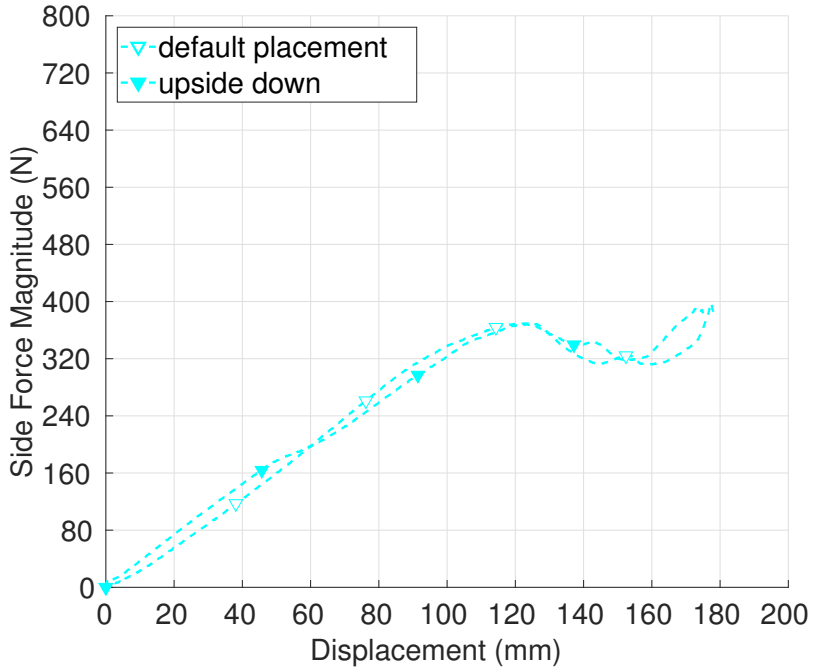


(b) Side Force

Figure 2.8. Measured load response for specimen 3.

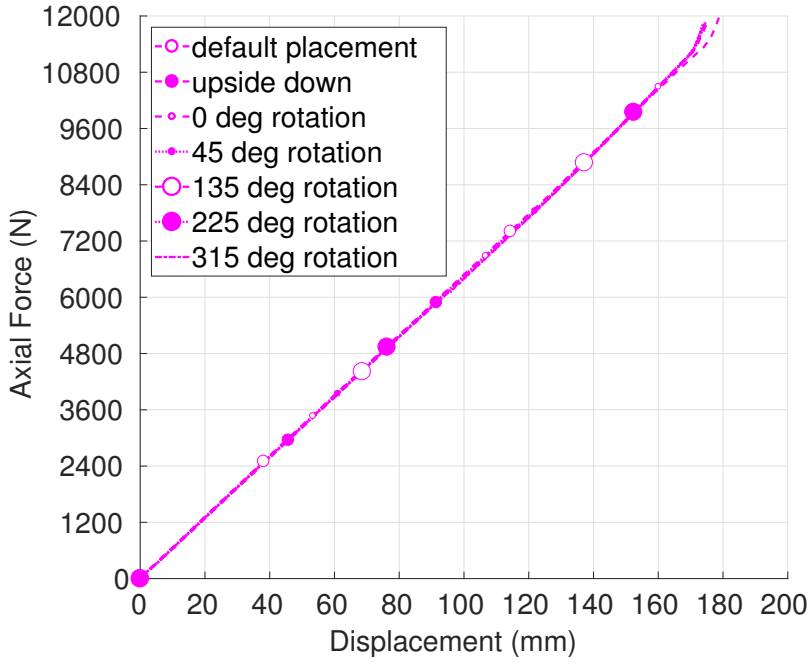


(a) Axial Force

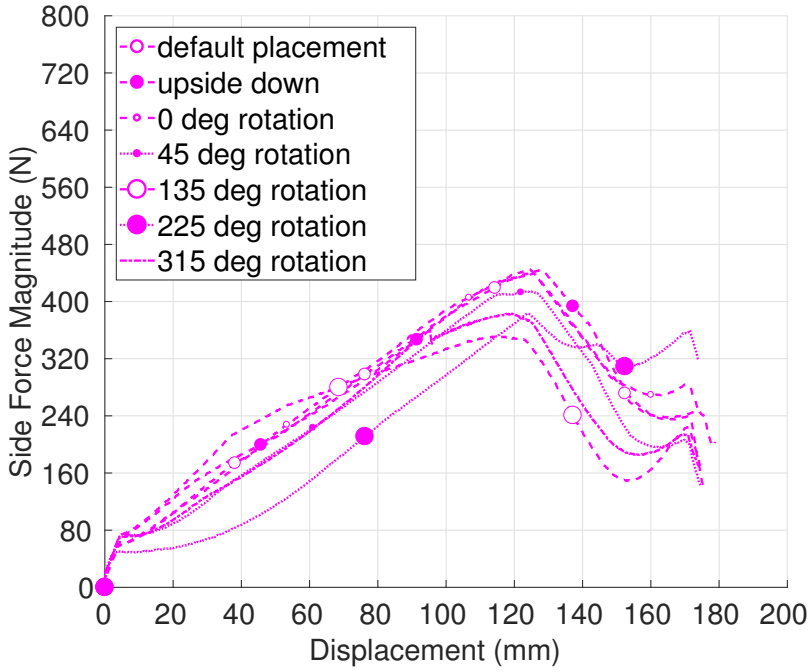


(b) Side Force

Figure 2.9. Measured load response for specimen 4.

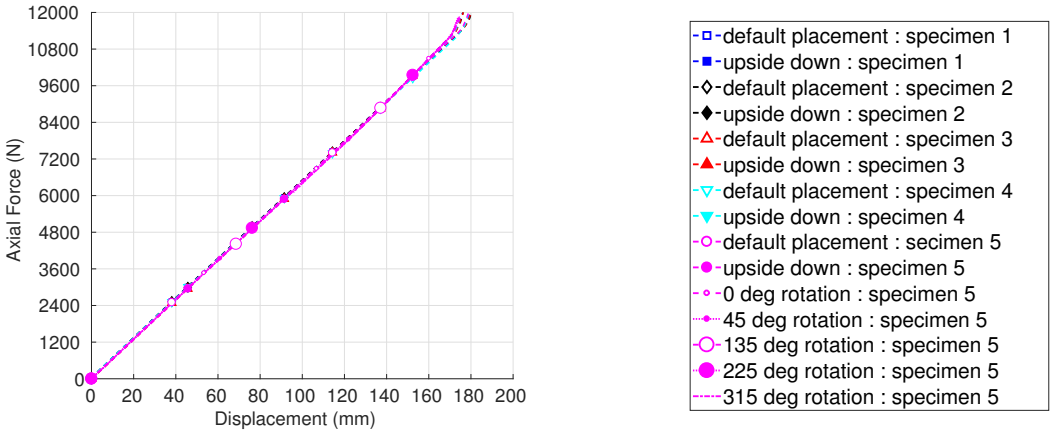


(a) Axial Force

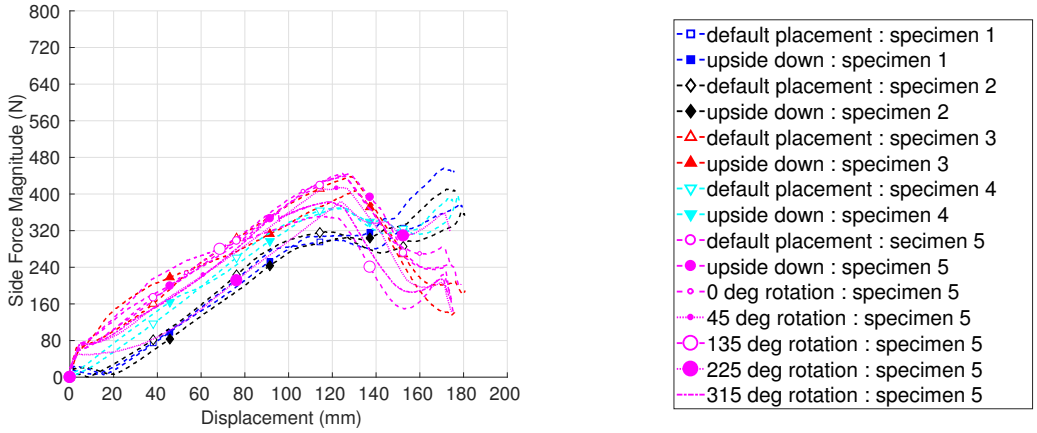


(b) Side Force

Figure 2.10. Measured load response for specimen 5.



(a) Axial Force



(b) Side Force

Figure 2.11. Measured load response for all specimens.

As we can observe from the plots in Fig 2.6, Fig 2.7, Fig 2.8,Fig 2.9 and Fig 2.10, the axial force response for the five specimens is the same and remains a linear response for the entire loading however, the side-force response varies a lot across the five specimens and is a non-linear response. We can also observe that the plots for the default placement and upside down placement do not vary as significantly as the plots shown for variation in rotation angle. Further, we observe such a spread of side-force even when the differences in profiles of the five specimens as shown in Fig

2.2 is very subtle. Therefore, we are interested in identifying the cause of side forces, explain why they are non-linear and identify what parameter affects the side force the most.

2.4 Mechanical Response of Barrel Springs

The mechanical response of the barrel springs can be affected by several factors such as spring geometry, configuration of the spring, material properties of spring and changing contact during the deformation of the spring. Several geometric characteristics of the spring can affect the side-force, for instance if the wire diameter of the spring is larger then the axial forces will increase as more volume of material will be compressed and as the axial forces increase the side forces should also increase as they would be directly proportional. Another geometric effect can be when the spring cross-section is of a slightly different shape then the way how the spring contacts the loading plates can change and thus the side-forces. Similarly, local geometric variations due to manufacturing tolerances can result in change of contact behavior which can affect the side-forces. Further, springs with a inclined/curved central axis can also affect the side-forces as it can allow/prevent buckling of spring. Furthermore, when the spring is loaded there can be self-contact between the coils which can affect the axial and side force response.

Also, several configuration parameters which dictate how the spring in placed in while loading can affect the side-force response. For instance, if the clearance between the spring and retainer is tight then the spring may contact the retainer leading to normal forces on the retainer which will contribute to the side forces. Another case can be when the clearance is loose, then the spring has more room to move and rotate during loading which can affect the side-forces as the contact status would change.

Furthermore, material properties of the spring can also affect the side-force response. As the spring goes under a combination of cold-work and heat treatment as discussed in Section 1.2, material crystalline structure of would change across the

cross-section of the spring and can also be different for different batch of manufactured springs. This can result in variation in Young's Modulus (E) and Poisson ratio (ν) of the spring which can affect the axial force and correspondingly side force response. Also, compression springs undergo the process of presetting which induces useful residual stresses which are opposite in direction to those induced in service, thus increasing the axial strength of the spring which would also reflect in side-force response. Furthermore, during deformation as the spring is loaded only localized parts of the spring come into contact with the plate, the stresses at these locations can be higher than the bulk which can cause localized plasticity effects which would affect the axial and side force response.

To understand the force mechanism of the spring during its loading it will be now useful to discuss about the free body diagram of a section of of spring. The free body diagram for the barrel spring is shown in Fig 2.12. When the spring is compressed, it will develop axial (along Y axis) forces (F) to resist the compression. These axial forces will be distributed throughout the length of the spring coils which are in contact with either of the plates. Also, due to friction between the plate & spring and normal force due to contact of spring with the retainer there will be side forces (S) coming into the picture. Furthermore, due to the distribution of axial and side forces there will be moments (m_x, m_y, m_z) in all principal directions over a single element of spring which will cause bending and torsional effect on the element. The magnitude of moments will depend on the distribution of the axial and side forces which inherently depends on how the spring contacts the plates. Now, as the spring deforms during its loading, more and more portion of spring will be coming into contact with plates which will change the distribution of axial and side forces.

The free body diagram shown in Fig 2.12 is more general than what is discussed in conventional spring design textbooks such as Shigley's [50]. In conventional spring design the axial loading is assumed to be a point load centered at the plate center and side-forces are not even taken into consideration whereas in our case we are assuming a distribution of axial and side force on the spring coil based on contact.

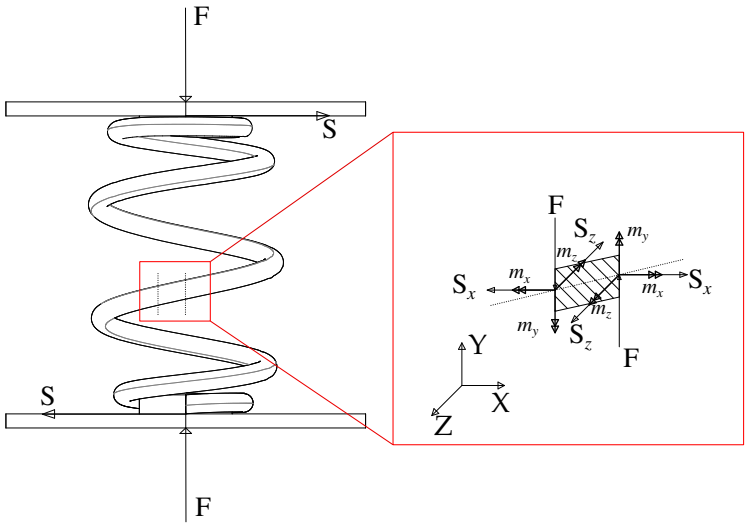


Figure 2.12. Free body diagram of an element (in inset) of spring.

Furthermore, conventional design gives us analytical solutions for simple geometry of spring and will not be applicable for complex geometry as discussed in Section 2.2. Finally, conventional design is only valid for small deformations which will have a linear response whereas in this problem we want to calculate the spring response from free length to nearly solid height i.e. large deformations.

The side force have been previously studied with the help of a zero-moment force line in several works such as [10–18], where the distribution of forces was simplified by assuming a point load with fixed ends providing reaction moments. Whereas, in this study no loading simplification is made and the distribution of forces is taken in to consideration.

As this problem involves several challenges such as geometric variations, several possible placement configurations, large deformations, evolving status of contact of the spring with the plate and/or retainer which results in unpredictable non-linear side-force response. Therefore, we will analyze the problem with the help of finite element analysis with geometric non-linearity and contacts. The details the models built for this problem will be discussed in the next chapter.

3. MODELING BARREL SPRINGS

In this chapter, details of two types of Finite Element Models generated to simulate the problem are discussed. When generating any finite element model attention should be given to the choice of material definitions, Boundary conditions and choice of elements, these details should be accurately chosen such that they mimic the actual process which in this case is the load testing procedure of the spring. Further, as we have uncertainty over material property values the model needs to be calibrated to the experimental results. We will discuss these details in context to the two models and then will be presenting results obtained from these models for the five specimens.

3.1 High-fidelity Model using Continuum Elements and Contact

The key idea here is to model the exact physical problem without any approximations. In this model we will be using continuum elements to model both the plate & retainer and the spring, with contact formulations which makes the model computationally expensive. With this model we will be able to visualize the contact behavior and answer as to why the side-forces have a non-linear behavior but we pay the price by having more number of elements and then contact formulations can require very small time-steps and sometimes are difficult to converge, which increases our computation and memory cost.

3.1.1 Model Assumptions

It should be noted that while defining the model, certain modeling assumptions were made. The cross-section of the spring was assumed to be circular, however, the spring is cold-formed using a circular wire whose cross-section can change during the forming process. Also, the material properties of the spring were assumed to

be constant, however, when the spring goes through the heat treatment process the material crystalline structure can vary across the cross-section of the spring due to different temperatures of the core and crux of the spring cross-sections, thus the they can have different material properties. Further, the spring is pre-setted which will result in residual stresses in the spring, these stresses are not modelled in our model.

3.1.2 Geometry & Type of Elements

In the discussion to follow we will be using the global coordinate system and notation defined in Section 2.1. As discussed earlier, the geometry of the spring is generated using the center-line coordinates of the spring. These points, $\{\boldsymbol{x}_i\}_{i=1}^{N}$, are used to create a wire feature in `ABAQUS` [47] to define the spring. After creation of the wire, we sweep a circular cross-section along the wire to generate our solid spring. Further, in this model we will create solid parts for the plate and retainer, these features are created using extrusion in `ABAQUS` [47] . The model can be seen in Fig 3.1.

As discussed earlier, during the loading of the spring, it will experience bending loads and therefore we need to choose an element which has good response in bending. In this model we are using 8-Node Continuum 3D element with incompatible modes (C3D8I) as they are better suited for bending applications. These elements are also called as non-conforming elements and differ from a regular linear element as they have additional quadratic modes of deformation corresponding to the exact pure bending solution. For more information the reader can refer to the textbook by Thomas Hughes [51] .These elements offer us quadratic behavior for bending without increasing the the number of nodes, thus less expensive than a full quadratic element. For the plate and retainer we are using 8-Node Continuum 3D linear element (C3D8).

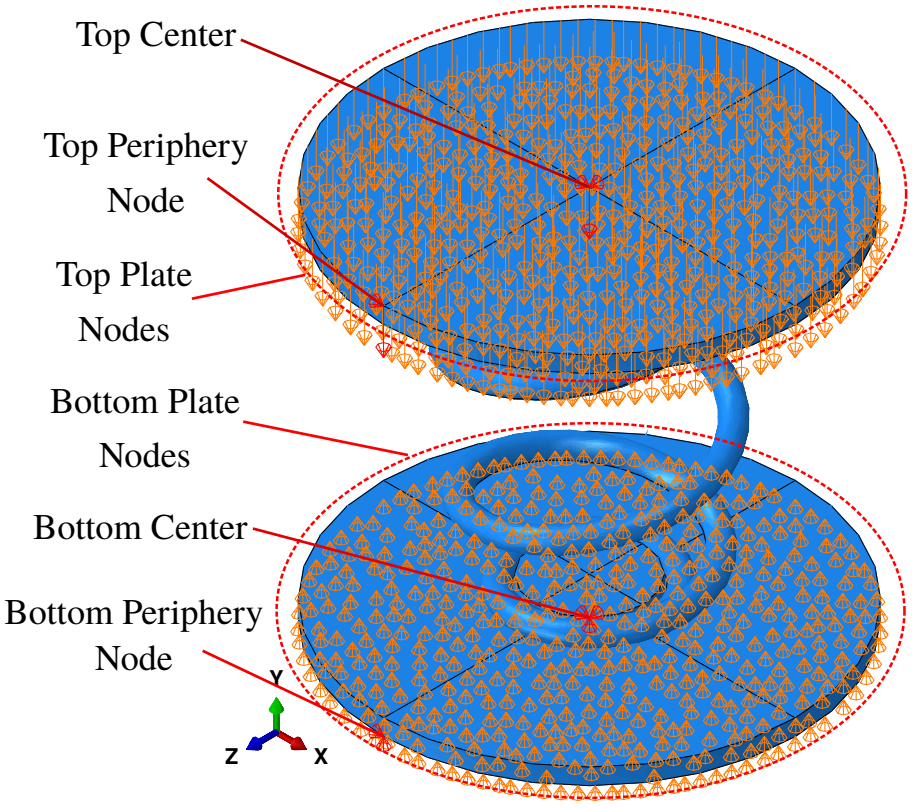


Figure 3.1. Continuum model with annotations indicating regions for boundary conditions with fixed displacement for bottom plate and specified displacement for top plate. Boundary conditions for plate centers and plate periphery points are colored in red.

3.1.3 Material Properties

In this model we have two different materials for loading plates and spring respectively. As both the components do not experience any permanent deformation, therefore it is safe to assume a linear material model. The loading plate is made of Steel with Young's Modulus of 207 GPa and a Poisson ratio of 0.3. As discussed earlier, since we don't have reliable estimate of the material properties for the spring therefore we need to tune them with the help of experimental results.

3.1.4 Boundary Conditions

The boundary conditions defined for this model will be with respect to the coordinate system defined earlier in Section 2.1. We would also use the sequence of points $\{\boldsymbol{x}_i\}_{i=1}^{N}$ which defines the center line of the spring. The boundary conditions for the model are defined as follows:

1. To simulate the testing procedure all dofs of the bottom center (shown in Fig 3.1) are fixed to zero.

$$\boldsymbol{u}(\boldsymbol{x}_b) = \boldsymbol{0} \tag{3.1}$$

$$\text{where,} \quad \boldsymbol{x}_b = (t + \frac{d}{2})\boldsymbol{e}_2 \tag{3.2}$$

2. To simulate loading the top center (shown in Fig 3.1) point is given a fixed displacement.

$$\boldsymbol{u}(\boldsymbol{x}_t) = -\alpha \boldsymbol{e}_2 \tag{3.3}$$

$$\text{where,} \quad \boldsymbol{x}_t = (t + d + H)\boldsymbol{e}_2 \tag{3.4}$$

The magnitude of the displacement is such that the spring height at max displacement shall be 1.5 inches.

$$\alpha = (H + d - 1.5 * 25.4) \tag{3.5}$$

3. To simulate fixed position of bottom plate, zero displacement is given for bottom plate nodes (shown in Fig 3.1) in the Y direction and the points are free to move in X and Z directions.

$$\boldsymbol{u}(\boldsymbol{x}) \cdot \boldsymbol{e}_2 = 0 \quad \forall \quad \boldsymbol{x} \in S_b \tag{3.6}$$

$$\text{where,} \quad S_b = \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + \frac{d}{2})\boldsymbol{e}_2 \quad \forall \quad i = \{1, 2, .., N\}\} \tag{3.7}$$

4. To simulate movement of top plate, fixed displacement is given for top plate nodes (shown in Fig 3.1) in the Y direction and the points are free to move in the X and Z directions.

$$\boldsymbol{u}(\boldsymbol{x}) \cdot \boldsymbol{e}_2 = -\alpha \quad \forall \quad \boldsymbol{x} \in S_t \tag{3.8}$$

$$\text{where,} \quad S_t = \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + d + H)\boldsymbol{e}_2 \quad \forall \quad i = \{1, 2, .., N\}\} \tag{3.9}$$

5. To resist rigid body rotation of the bottom plate about its center, $\boldsymbol{x}_b$, we restrict the tangential dof of a node on the periphery, $\boldsymbol{x}_l$, of the bottom plate on its bottom surface in the direction of Z axis.

$$\boldsymbol{u}(\boldsymbol{x}_l) \cdot \boldsymbol{e}_3 = 0 \tag{3.10}$$

$$\text{where,} \quad \boldsymbol{x}_l = \boldsymbol{x}_b + (\frac{d_p}{2})\boldsymbol{e}_3 \tag{3.11}$$

   Where $\boldsymbol{x}_b$ is from Equation (3.2) and d_p is the plate diameter

6. To resist rigid body rotation of the top plate about its center, $\boldsymbol{x}_t$, we restrict the tangential dof of a node on the periphery, $\boldsymbol{x}_u$, of the top plate on its top surface in the direction of Z axis.

$$\boldsymbol{u}(\boldsymbol{x}_u) \cdot \boldsymbol{e}_3 = 0 \tag{3.12}$$

$$\text{where,} \quad \boldsymbol{x}_u = \boldsymbol{x}_t + (\frac{d_p}{2})\boldsymbol{e}_3 \tag{3.13}$$

   Where $\boldsymbol{x}_t$ is from Equation (3.4) and d_p is the plate diameter

It should be noted that, in the above discussion for boundary conditions, whenever any component of displacement, $\mathbf{u}$, is not assigned any value then it is assumed to have a free dof.

With such a setup the axial reaction on the bottom plate can be obtained by summing up the reactions on all the nodes in $S_b \cup \boldsymbol{x}_b \cup \boldsymbol{x}_l$. For the side force we need to sum up the reactions at the bottom center node, $\boldsymbol{x}_b$, and the periphery node, $\boldsymbol{x}_l$.

In addition to the above boundary conditions contact interaction using general contact in `ABAQUS` [47] was defined for the model. The normal and tangential contact

were defined using penalty formulation with strain free adjustments for initialization of contact. As the spring is always in contact with the plate and can also have self-contact during loading therefore the choice of general contact is appropriate here. For the tangential contact, a fixed coefficient of friction of $\mu = 0.7$ was selected.

An alternate choice of boundary conditions for the bottom & top plate can be such that all the points are fixed for in-plane movement, defined as follows:

1. For bottom plate:

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{0} \quad \forall \quad \boldsymbol{x} \in S_b \tag{3.14}$$

$$\text{where,} S_b = \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + \frac{d}{2})\boldsymbol{e}_2 \quad \forall \quad i = \{1, 2, .., N\}\} \tag{3.15}$$

2. For top plate:

$$\boldsymbol{u}(\boldsymbol{x}) = -\alpha * \boldsymbol{e}_2 \quad \forall \quad \boldsymbol{x} \in S_t \tag{3.16}$$

$$\text{where,} \quad S_t = \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + d + H)\boldsymbol{e}_2 \quad \forall \quad i = \{1, 2, .., N\}\} \tag{3.17}$$

even with this choice of boundary conditions it was observed that the model behavior for the force-response and contact behavior remained un-changed.

3.1.5 Analysis & Post-Processing

We are carrying out static simulation with geometric non-linearity. As there are contacts involved in this model they make model convergence hard, in order to aid the convergence we have chosen penalty formulations instead of Lagrange multiplier formulation because penalty formulation induces numerical softness. Also, we are using automatic stabilizing with volume proportional damping to aid in solving of unstable problems . The description of automatic stabilization in `ABAQUS` [47] states that viscous forces ($\boldsymbol{F}_v$) of the form:

$$\boldsymbol{F}_v = c\boldsymbol{M}^*\boldsymbol{v} \tag{3.18}$$

are added to the global equilibrium equations

$$\boldsymbol{P} - \boldsymbol{I} - \boldsymbol{F}_v = \boldsymbol{0} \tag{3.19}$$

Where $\boldsymbol{M}^*$ is an artificial mass matrix calculated with unit density, c is damping factor, $\boldsymbol{v} = \Delta\boldsymbol{u}/\Delta t$ is the vector of nodal velocities, Δt is the increment of time, $\boldsymbol{P}$ is the external load and $\boldsymbol{I}$ is the internal forces developed due to loading.

From the above definition of viscous forces, we can expect that the internal forces ($\boldsymbol{I}$) can be insignificant even when we have significant external forces ($\boldsymbol{P}$). This can happen when the viscous forces are equal and opposite to external forces. This means that we can expect to see insignificant reaction forces even when the spring has already started compressing. Such an artifact will be evident during the initial phase of loading of the spring, when the contacts are not fully established.

After the simulation is completed, we extract the values of displacement and reaction forces at specific nodes for post-processing. The node locations of the quantities extracted is detailed as follows:

- Side-force: Vector sum of reaction force at the bottom center node ($\boldsymbol{x}_b$) and bottom plate periphery node ($\boldsymbol{x}_l$).
- Displacement: Value of the displacement at the top center node of the top plate ($\boldsymbol{x}_t$).
- Axial-force: Sum of axial force (along Y direction) for all the nodes, $\boldsymbol{x} \in S_b \cup \boldsymbol{x}_b \cup \boldsymbol{x}_l$, on the bottom face of the bottom plate.

3.1.6 Model Results

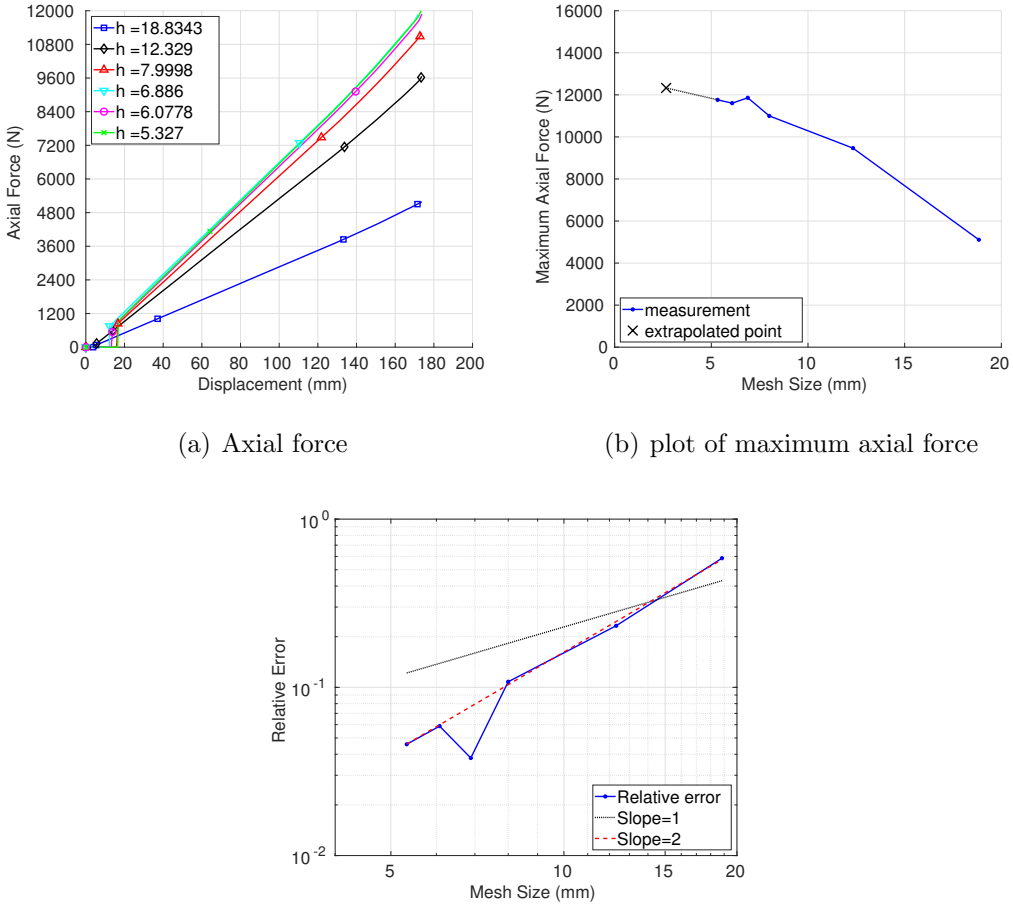
Prior to the discussion of the results of the model we need to prove the convergence of our model, the model parameters chosen for proving the convergence are:

- Young's Modulus of spring: $E = 250$ GPa
- friction coefficient: $\mu = 0.7$

After fixing the values of the above parameters, to prove convergence we need to check if a particular metric converges with respect to the number of elements. As we want our model to first correctly predict the axial response and then the sideforce response, therefore we have chosen the maximum value of axial force to compute relative error which then acts as the metric for convergence. As we don't know the true solution of the problem, we assume that the true solution will be at half the mesh size of our finest mesh sizes. Then we extrapolate the maximum forces to this assumed true solution mesh size and use the extrapolated value to compute the relative errors. The results for the convergence study are shown in Fig 3.2.

From Fig 3.2(c) we can see that the relative error of maximum axial force converges with a quadratic rate (Slope is 2). Furthermore, for mesh sizes smaller than and equal to 6.886 have relative errors less than 10%. Therefore, in order to reduce computational time for the simulation for the continuum model choosing mesh size of 6.886 mm will be adequate as it has a relative error of 4% and is computationally cheaper than mesh sizes 6.0778 and 5.327. Also, for the mesh size of 6.886 mm we have a total number of $61,315$ nodes.

After estimating the mesh size for converged model, the next step is to calibrate the converged model such that its response matches the experimental data. For the continuum model there are only two parameters,Young's Modulus of spring and coefficient of friction, that can be tuned in order to calibrate the model. In order to judge the quality of calibration, squared difference of axial force response of model and experimental data was taken as the metric. To estimate the parameters, an array of values were tried for each parameter separately and the value which had the smallest squared difference was chosen as the tuned value of the parameter. The array of values chosen for Young's Modulus (E) were 200, 225, 250, 275, 300, 325, 350 GPa. It was observed that the Young's Modulus was a sensitive parameter and a change in its value would change the slope of the axial force response. The minimum squared difference was observed for Young's Modulus of 250 Gpa. The array of values chosen for coefficient of friction (μ) were 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.5. It was observed



(a) Axial force

(b) plot of maximum axial force

(c) Log-Log plot of convergence

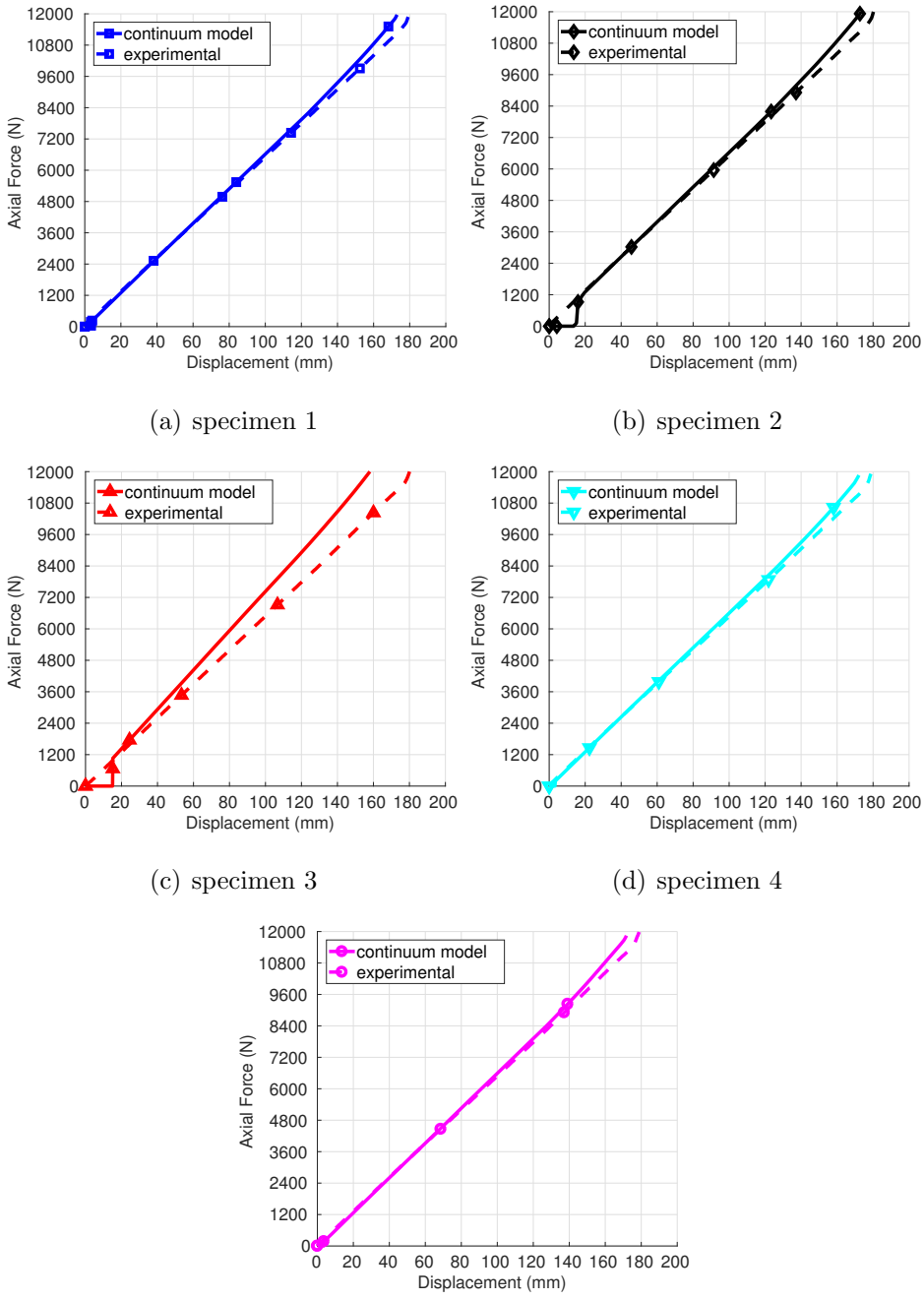
Figure 3.2. Convergence of Solid Model *(a)* Axial response of the model converging to the same sloped line as the mesh size decreases *(b)* Measured value of maximum axial force for different mesh sizes along with the extrapolated point which is computed at half the edge length of finest mesh *(c)* Relative error (using extrapolated point as the assumed true solution) can be observed from the log-log plot to converge at a quadratic rate with respect to the mesh size.

that the model response remained un-affected to change in μ, which implies that the frictional force experienced by the spring is always less than the static friction i.e. there is no slipping.

The simulation results for the five test specimens along with their experimental results for converging model with calibrated parameters are shown in Fig 3.3 and Fig 3.4.

From Fig 3.3 and Fig 3.4 we can see that the model is able to produce axial force response which match the experimental results for all the specimens. This indicates that the parameter value that we obtained is a reliable estimate. On the other hand, the side-force response is stiffer than the experimental results but it still follows the trend. Also, it should be noted that there is an initial nearly-zero force response state for all of the specimens, this artifact is because we are using automatic stabilization with volume proportional damping to aid in solving of initial contact detection by adding viscous forces to the global equilibrium equations as discussed in Section 3.1.5. This causes an initial nearly-zero response state as the viscous forces dominate over the internal forces.

We can notice that the side force response for specimen 1 and specimen 5 are significantly different as evident from Fig 3.4. A closer look at the results for specimen 1 and 5 in Fig 3.6-Fig 3.11 and Fig 3.13-Fig 3.18 respectively gives us insights into the behavior of side forces. These plots are generated at specific locations along the displacement history of the specimens as shown in Fig 3.5 and Fig 3.12 for specimen 1 and 5 respectively. Starting with specimen 1, from Fig 3.6(a) and Fig 3.6(b) we can see the contact region on the bottom and top plate which are in contact with the spring. It can be noticed from these plots that even though there is a wide range of contact region the reaction forces on the plates remain concentrated at certain specific regions only as shown in Fig 3.6(c) and Fig 3.6(d). The friction forces contributing to the side forces will therefore can be safely assumed to higher at the regions where we have higher reaction forces. Now, as the spring gets compressed the regions of contact and regions of reaction forces also change which is shown in Fig 3.6-Fig 3.11. This means that the frictional forces on the plate will change and hence the side-force will also change.



(a) specimen 1

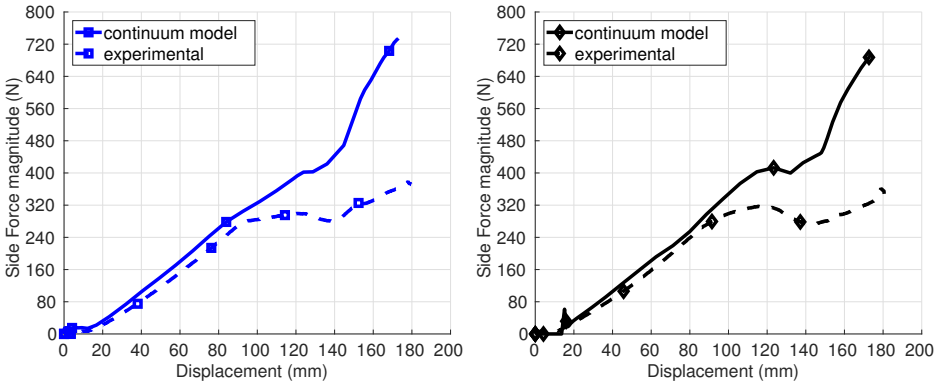
(b) specimen 2

(c) specimen 3

(d) specimen 4

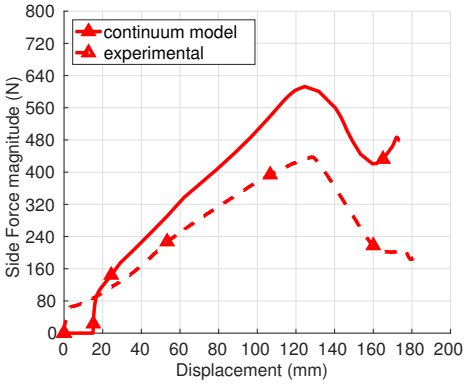
(e) specimen 5

Figure 3.3. Axial force for Continuum Model.

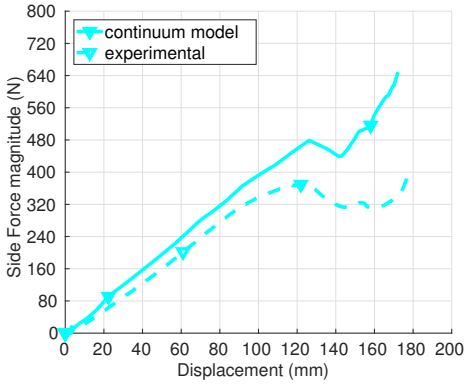


(a) specimen 1

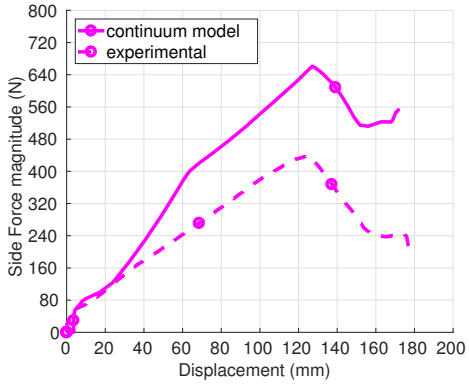
(b) specimen 2



(c) specimen 3



(d) specimen 4

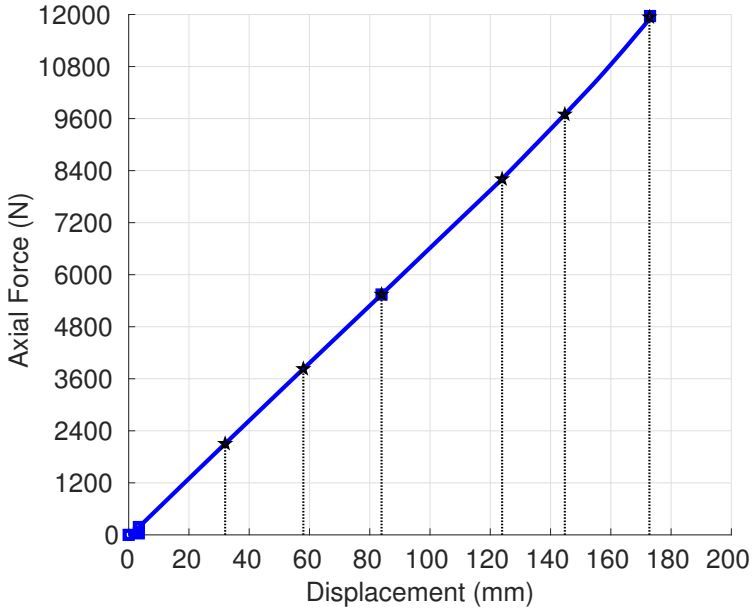


(e) specimen 5

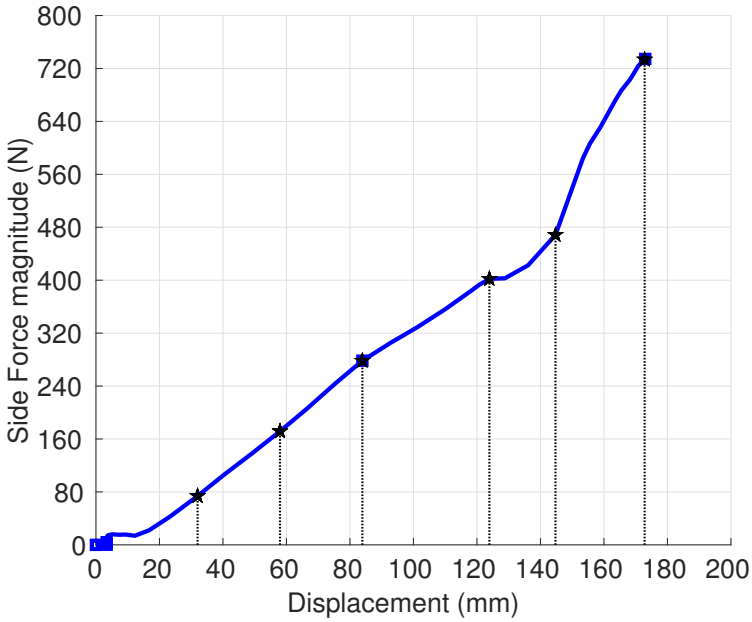
Figure 3.4. Side force for Continuum Model.

Another thing to note is that the concentrated regions of reaction forces are atmost 2 or 3 for the majority of the loading and only towards the end more regions emerge. This means that throughout the majority of the loading of the spring, it is supported by 2 or 3 points which corroborates with findings of Shimoseki et al. [20]. Also the 2 or 3 regions of concentrated forces mainly correspond to the first and last full turn of the spring, we will use this finding to our advantage in defining a computationally cheaper model in Section 3.2. Further, a similar behavior is evident for specimen 5 as evident in Fig 3.13-Fig 3.18.

On comparing the behavior of specimen 1 and specimen 5, we can observe from Fig 3.6(c) and Fig 3.13(c) that the regions of concentrated forces for specimen 1 are located diametrically apart whereas for specimen 5 they are angled at approximately 150 degrees ($<$ 180 degrees). The regions of contact also vary for the two specimens as shown in Fig 3.6(a) and Fig 3.13(a). This can be explained from minor differences in the profile of the two springs. Any point on the spring which is normally (i.e. along Y axis) closer to the plate will come into contact earlier and thus dictate the regions of contact and regions of concentrated forces. When we zoom into the height profile for the specimens as shown in Fig 2.2 we can get two plots for the two ends of the spring as shown in Fig 3.19. It can be seen from Fig 3.19(a) that the lowest point for all springs is at the starting of the spring, and the next lowest point for specimen 1 and specimen 5 is at approximately 2.5 and 3 radians. This means that the regions of contact for these two specimens should show initial concentrated regions at these particular angular locations which is in-fact true as shown in Fig 3.6 and Fig 3.13. This indicates that the side forces are sensitive to minor changes in profile of the spring.

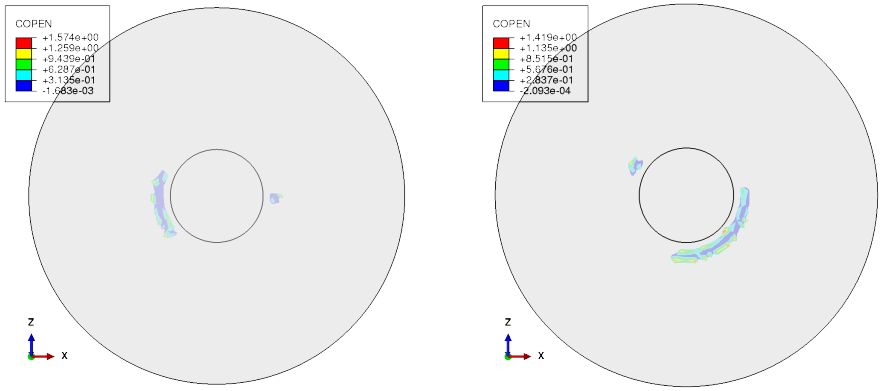


(a) axial force



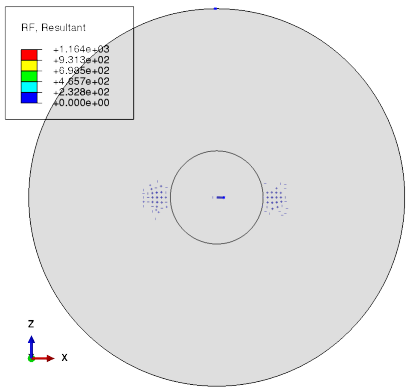
(b) side force magnitude

Figure 3.5. Specimen 1: Displacement locations selected for solid model plots.

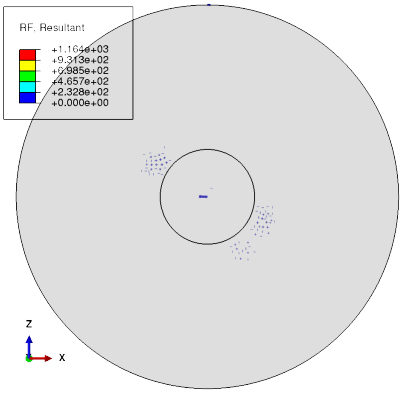


(a) Contact region on bottom plate

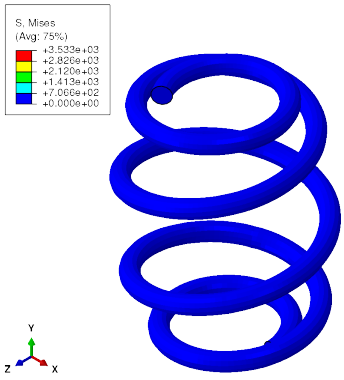
(b) Contact region on top plate



(c) Reaction force for bottom plate

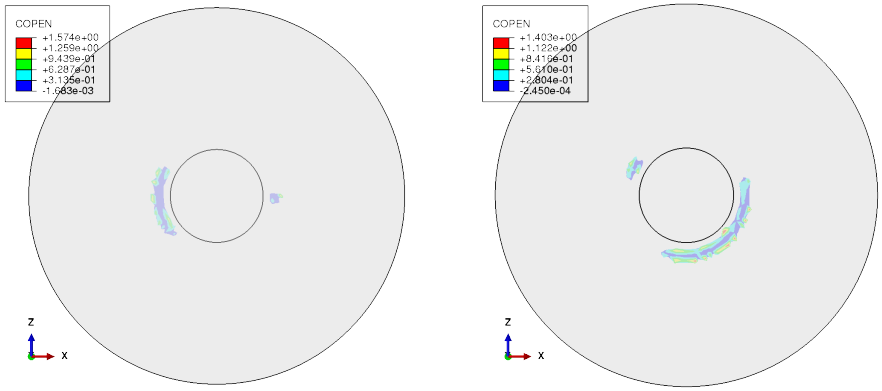


(d) Reaction force for top plate

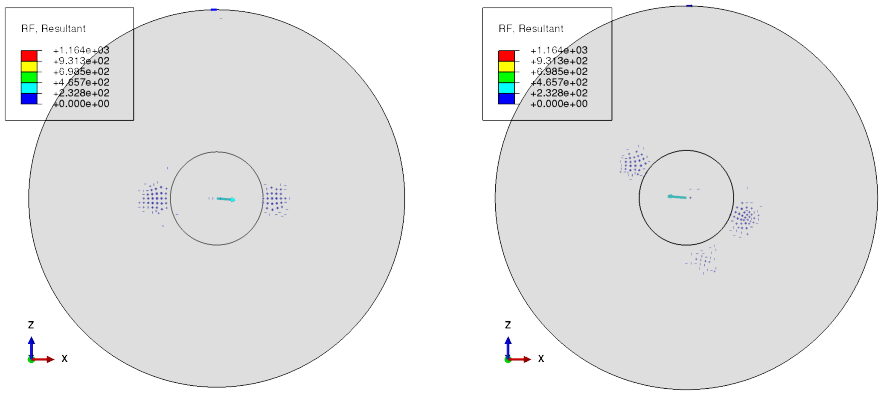


(e) Deformation shape of spring

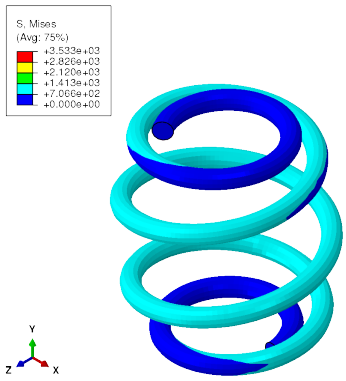
Figure 3.6. Specimen 1: Solid model at 32 mm displacement.



(a) Contact region on bottom plate (b) Contact region on top plate

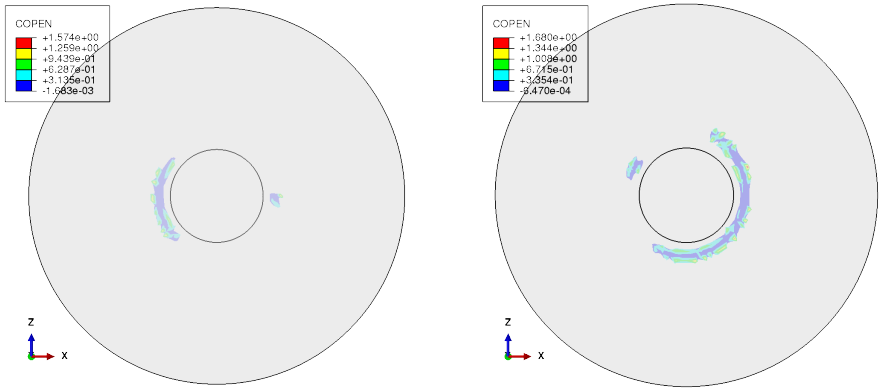


(c) Reaction force for bottom plate (d) Reaction force for top plate

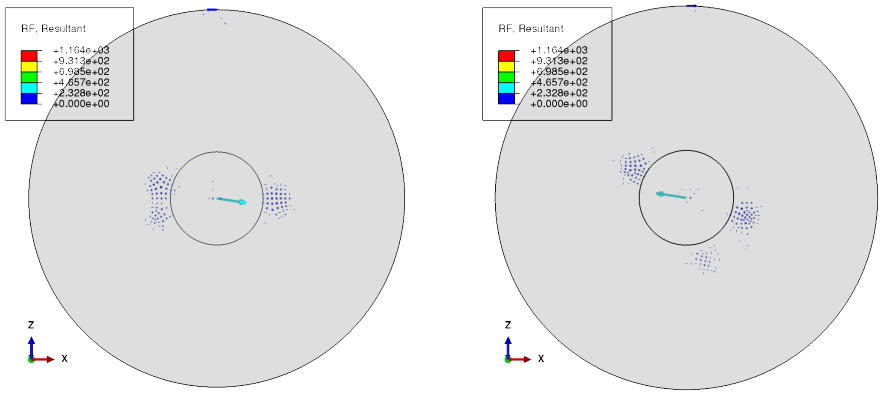


(e) Deformation shape of spring

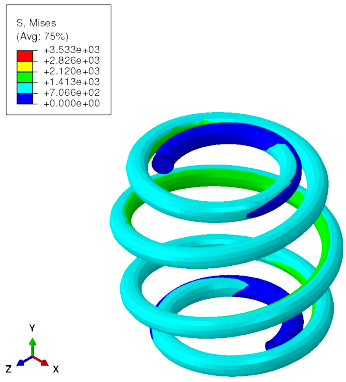
Figure 3.7. Specimen 1: Solid model at 58 mm displacement.



(a) Contact region on bottom plate (b) Contact region on top plate

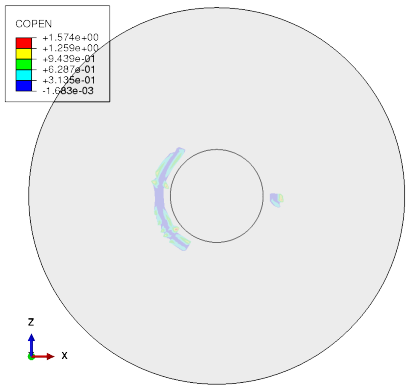


(c) Reaction force for bottom plate (d) Reaction force for top plate

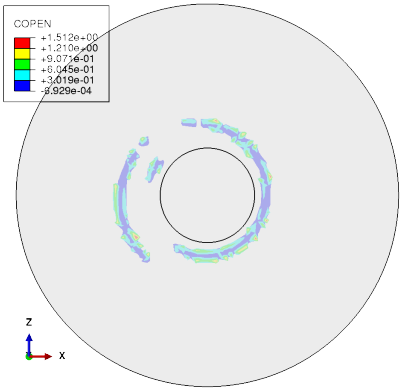


(e) Deformation shape of spring

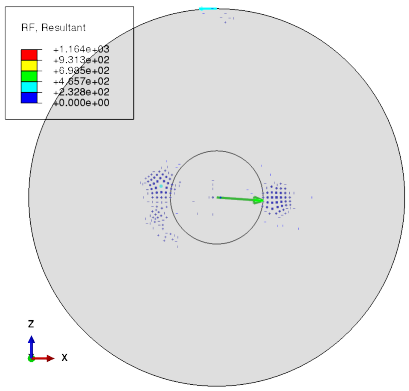
Figure 3.8. Specimen 1: Solid model at 84 mm displacement.



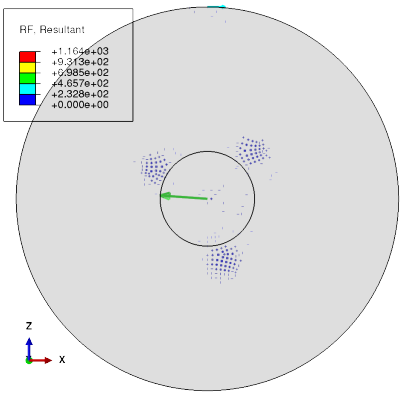
(a) Contact region on bottom plate



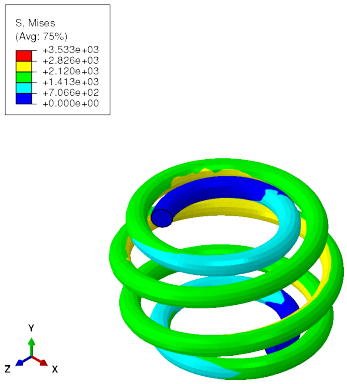
(b) Contact region on top plate



(c) Reaction force for bottom plate

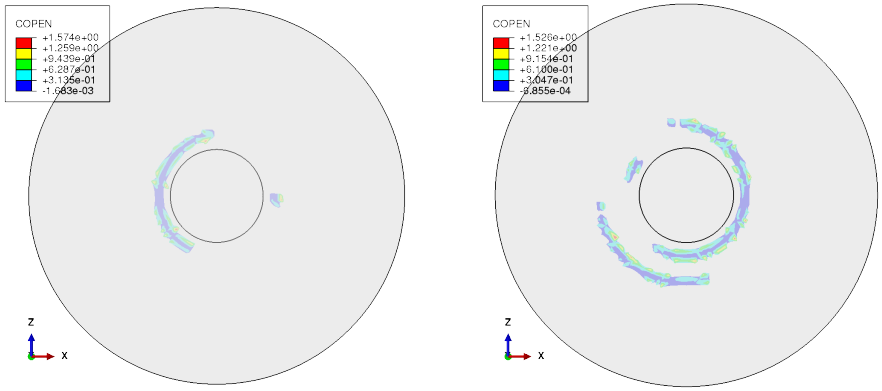


(d) Reaction force for top plate



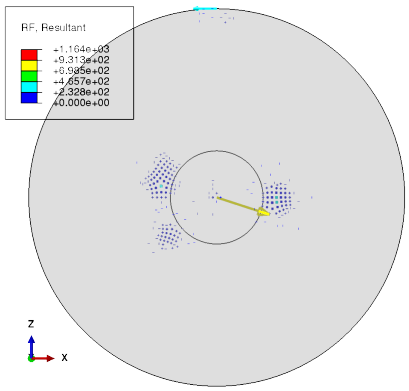
(e) Deformation shape of spring

Figure 3.9. Specimen 1: Solid model at 124 mm displacement.

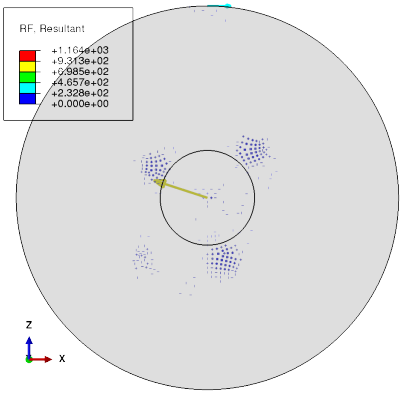


(a) Contact region on bottom plate

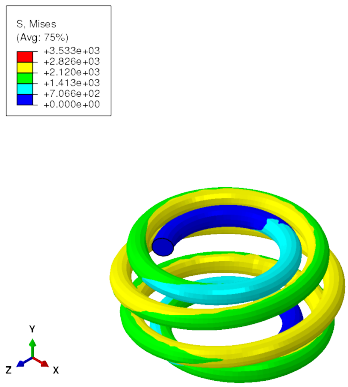
(b) Contact region on top plate



(c) Reaction force for bottom plate

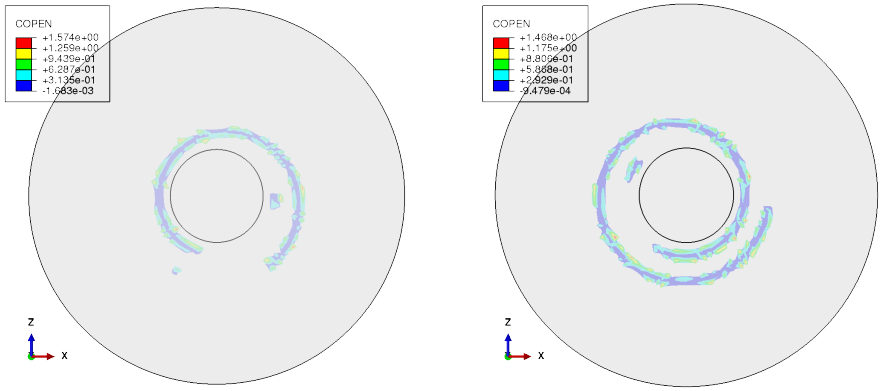


(d) Reaction force for top plate



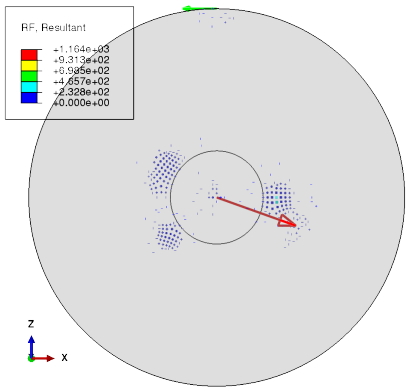
(e) Deformation shape of spring

Figure 3.10. Specimen 1: Solid model at 145 mm displacement.

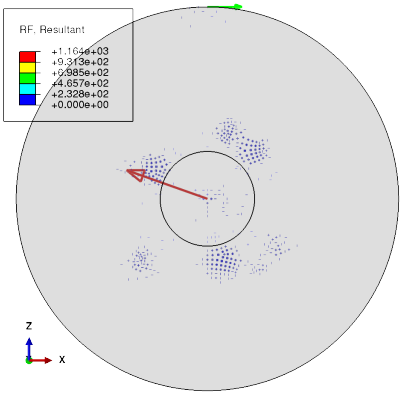


(a) Contact region on bottom plate

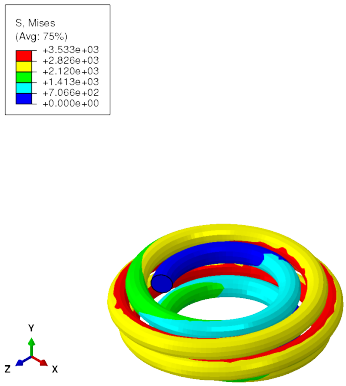
(b) Contact region on top plate



(c) Reaction force for bottom plate

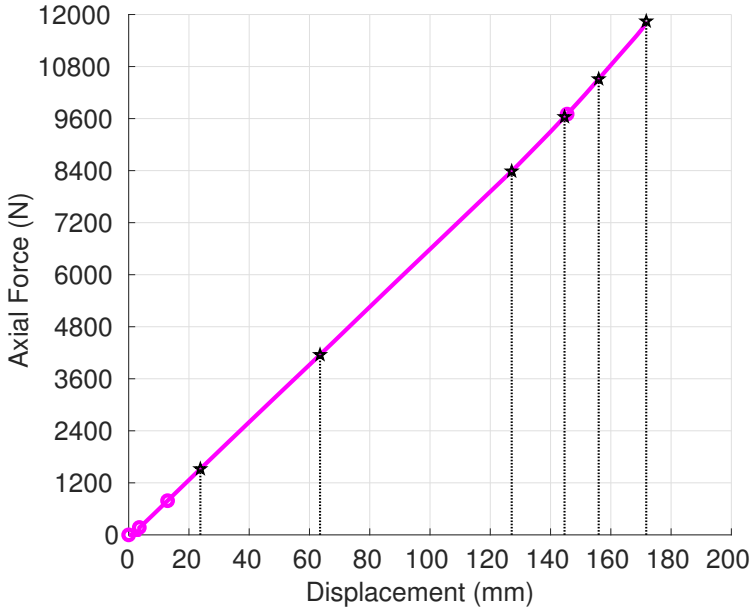


(d) Reaction force for top plate

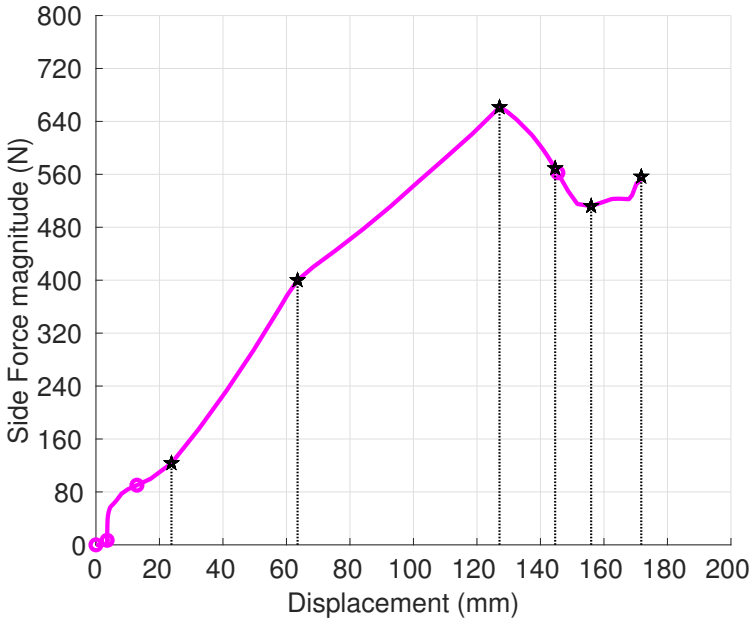


(e) Deformation shape of spring

Figure 3.11. Specimen 1: Solid model at 173 mm displacement.

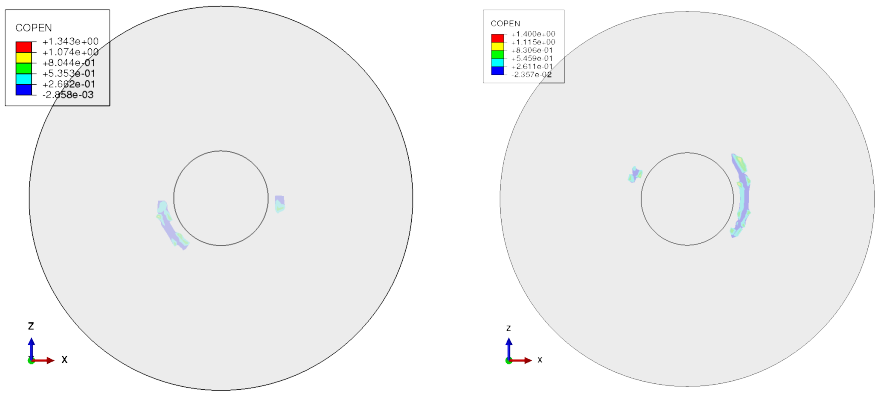


(a) axial force



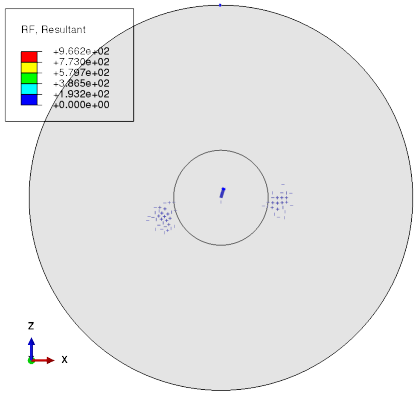
(b) side force magnitude

Figure 3.12. Specimen 5: Displacement locations selected for solid model plots.

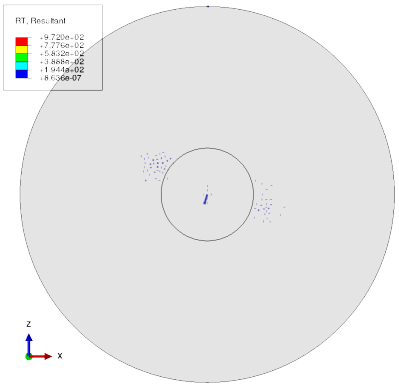


(a) Contact region on bottom plate

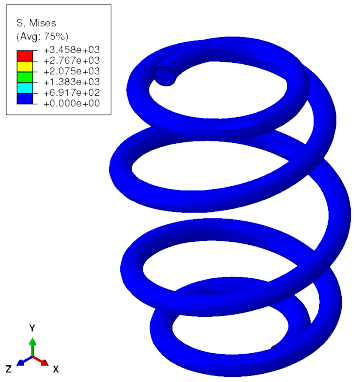
(b) Contact region on top plate



(c) Reaction force for bottom plate

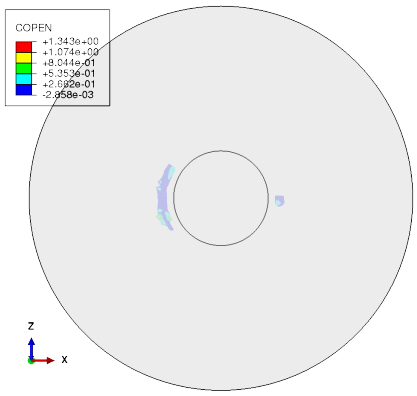


(d) Reaction force for top plate

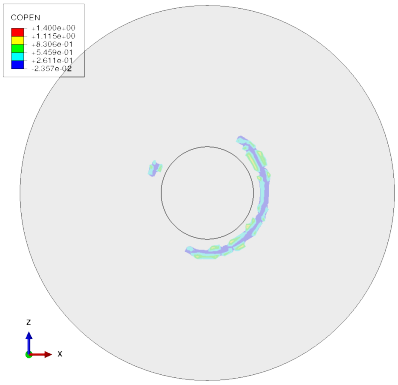


(e) Deformation shape of spring

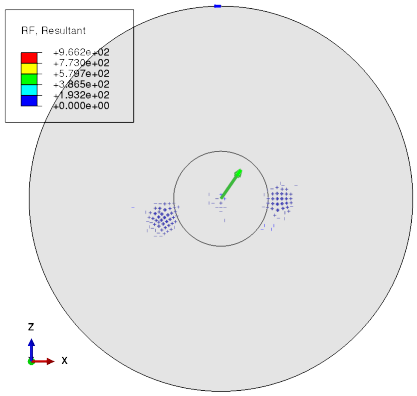
Figure 3.13. Specimen 5: Solid model at 24 mm displacement.



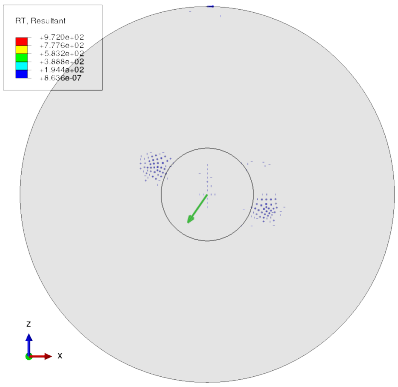
(a) Contact region on bottom plate



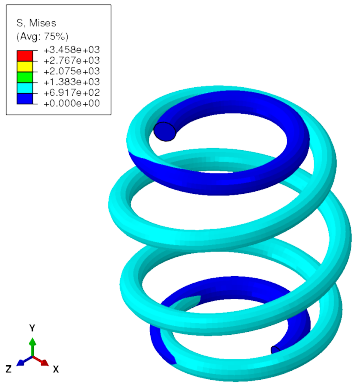
(b) Contact region on top plate



(c) Reaction force for bottom plate

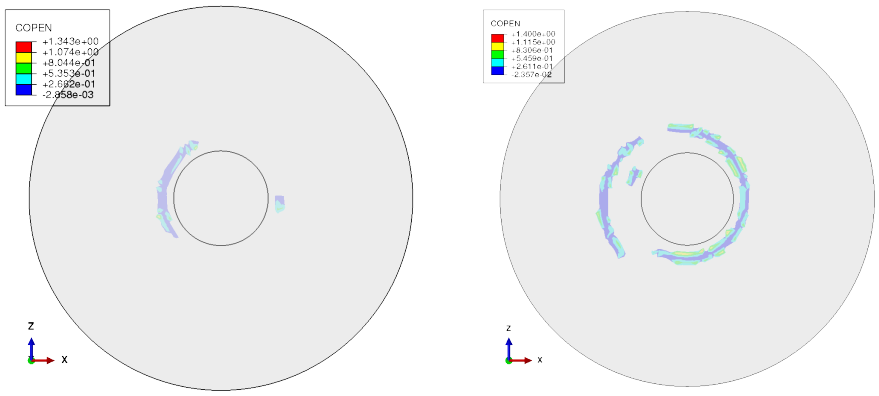


(d) Reaction force for top plate



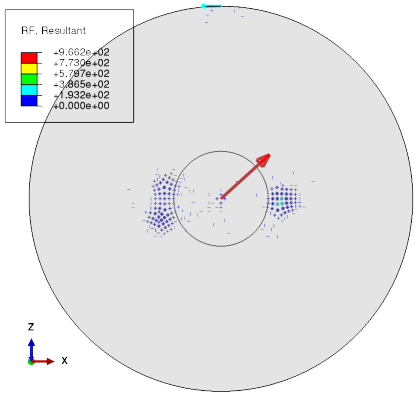
(e) Deformation shape of spring

Figure 3.14. Specimen 5: Solid model at 60 mm displacement.

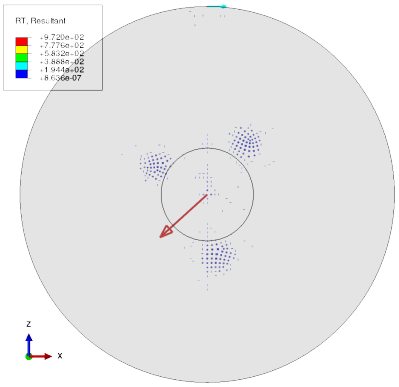


(a) Contact region on bottom plate

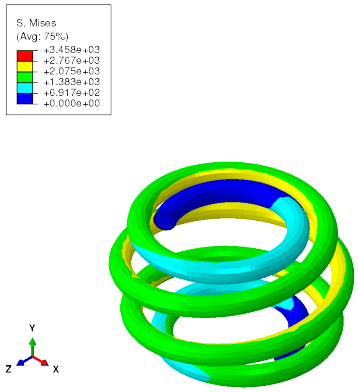
(b) Contact region on top plate



(c) Reaction force for bottom plate

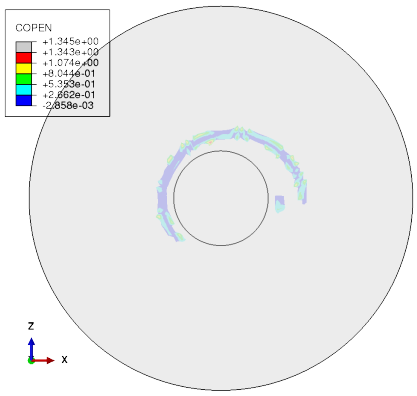


(d) Reaction force for top plate

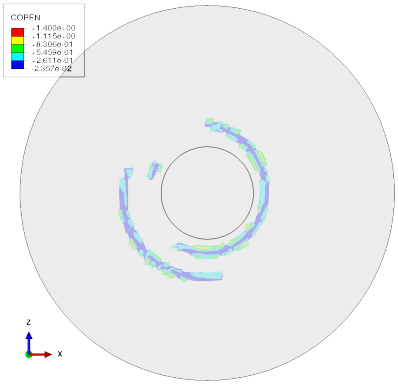


(e) Deformation shape of spring

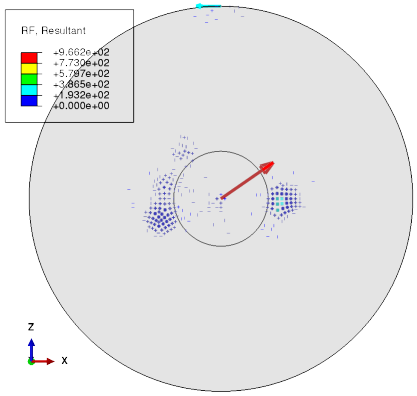
Figure 3.15. Specimen 5: Solid model at 118 mm displacement.



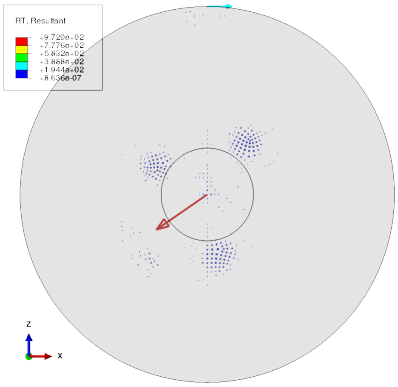
(a) Contact region on bottom plate



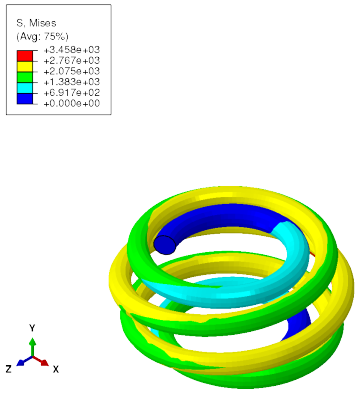
(b) Contact region on top plate



(c) Reaction force for bottom plate

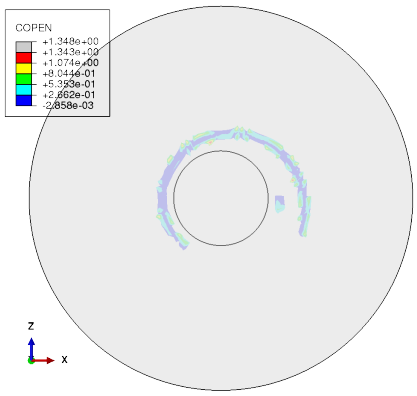


(d) Reaction force for top plate

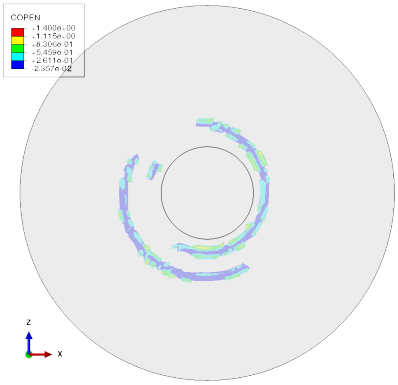


(e) Deformation shape of spring

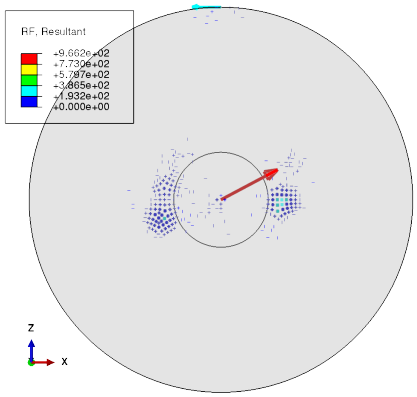
Figure 3.16. Specimen 5: Solid model at 142 mm displacement.



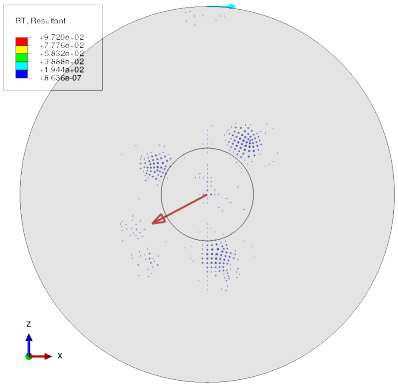
(a) Contact region on bottom plate



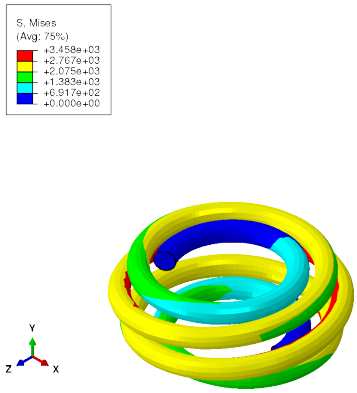
(b) Contact region on top plate



(c) Reaction force for bottom plate

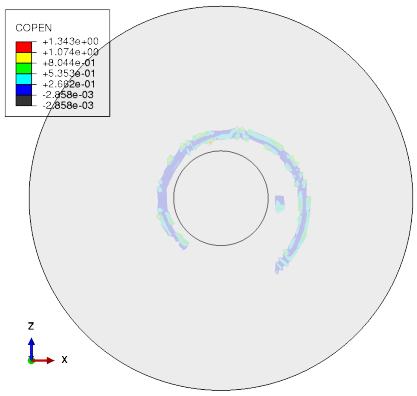


(d) Reaction force for top plate

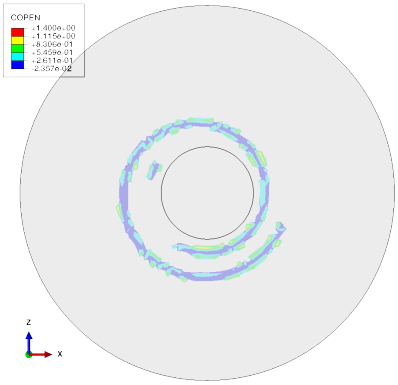


(e) Deformation shape of spring

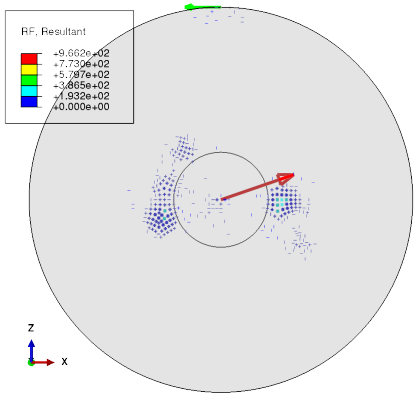
Figure 3.17. Specimen 5: Solid model at 152 mm displacement.



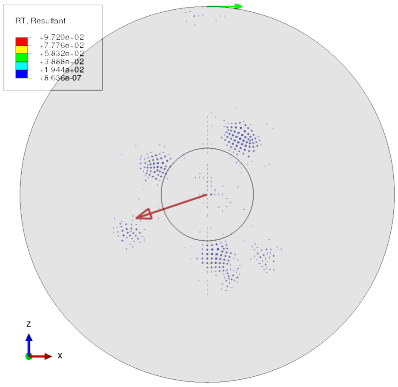
(a) Contact region on bottom plate



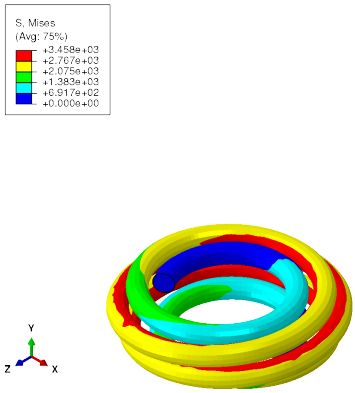
(b) Contact region on top plate



(c) Reaction force for bottom plate

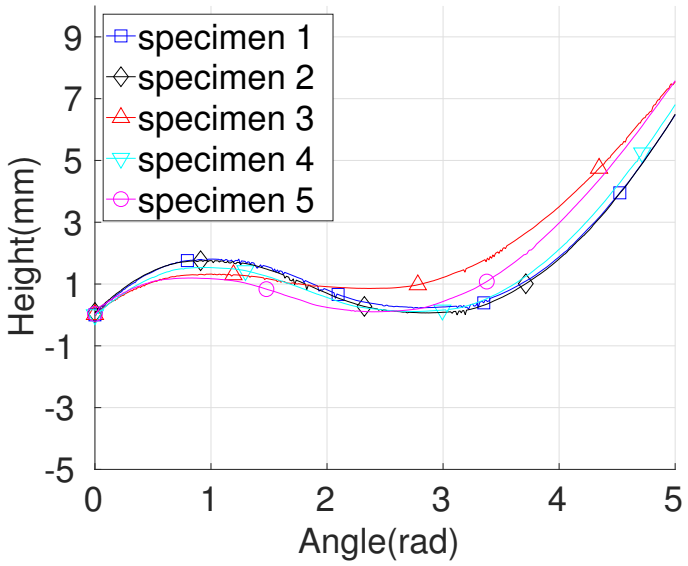


(d) Reaction force for top plate

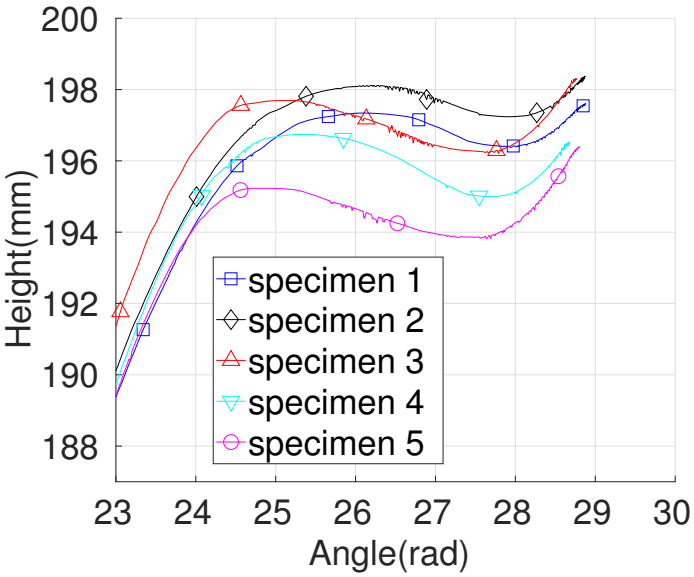


(e) Deformation shape of spring

Figure 3.18. Specimen 5: Solid model at 172 mm displacement.



(a) starting turn



(b) ending turn

Figure 3.19. Zoomed-in plots for height profiles of all specimens at the two ends of the springs.

3.2 Reduced-order Model using Beam Elements and Connectors

The key idea here is to obtain a simplified model which will have smaller degree of freedom compared to a continuum model and thus would be computationally cheaper. In this model we have taken hints from the work presented by Shimoseki et al. in [20] by using beam elements with springs with the two exceptions. The first difference in this approach is that both axial & side forces are being simulated using springs whereas in [20] only axial force was simulated. The Second difference is that non-linear springs which are designed to mimic the contacts are used in this model.

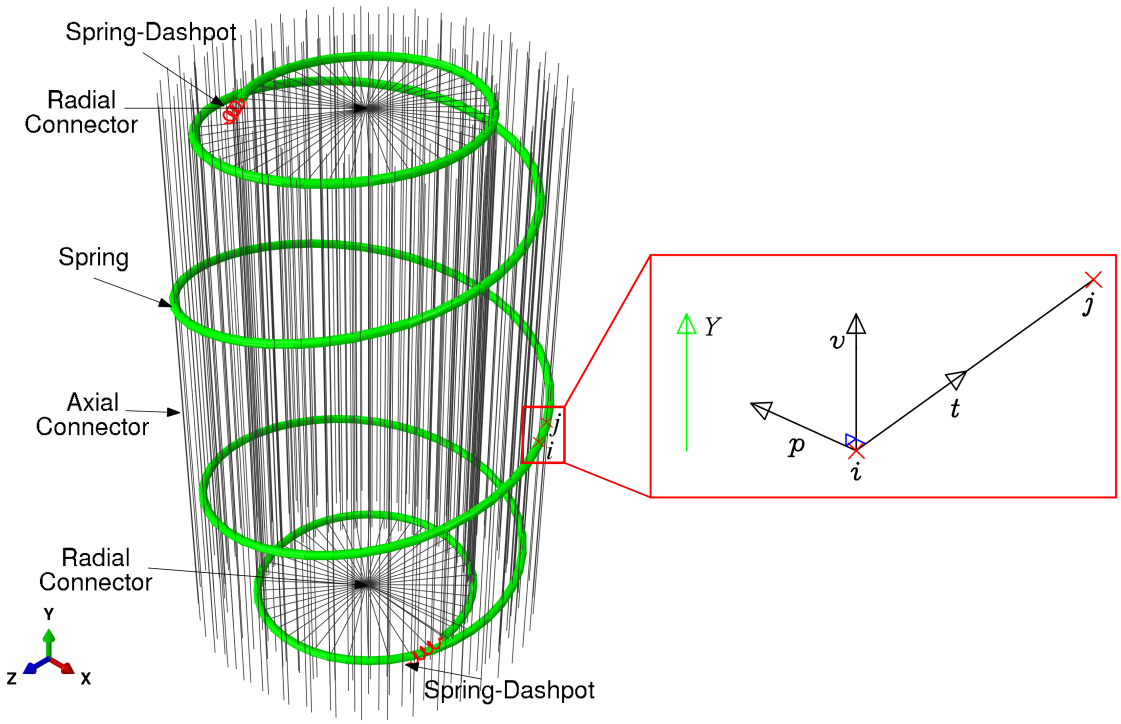


Figure 3.20. Beam connector model with rendered beam profile (0.2x) and annotations labeling the connectors, spring-dashpot elements & the barrel spring. Vector diagram (inset) depicting method for estimating orientation vector ($\boldsymbol{p}$) for beam element along $\boldsymbol{t}$ between nodes i and j (in red). $\boldsymbol{p}$ is estimated using $\boldsymbol{v}$ (along global Y) as $\boldsymbol{p} = \boldsymbol{t} \times \boldsymbol{v}$.

3.2.1 Model Assumptions

It should be noted that while defining the model, certain modeling assumptions were made. The assumptions made for the continuum model as discussed in Section 3.1.1 are equally applicable for this model. Additionally, self-contact between the beams was not modeled. Furthermore, as brought out by the results of continuum model in Section 3.1.6, only the first and last full turn of the spring is assumed to contribute to the side-forces.

3.2.2 Geometry & Type of Elements

In the discussion to follow we will be using the global coordinates system and notations defined in Section 2.1. As discussed earlier, the geometry of the spring is generated using the center-line coordinates of the spring. These points, $\{\boldsymbol{x}_i\}_{i=1}^{N}$, are used to create a wire feature in `ABAQUS` [47] to define the spring. As for the plate and the retainer, we don't create any parts for them separately and rather try to define the geometric constraints offered by them, i.e. normal and tangential contact, by using connector elements in `ABAQUS` [47] .

In this model we are using 3D quadratic Timoshenko beam elements (B32) to model the spring with up-to 500 elements to represent the whole length of the spring. As these are 3D beam elements so we need to specify their orientation vector to fully define them. It is important to note that since geometry of the spring is defined by the profile curves which cannot be expressed analytically, we do not have an analytical expression for the orientation vector for the beam element.

To find the orientation vector for the beam element connecting the i^{th} and j^{th} nodes, we make use of the prior information that none of the beam elements will have a tangential direction parallel to the spring's central axis (global Y axis for our model). With this prior we can set the global Y direction as our reference vector $\boldsymbol{v}$, we can then find the tangential vector,$\boldsymbol{t}$, by using the two coordinates of the nodes of a beam element, thus $\boldsymbol{t} = (\boldsymbol{x}_j - \boldsymbol{x}_i)$. Then by taking a cross product of these two

vectors $\boldsymbol{p} = \boldsymbol{t} \times \boldsymbol{v}$, the vector $\boldsymbol{p}$ will be perpendicular to the tangential vector $\boldsymbol{t}$ and can be chosen as a candidate for the orientation of the beam element.This is also explained pictorially in Fig 3.20.

The beam elements are connected to non-linear spring elements as shown in Fig 3.20 with the help of connectors from `ABAQUS` [47] . The connector chosen for our model was a slot connector as defined in abaqus user manual [47] with non-linear behavior. In this model two types of connectors namely radial and axial connectors are defined. As the name suggests, these connectors are used to simulate the radial and axial force transfer from the spring to the plate. These elements are required to have a property such that they can mimic the contact of the spring with the plates and retainer. This is done by defining a non-linear elastic property for the connector as shown in figure 3.21(b)&(d).

In order to define the connectors we need to specify its end points. For the axial connector the end points will be point on the wire, $\boldsymbol{x}_i$, and its corresponding projection on XZ plane, at both Y coordinates corresponding to the top surface of the bottom plate (Y = t) and at bottom surface of the top plate (Y = $H + d + t$) as shown in Fig 3.21(a). Therefore the top and bottom endpoints of axial spring is given by:

$$\text{bottom end points:} \quad \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + \frac{d}{2})\boldsymbol{e}_2 \quad \forall i = \{1, 2, .., N\} \tag{3.20}$$

$$\text{top end points:} \quad \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + d + H)\boldsymbol{e}_2 \quad \forall i = \{1, 2, .., N\} \tag{3.21}$$

Similarly for defining the radial connector we have to specify two points which are the center of the bottom ($\boldsymbol{x}_b$) or top ($\boldsymbol{x}_t$) retainer and the spring point ($\boldsymbol{x}_i$) as shown in Fig 3.21(c). The bottom and top retainer centers are given as follows:

$$\text{bottom center:} \quad \boldsymbol{x}_b = (t + \frac{d}{2})\boldsymbol{e}_2 \tag{3.22}$$

$$\text{top center:} \quad \boldsymbol{x}_t = (t + d + H)\boldsymbol{e}_2 \tag{3.23}$$

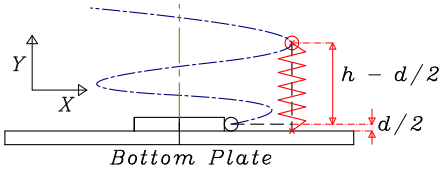
It is important to add that we have chosen to place the connectors such that all the beam nodes will have two axial connectors for the bottom and top plates and

depending on the angular position, θ_i , of the point it will either have or will not have a radial connector. As it was observed from the model results discussed in Section 3.1.6 that the reaction forces on the plates were predominantly present only at certain zones which were mainly in the first and last full turn of the spring, therefore we have placed the radial connectors only for the points ($\boldsymbol{x}_i$) which are on the first and last full turn of the spring. Also, please note that ABAQUS [47] calls the end points as reference points we will use this nomenclature interchangeably.

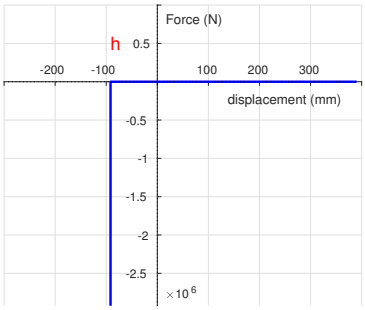
For axial connectors we want them to offer stiffness to the point, $\boldsymbol{x}_i$, only when the deflection of the point is more than h_i, this would mean that when the displacement of the point crosses this threshold then the point shall come into contact with the plate and thereafter should experience very high stiffness (K_a) so as to not penetrate into the plate. Also, the spring should have zero resistance before crossing such a threshold as it is allowed to move freely in that space. The combination of such a property can be seen in the plot shown for axial connector in Fig 3.21(b). The next obvious question would be that such a constraint should only be placed on the Y component of the displacement, this is achieved by setting boundary conditions for the other end of the connector which is on the plate such that it is free to move in X and Z directions.

For the radial connectors we want them to simulate both the normal contact offered when a spring touches the retainer and also the tangential friction offered by the plate to the spring. This is achieved by specifying very high stiffness (K_c) for the connector in the compressive direction, when it would touch the retainer, and some stiffness (K_t) in the tensile direction as shown in Fig 3.21(d). Therefore, it becomes important to pick the correct stiffness values for these parameters.

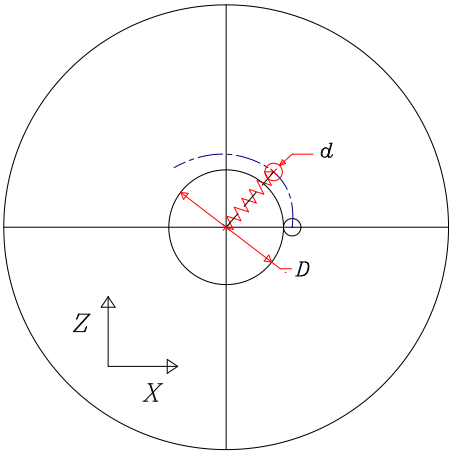
In addition to the above elements, we have defined two spring-dashpot elements, hereon called as end springs, in ABAQUS [47] to eliminate rigid body rotation of barrel spring along the Y axis. We have used two springs (as shown in Fig 3.20) attached to the end points ($\boldsymbol{x}_1$ & $\boldsymbol{x}_N$) of the barrel spring such that the spring-dashpot element gets connected tangentially. To define the geometry of this element we need



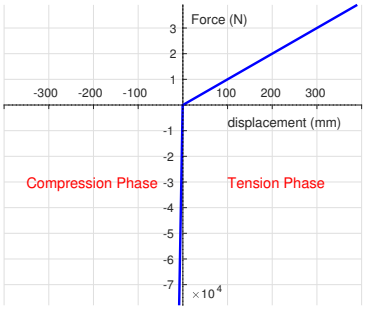
(a) axial connector



(b) axial connector response



(c) radial connector



(d) radial connector response

Figure 3.21. Stiffness definition for connectors. *(a)* Axial connector (spring in red) connecting a node on spring at height h from plate to a its projection on top surface of bottom plate and similarly we will have another connection to the bottom surface to top plate *(b)* Stiffness for axial connectors becomes non-zero only when the node displaces more than its un-deformed height (h) *(c)* Radial connector (spring in red) connecting a node on spring to center of bottom plate *(d)* Stiffness (slope) for radial connector is higher in compression to simulate contact with retainer and is lower in tension to simulate friction.

to specify the end points of the element, which would be the end point of the spring

and a translated point in the tangential direction at the same height as that of the end point. The translated points for the two ends are given by:

$$\text{translated point at starting face:} \quad \boldsymbol{x}_s = \boldsymbol{x}_1 + a\frac{(\boldsymbol{x}_1 - \boldsymbol{x}_2)}{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2} \tag{3.24}$$

$$\text{translated point at end face:} \quad \boldsymbol{x}_e = \boldsymbol{x}_N + a\frac{(\boldsymbol{x}_N - \boldsymbol{x}_{N-1})}{\|\boldsymbol{x}_N - \boldsymbol{x}_{N-1}\|_2} \tag{3.25}$$

Where a can be a user defined constant. We have chosen $a = 20$ mm for our model. It should be noted that the value of a does not affect the model response in any way. Furthermore, we would want that the end springs should have very low stiffness so that they do not affect our solution and just restrict the rigid body motion. We have chosen to define its stiffness as $K = 1$ N/mm, with such a choice the magnitude of spring force encountered by the end spring was on average 1% of the magnitude of side-force and therefore the end springs will not affect the solution significantly.

3.2.3 Material Properties

We need to define the sectional properties for the beam elements which will need the radius of the spring cross-section to calculate the moment of inertia, also the model needs the information of the Young's Modulus of the spring material and its Poisson ratio. As discussed earlier, we don't have a reliable information for the material properties especially the Young's modulus. As for now we will assume it to be 270 GPa. We will need to tune this parameter with the help of experimental results.

3.2.4 Boundary Conditions

The boundary conditions defined for the model will be with respect to the coordinate system defined earlier in Section 2.1. We would also use the sequence of points $\{\boldsymbol{x}_i\}_{i=1}^{N}$ which define the center line of the spring. The boundary conditions for the model are defined as follows:

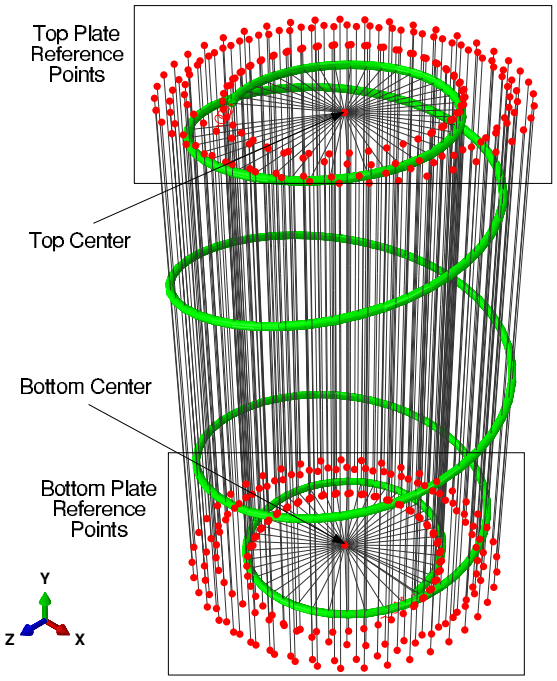


Figure 3.22. Reference points annotations indicating the location of top & bottom center and top & bottom plate reference points.

1. To simulate the testing procedure all dofs of the bottom center (shown in Fig 3.22) are fixed to zero.

$$\boldsymbol{u}(\boldsymbol{x}_b) = \mathbf{0} \tag{3.26}$$

Where $\boldsymbol{x}_b$ is from Equation (3.22)

2. To simulate loading the top center (shown in Fig 3.22) point is given a fixed displacement.

$$\boldsymbol{u}(\boldsymbol{x}_t) = -\alpha \boldsymbol{e}_2 \tag{3.27}$$

Where $\boldsymbol{x}_t$ is from Equation (3.23). The magnitude of the displacement is such that the spring height at max displacement shall be 1.5 inches.

$$\alpha = (H + d - 1.5 * 25.4) \tag{3.28}$$

3. To simulate fixed position of bottom plate, zero displacement is given for bottom plate reference points (shown in Fig 3.22) in the Y direction and the points are free to move in X and Z directions.

$$\boldsymbol{u}(\boldsymbol{x}) \cdot \boldsymbol{e}_2 = 0 \quad \forall \quad \boldsymbol{x} \in S_b \tag{3.29}$$

$$\text{where,} \quad S_b = \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + \frac{d}{2})\boldsymbol{e}_2 \quad \forall \quad i = \{1, 2, .., N\}\} \tag{3.30}$$

4. To simulate movement of top plate, fixed displacement is given for top plate reference points (shown in Fig 3.22) in the Y direction and the points are free to move in the X and Z directions.

$$\boldsymbol{u}(\boldsymbol{x}) \cdot \boldsymbol{e}_2 = -\alpha \quad \forall \quad \boldsymbol{x} \in S_t \tag{3.31}$$

$$\text{where,} \quad S_t = \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}_i + (t + d + H)\boldsymbol{e}_2 \quad \forall \quad i = \{1, 2, .., N\}\} \tag{3.32}$$

5. To resist rigid body rotation of the spring about the Y axis, the starting point of the spring-dashpot element (shown in Fig 3.20) is given zero displacement in the Z direction and the point is free to move in X and Y directions.

$$\boldsymbol{u}(\boldsymbol{x}_s) \cdot \boldsymbol{e}_1 = 0 \tag{3.33}$$

   Where $\boldsymbol{x}_s$ is defined in Equation (3.24)

6. To resist rigid body rotation of the spring about the Y axis, the ending point of the spring-dashpot element (shown in Fig 3.20) is given a fixed displacement of α in negative Y direction with zero displacement in X and Z directions.

$$\boldsymbol{u}(\boldsymbol{x}_e) = -\alpha \boldsymbol{e}_2 \tag{3.34}$$

   Where $\boldsymbol{x}_e$ is defined in Equation (3.25) and α is from Equation (3.28).

It should be noted that, in the above discussion for boundary conditions, whenever any component of displacement, $\mathbf{u}$, is not assigned any value then it is assumed to have a free dof.

3.2.5 Analysis & Post-Processing

We are carrying out static simulations with geometric non-linearity. As there are non-linear connectors involved in this model and we have large deformations which makes convergence of the model hard. In order to aid the convergence we are using automatic stabilization with volume proportional damping in `ABAQUS` [47] which is described in detail in Section 3.1.5. Due to damping we would again expect to experience the artifact of insignificant reaction force response even when the spring is being compressed for the initial phase of loading.

After the simulation is finished we extract the value of displacement and reaction forces at all of the reference points. It is important to discuss at what node locations/ reference point locations these quantities are extracted, the details are as follows:

- Sideforce: Reaction force coming at the bottom center reference point ($\boldsymbol{x}_b$).
- Displacement: Value of displacement at the top center reference point ($\boldsymbol{x}_t$).
- Axialforce: Sum of axialforce (along Y direction) for all the bottom reference points ($x \in S_b$)

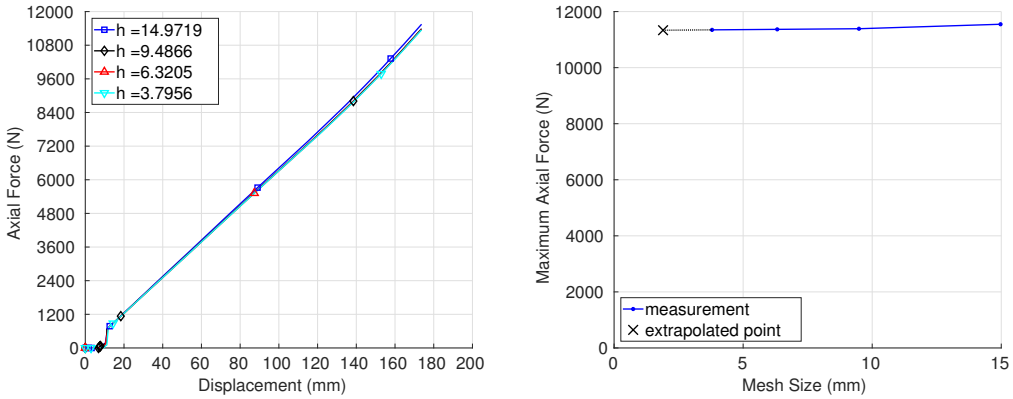
3.2.6 Model Results

Prior to discussion of results of the model, we need to prove the convergence of our model, for this task we have fixed the value of parameters as:

- Young's Modulus of spring material: $E = 225$ GPa
- Stiffness of axial connectors: $K_a = 10^6$ N/mm
- Stiffness of radial connectors in compression: $K_c = 10^4$ N/mm
- Stiffness of radial connectors in tension: $K_t = 100$ N/mm

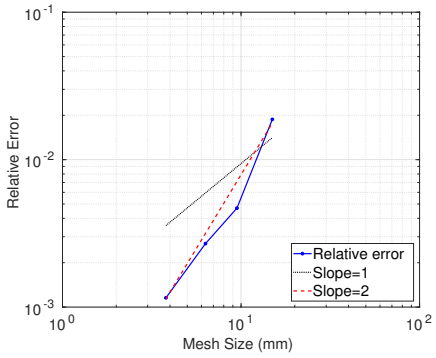
After fixing the values for the above parameters, to prove convergence we need to check if a particular metric converges with respect to mesh size. As, we want our model to

first correctly predict the axial response and then the side-force response, therefore we have chosen the maximum value of axial force as our metric for convergence. We have carried out simulations for decreasing mesh sizes, the results are shown in Fig 3.23.



(a) axial force

(b) convergence plot



(c) Log-Log plot

Figure 3.23. Convergence of Beam Model *(a)* Axial response of the model converging to the same sloped line as the mesh size decreases *(b)* Measured value of maximum axial force for different mesh sizes along with the extrapolated point which is computed at half the edge length of finest mesh *(c)* Relative error (using extrapolated point as the assumed true solution) can be observed from the log-log plot to converge at a quadratic rate with respect to the mesh size.

From Fig 3.23(c), we can see that the relative error of maximum axial force converges with a quadratic rate. Furthermore, for mesh sizes smaller than 14.97 mm the relative errors are less than 1%. Therefore, in order to reduce the computational time for the simulation for the beam model choosing mesh size of 9.48mm will be adequate as it has a relative error of 0.5% and is not computationally expensive as it has only 200 nodes.

As discussed earlier, we need to estimate a value for several parameters for this model. The parameters being E, K_a,K_c and K_t. We have made use of the experimental data to estimate these values in such a way so that our simulations are as close to the experimental data as possible. We have tuned these parameters for one spring specimen (specimen 3) and then evaluate the performance of the parameters with respect to other specimens. When carrying out analysis for different parameter value it was observed that E was the most influential parameter for axial-force and side-force response. To find the best E we did a discrete sweep over a range of candidate values and picked the one which had the highest correspondence with the experimental data.

After estimating the mesh size for converged model, the next step is to calibrate the converged model such that its response matches the experimental data. For the beam connector model there are four parameters, E, K_a,K_c and K_t, that can be tuned in order to calibrate the model. In order to judge the quality of calibration, squared difference of axial force response of the model and experimental data was taken as the metric. To estimate the parameters an array of values were tried for each parameter and the value which had the smallest squared difference was chosen as the tuned value of the parameter. The array of values chosen for E were 200, 225, 250, 275, 300, 325 GPa. It was observed that E was the most sensitive parameter and a change in its value would change the slope of axial force response. The minimum squared difference was observed for a value of $E = 225$ Gpa. The array of values chosen for K_a were 10^4, 10^5, 10^6 N/mm. It was observed that the model response remained un-affected to changes in K_a, however, for smaller values of K_a there was

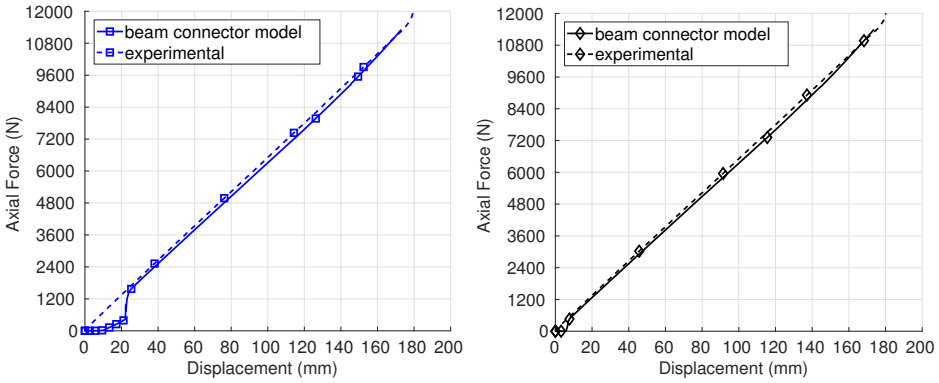
larger penetration of spring into the plates which is in-feasible. Therefore $K_a = 10^6$ N/mm was chosen as the tuned value of K_a. The rest of the parameters, K_c & K_t were chosen heuristically as $K_c = 10^4$N/mm & $K_t = 10^2$ N/mm so as to match the side-force response of the model with the experimental data.

The simulation results for the five test specimens along with their experimental results for converging model with calibrated parameters are shown in Fig 3.24 and Fig 3.25.

From Fig 3.24 and Fig 3.25 we can see that the model is able to produce both axial and side forces which match closely with the experimental results for all the specimens. This result also indicates that the parameter values that we obtained from one particular specimen is able to produce good result for other specimens and thus we can be fairly confident about them. Also, it should be noted that there is an initial nearly-zero force response state for some of the specimens, this artifact is again because we are using automatic stabilization with volume proportional damping to aid in solving of initial simulated contact detection by adding viscous forces to the global equilibrium equations as discussed in Section 3.2.5. This causes an initial nearly-zero response state as the viscous forces dominate over the internal forces.

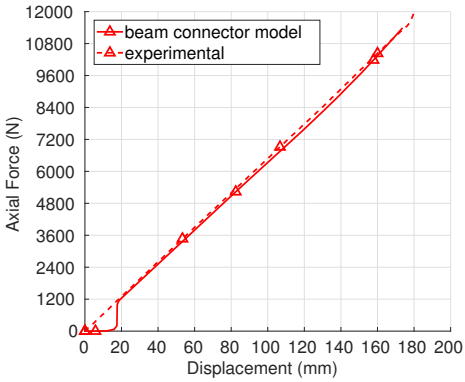
Although the model performs well as shown in figures 3.24 and 3.25, it does not give a complete interpretation of the actual contact mechanism. Since we are simulating the contact with the help of connectors, we can only extract reaction forces at nodes and deformation shape of the barrel spring and do not have any information about the actual contact. A closer look at the results for the specimen 1 and 5 in Fig 3.27-Fig 3.32 and Fig 3.34-Fig 3.39 respectively gives us insights into the behavior of the beam connector model. These plots are generated at specific location along the displacement history of the specimens as shown in Fig 3.26 and Fig 3.33.

Similar to the results of the continuum model, as discussed in Section 3.1.6, the reaction forces for the beam connector model are also concentrated at certain regions as evident from reaction forces plots in Fig 3.27-Fig 3.32 for specimen 1 and Fig 3.34-Fig 3.39 for specimen 5. Also, throughout the majority of loading there are

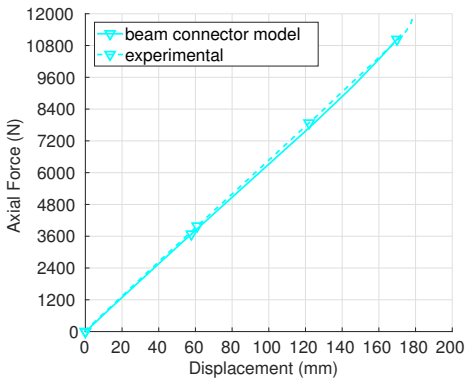


(a) specimen 1

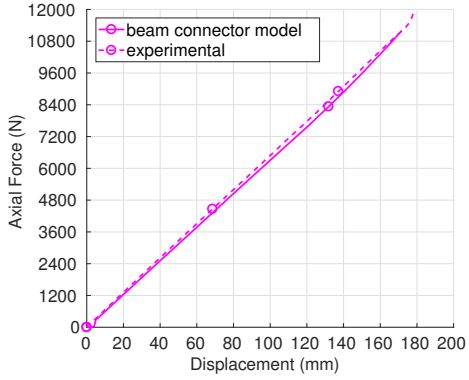
(b) specimen 2



(c) specimen 3

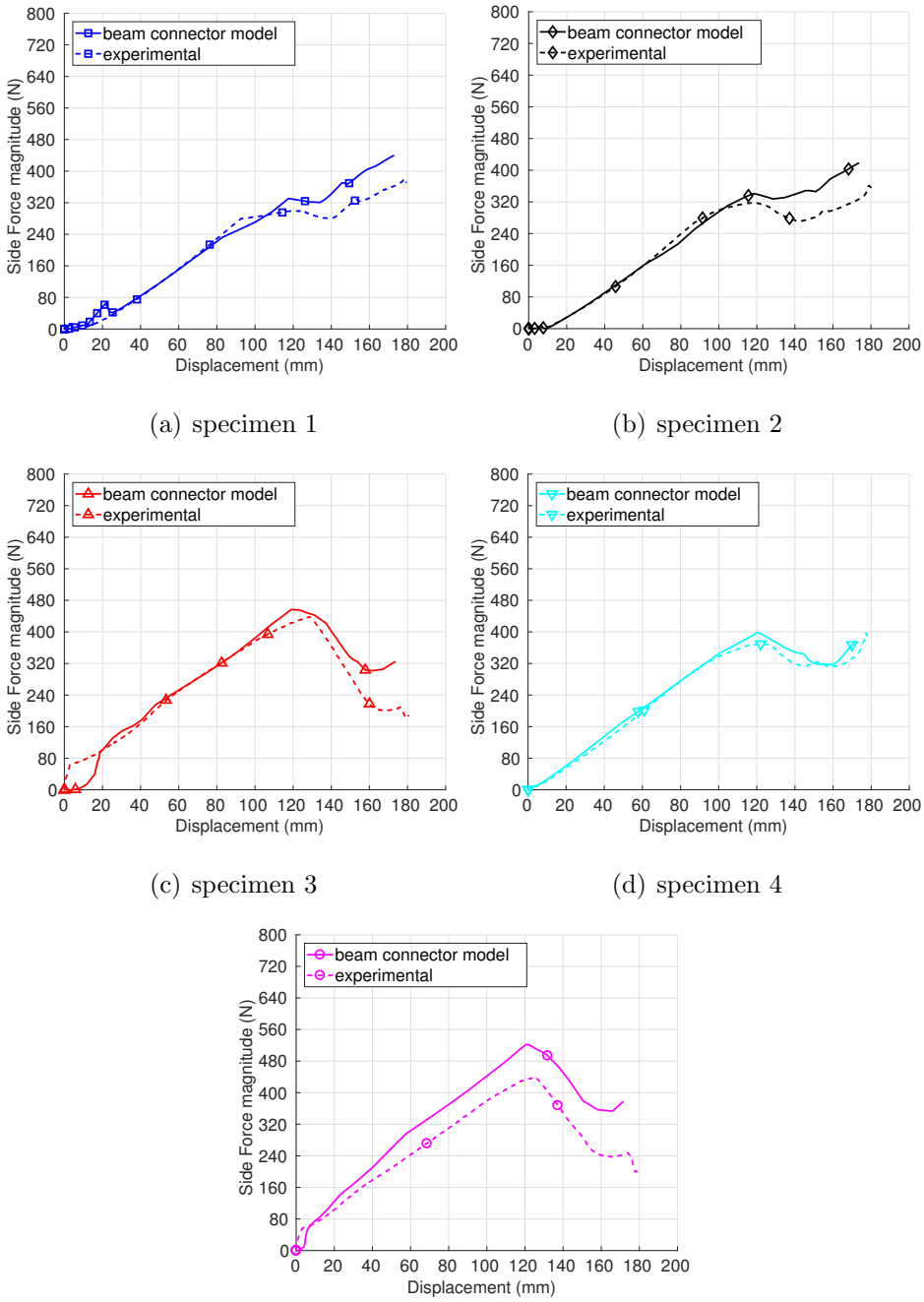


(d) specimen 4



(e) specimen 5

Figure 3.24. Axial force for Beam-Connector Model.



(a) specimen 1

(b) specimen 2

(c) specimen 3

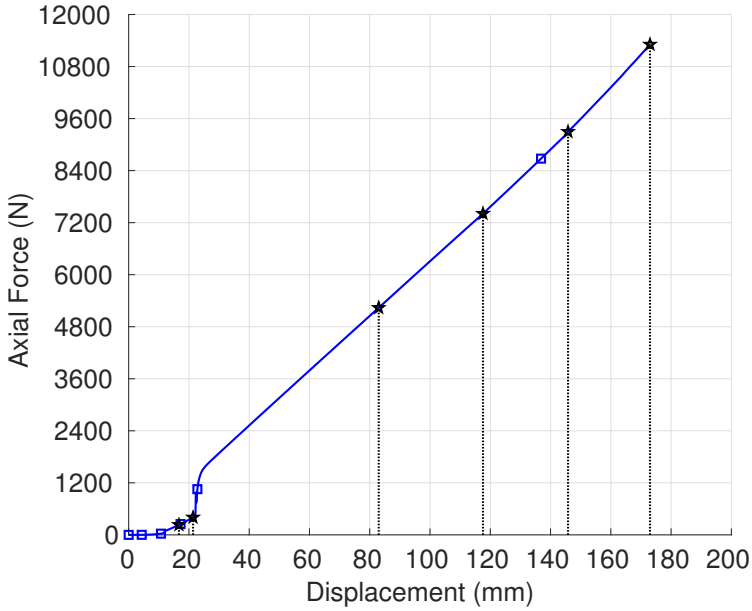
(d) specimen 4

(e) specimen 5

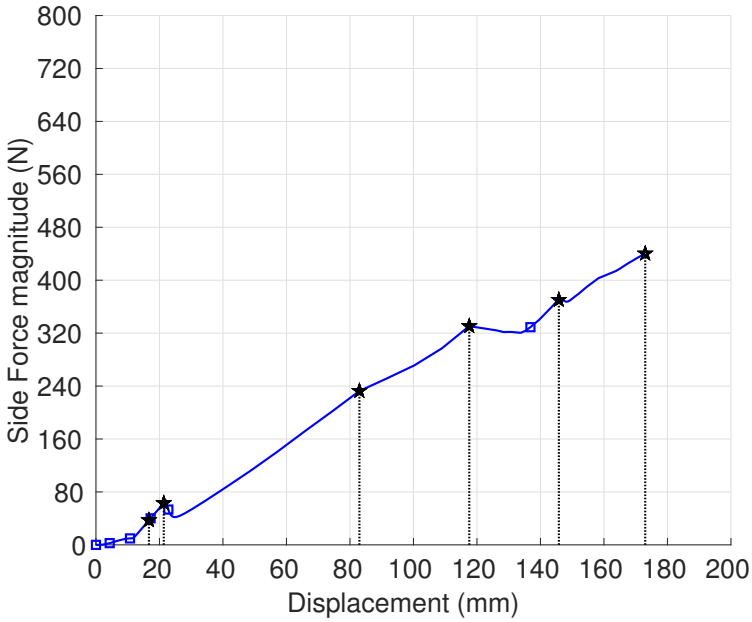
Figure 3.25. Side force for Beam-Connector Model.

at-most 2-3 regions of maximum reaction forces at the two ends of the barrel spring and only towards the end of the loading more regions emerge. Further, the 2 or 3 regions of concentrated forces mainly correspond to the first and last full turn of the spring which indicates that our choice of placing the radial connectors over the first and last full turn will be able to capture the side-force behavior of the barrel spring adequately.

The location of these concentrated reaction forces for specimen 1 and 5 are similar to as discussed in Section 3.1.6. As evident from Fig 3.27(a),Fig 3.28(a) the reaction forces for specimen 1 are concentrated diametrically apart at the first turn of the spring, whereas for specimen 5 they are at an angular position of nearly 150 degrees ($<$ 180 degrees) as shown in Fig 3.34(a), Fig 3.35(a). This can be explained using the height profiles of the two specimens using Fig 3.19 in a similar manner as was done in Section 3.1.6.

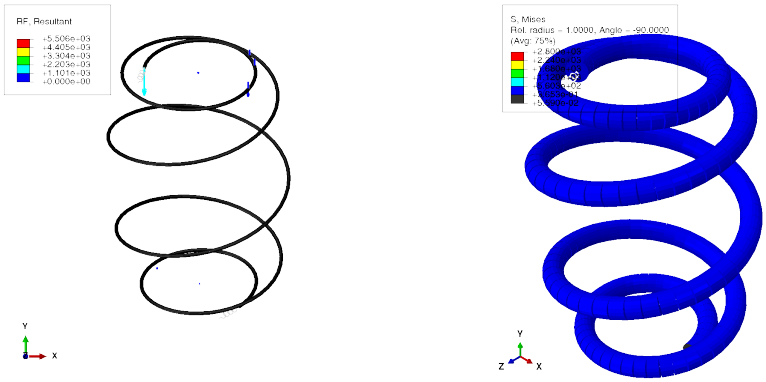


(a) axial force



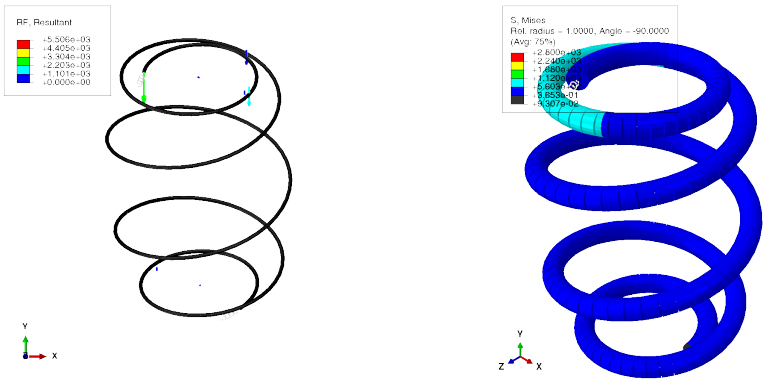
(b) side force magnitude

Figure 3.26. Specimen 1: Displacement locations selected for beam connector model plots.



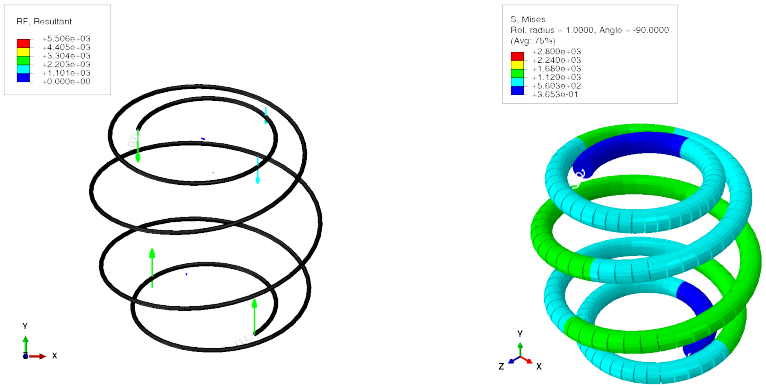
(a) Reaction forces (b) Von-Mises stress

Figure 3.27. Specimen 1: Beam model at 16 mm displacement.



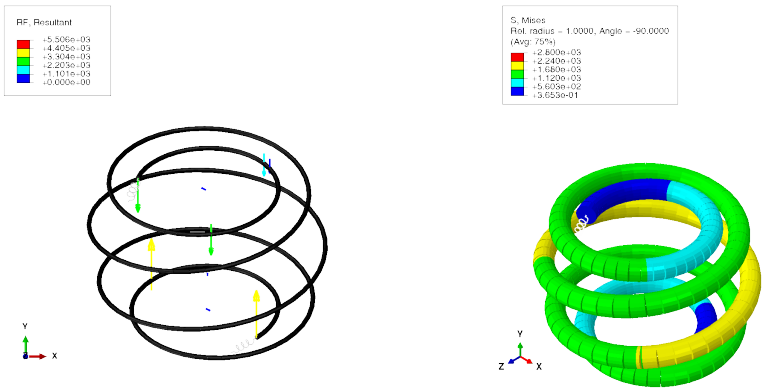
(a) Reaction forces (b) Von-Mises stress

Figure 3.28. Specimen 1: Beam model at 21 mm displacement.



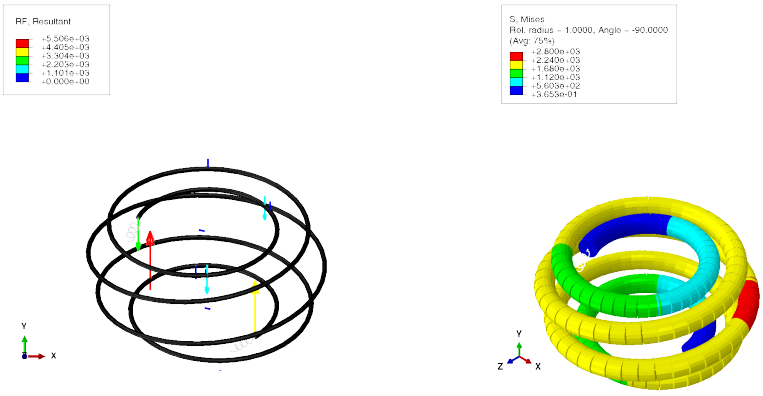
(a) Reaction forces (b) Von-Mises stress

Figure 3.29. Specimen 1: Beam model at 74 mm displacement.



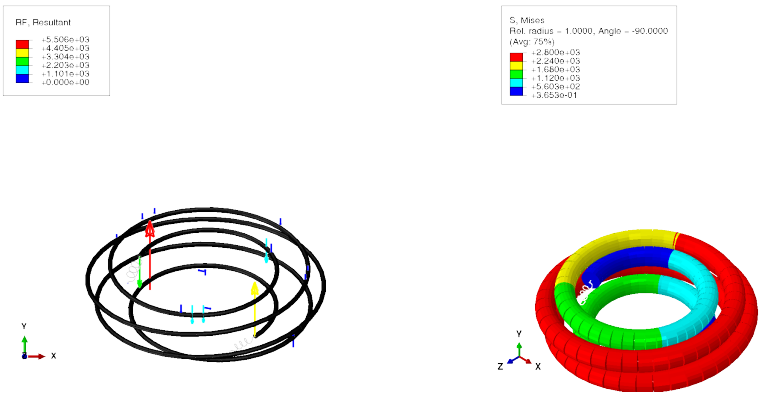
(a) Reaction forces (b) Von-Mises stress

Figure 3.30. Specimen 1: Beam model at 108 mm displacement.



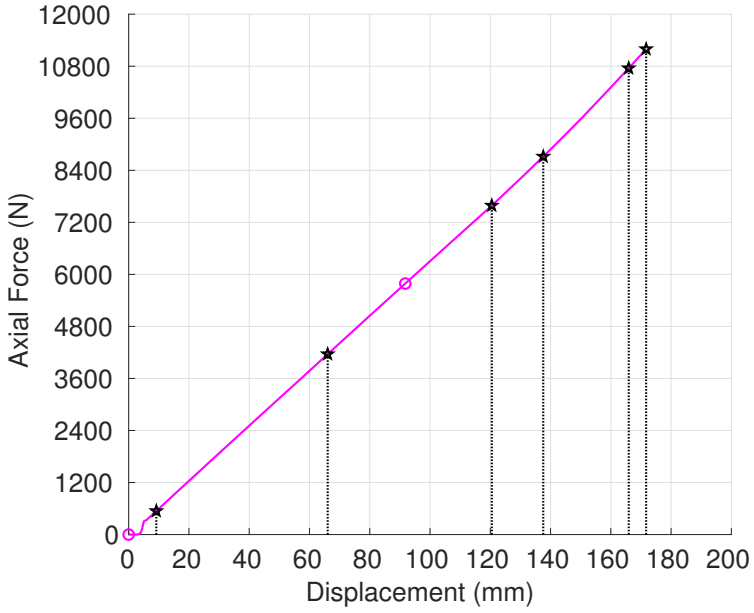
(a) Reaction forces (b) Von-Mises stress

Figure 3.31. Specimen 1: Beam model at 144 mm displacement.

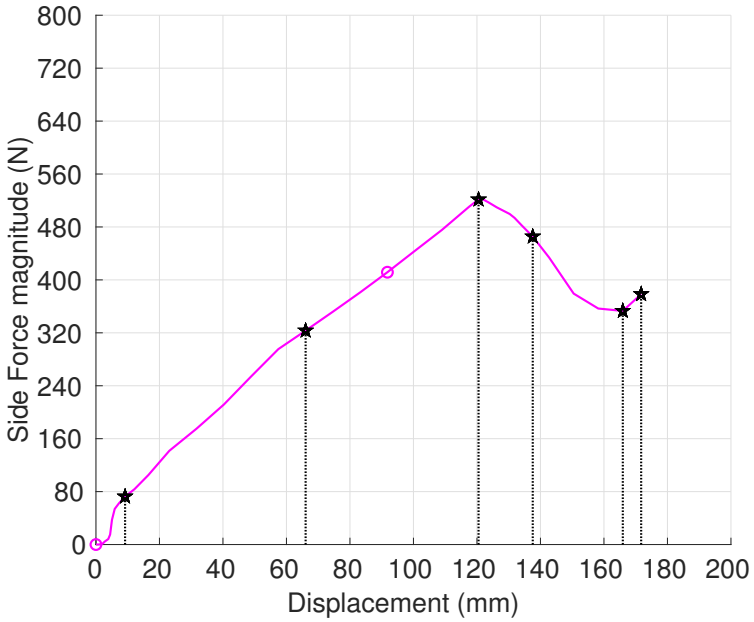


(a) Reaction forces (b) Von-Mises stress

Figure 3.32. Specimen 1: Beam model at 172 mm displacement.

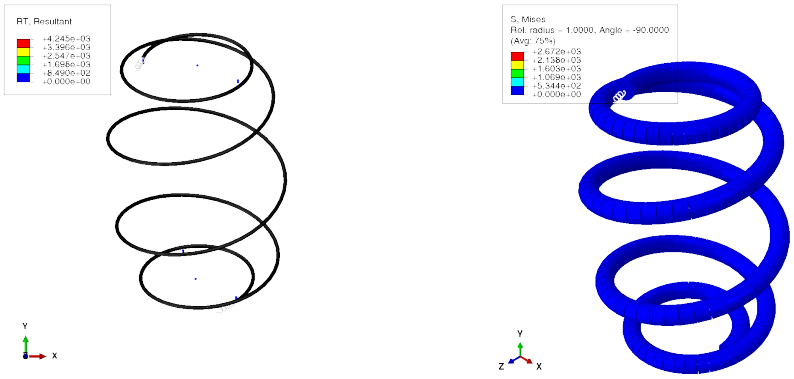


(a) axial force



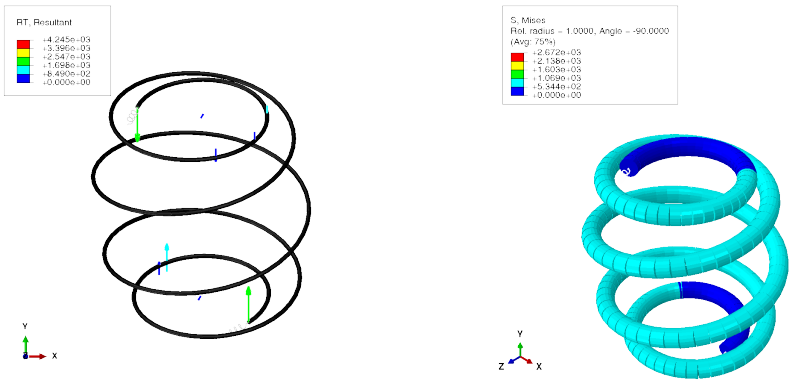
(b) side force magnitude

Figure 3.33. Specimen 5: Displacement locations selected for beam connector model plots.



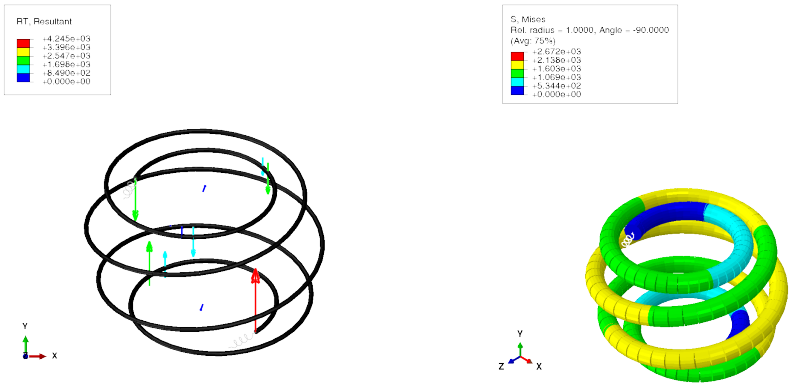
(a) Reaction forces (b) Von-Mises stress

Figure 3.34. Specimen 5: Beam model at 7 mm displacement.



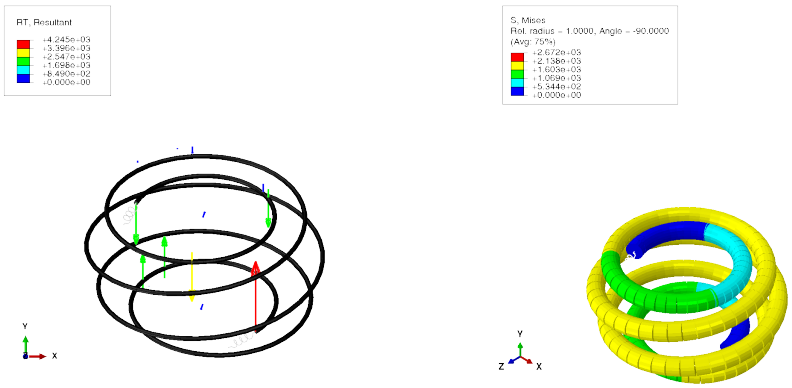
(a) Reaction forces (b) Von-Mises stress

Figure 3.35. Specimen 5: Beam model at 57 mm displacement.



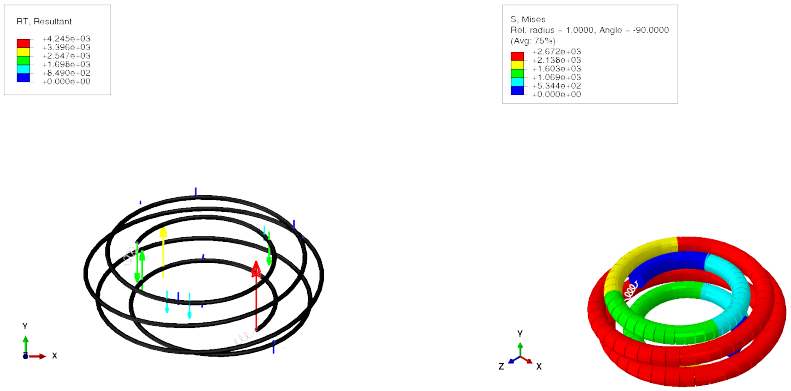
(a) Reaction forces (b) Von-Mises stress

Figure 3.36. Specimen 5: Beam model at 120 mm displacement.



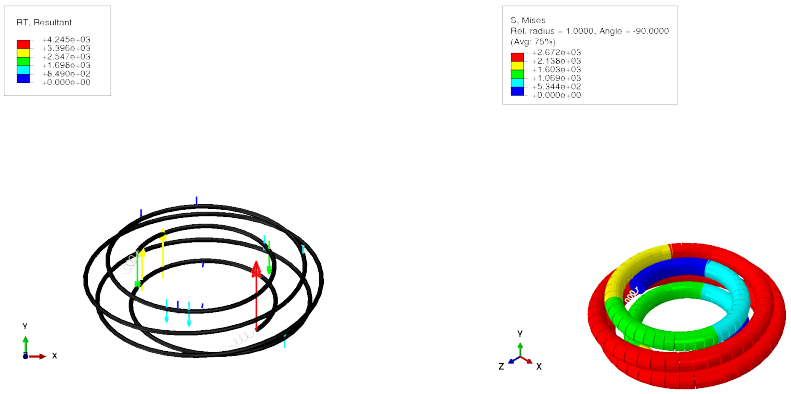
(a) Reaction forces (b) Von-Mises stress

Figure 3.37. Specimen 5: Beam model at 132 mm displacement.



(a) Reaction forces (b) Von-Mises stress

Figure 3.38. Specimen 5: Beam model at 150 mm displacement.



(a) Reaction forces (b) Von-Mises stress

Figure 3.39. Specimen 5: Beam model at 172 mm displacement.

4. INFLUENCE OF MODEL PARAMETERS ON SIDE-FORCES

Now that we have built two finite element models for simulating our problem and have tested the performance of the models against the experimental results, the next step is to analyze the behavior of the side-forces with respect to changes in certain parameters. As we observed from previous sections that the side-forces were different for different specimens, these specimens only differ in their profiles which points us to investigate the effect of variations in profile on the side-forces. Furthermore, we also observed that for a single spring the side-force response also differed when the spring was placed in a slightly different orientation/placement which points us to investigate the effect of rigid body placement variations on the side-forces.

In this chapter we will formulate the two types of variations and carry out several simulations to analyze the effect of these variations on the side-forces. As we will be carrying out large number of simulations we have chosen to use the beam-connector model due to its lower computational cost.

4.1 Effect of Variations in Spring Configuration

As discussed earlier, during experimental testing of springs we noticed that the side-force response of springs changes when the spring in placed in a different orientation (slightly moved or rotated manner) whereas the axial response remains identical. These changes in orientation are rigid body motions (translations or rotations) which we call as placement variation. Our aim here is to model the spring with these variations and study the effects of the variations on the side-force response of springs. We will be identifying those variations which affect the response and compare sensitivity of each variation.

Placement variations can be of only six types corresponding to the six rigid body motions, i.e. translations along X ,Y & Z and rotations along X, Y & Z. As we suspect that the side-force response depends is a consequence of contact of spring and its changing status, with a change in rotational placement of the spring it can form new contact pairs which can affect the side-force response.

While modeling the placement variations, we need to keep in mind the setting of our testing setup. We do not want our variations to be an in-feasible amount of translation/rotation which will result in an impossible testing setup. Further, these variations will be only possible because of the clearances present in our setup for example the spring can only translate up-to a maximum distance of the minimum clearance present between the spring and retainer. Therefore, for every type of variation we need to find the available room of movement or the feasible bounds. Also, we need to generalize this framework so as to be applicable for all specimens possible.

In this discussion we will use the coordinate system and notation as discussed in Section 2.1. The only physical constant that we have for our setup is the diameter of the retainer and the wire diameter, them being $D = 2.5$" and $d = 13.5$mm. For our problem we have modeled four different kinds of variations, lateral translation along X & Z and rotations along X & Z with center of rotation as the center point of the vertical axis. We have left out translation along Z as the barrel spring is always constrained to touch the bottom plate. Also, rotation along Y will not produce new contact configuration and was verified to have no affect at all on the model response. Further, note that the barrel spring center line is formed using a cubic spline which is defined by a sequence of N 3D points $\{\boldsymbol{x}_i\}_{i=1}^{N}$ such that the sequence of points is arranged with increasing angular position values (θ_i). We will be using this information in our framework.

4.1.1 Feasible Bounds

When trying to figure the feasible bounds we need to find the maximum amount of variation such that the barrel spring volume does not intersect with the retainer volume. Also, note that if the spring volume intersects with the plate volume then we can always translate the barrel spring and top plate in a manner such that they are just touching each other. This problem can be formulated as an optimization problem as follows:

$$p^* = \underset{p}{\operatorname{argmax}}\, p \tag{4.1}$$

$$\text{subject to:} \quad \boldsymbol{z}_i(p) = \mathcal{A}(p)\boldsymbol{x}_i \notin \mathcal{C} \quad \forall \quad i = \{1, 2, ..., N\} \tag{4.2}$$

Where p is the parameter that we want to vary (amount of translation of rotation) and $\mathcal{A}(p)$ is an operator which operates on $\boldsymbol{x}_i$ to gives us the transformed points $\boldsymbol{z}_i(p)$. For placement variations the operator $\mathcal{A}(p)$ is a linear operator, in case of translations its just an additive operation and for rotation its the rotation matrix. Also $\mathcal{C}$ is the cylindrical volume (refer Fig 4.1) spanned by a unbounded cylinder centered at retainer center with a radius of $(D+d)/2$ and infinite height. If any transformed point $\boldsymbol{z}_i(p)$ resides in the volume $\mathcal{C}$ then we will have an in-feasible setup as the volumes of the spring and retainer will be intersecting.

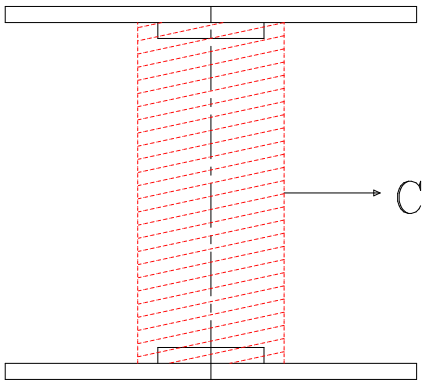


Figure 4.1. Cylindrical volume constraint.

Further, if we closely examine the constraint we have only a single constraint defining the volume $\mathcal{C}$ which is the radius of the cylinder. Therefore we can simplify our original constraint, which makes the new optimization problem as:

$$p^* = \underset{p}{\operatorname{argmax}}\, p \tag{4.3}$$

$$\text{subject to:} \quad r_i(p) = \|\boldsymbol{P}\boldsymbol{z}_i(p)\|_2 \geq \frac{(D+d)}{2} \quad \forall \quad i = \{1, 2, ..., N\} \tag{4.4}$$

or

$$r_{min}(p) = \min_{i=1,2,..,N}(r_i(p)) \geq \frac{(D+d)}{2} \tag{4.5}$$

Where $\boldsymbol{P}$ is the projection tensor defined in Equation (2.4). Now that we have formulated our optimization problem, the next step is to formulate our operator $\mathcal{A}(p)$.

4.1.2 Variation Operator

As discussed earlier our variation operator will be a linear operator for rigid body motions. In this section we will explicitly write out those operators for our four types of variations.

Translation Operator

For translation, the operator will be an additive operator and will shift the corresponding coordinate of the spring. The two operators for the two types of translations are:

1. Translation along X:

$$\boldsymbol{z}_i(\delta_1) = \mathcal{A}(\delta_1)\boldsymbol{x}_i = \boldsymbol{x}_i \pm \delta_1 * \boldsymbol{e}_1 \tag{4.6}$$

2. Translation along Z:

$$\boldsymbol{z}_i(\delta_3) = \mathcal{A}(\delta_3)\boldsymbol{x}_i = \boldsymbol{x}_i \pm \delta_3 * \boldsymbol{e}_3 \tag{4.7}$$

For translation operator, in the optimization problem we will be solving for a single scalar quantity $\delta \geq 0$ as our parameter.

Rotation Operator

For rotation, the operator will be a multiplicative (i.e linear) operator for rigid body rotation. The operator will be nothing but rotation matrices. Let's write them explicitly:

1. Rotation along X:

$$\boldsymbol{z}_i(\phi_1) = \mathcal{A}(\phi_1)\boldsymbol{x}_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\pm\phi_1) & -sin(\pm\phi_1) \\ 0 & sin(\pm\phi_1) & cos(\pm\phi_1) \end{bmatrix} \boldsymbol{x}_i \tag{4.8}$$

2. Rotation along Z:

$$\boldsymbol{z}_i(\phi_3) = \mathcal{A}(\phi_3)\boldsymbol{x}_i = \begin{bmatrix} cos(\pm\phi_3) & -sin(\pm\phi_3) & 0 \\ sin(\pm\phi_3) & cos(\pm\phi_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}_i \tag{4.9}$$

For rotation operator, in the optimization problem we will be solving for a single scalar quantity $\phi \geq 0$ as our parameter.

4.1.3 Solving Procedure

Now that we have completely formulated our optimization problem, we need to identify a solving procedure. In the optimization problem we have a linear objective function and a non-linear constraint, one possible way to solve is to find the roots of the constraint functions and find the maximum parameter from those roots. However, we cannot find a gradient for our constraint as the $min(..)$ function is not differentiable. We can use numerical gradients or use gradient-less methods but they will be more expensive. Also, we are not interested in finding the exact optimum value up-to full precision, we are just interested in finding the closest possible value so that we can see the effect of variation on the side-force. Therefore, we solve the problem using enumeration by iterating over a sequence of parameter values starting from 0

with a user specified step length (Δ) as our choice of update and for each iterate we evaluate the constraint and check if we satisfy the constraint. Algorithm 1 concisely describes this process.

Algorithm 1: Feasible bound.

Result: p^*

$p^* = 0$;

Constraint=1;

Δ=0.1 ; // user parameter

while *Constraint==1* **do**

$\boldsymbol{z}_i(p) = \mathcal{A}(p)\boldsymbol{x}_i$;

$r_i(p) = \|\boldsymbol{P}\boldsymbol{z}_i(p)\|_2$; // $\forall \quad i = \{1, 2, ..., N\}$

$r_{min} = min(r_i(p))$;

Constraint = $(r_{min}(p) \geq \frac{(D+d)}{2})$; // logical 1 or 0

$p^* = p^* + \Delta$

end

We can use the above algorithm to find the lower and upper bounds for translational and rotational variations by just changing the signs in our operators. A negative sign will give us the lower bound and a positive sign will give us the upper bound. Furthermore, it is to be noted that the bounds obtained by such a procedure will be dependent on the geometry of spring and the initial position of points $\boldsymbol{x}_i$. Therefore based on the sequence of rigid body motions that we choose to carry out, we will have different bounds for parameters.

4.1.4 Design of Experiments

After evaluating the variation bounds we can now design a set of experiments (i.e. computer simulations) for placement variation. When we say experiments, it means that we will be carrying out a sequence of simulations for different combination of

variations by jumping between lower and upper bounds for each parameter. These experiments can be chosen randomly or in an ordered fashion, for instance using Full Factorial Design or Box-Behnken Design or Central Composite Design. To study the effect of different placement variations we have carried out two sets of experiments. In the first set of experiments we will be identifying those parameters which affect the side-force the most and in the second set we will be honing onto the critical parameters and observe their effects for a wider range. The details of each set of experiments is as follows:

Experiment 1

In this experiment we are allowing variation in four parameters, i.e. two for translation (δ_1 & δ_3) and two for rotation (ϕ_1 & ϕ_3). We are solving for upper and lower bounds for translational variations however for rotational variations we are fixing the lower bound as zero and only finding the upper bounds. The sequence of rigid body motions chosen for this experiment is first a rotation along X axis then rotation along Z axis then translation along X axis and finally translation along Z axis. The number of combination and the amount of variation to be carried out is chosen using a full factorial design with three factors (-1,0,1) i.e. we will be choosing amongst three possible values, lower & upper bounds and mid point of the bound, for each parameter. This means that we will carry out a total of $3^4 = 81$ number of simulations. The results of the experiments are discussed in Section 4.1.5. Also, note that we have carried out this experiment for spring specimen 5 only as we have experimental data for different placements available for this specimen to compare with.

Experiment 2

After analyzing the results from experiment 1, we observed two of the parameters corresponding to translational variations (δ_1 & δ_3) did not affect the side forces and

therefore were discarded for this experiment. Now we have honed to only two critical parameters (ϕ_1 & ϕ_3) and would observe their effect on a wider range of values. In this experiment we are now solving for both the lower and upper bounds for rotational parameters. The number of combination and the amount of variation to be carried out is again chosen using a full factorial design with three levels (-1,0,1) i.e. we will be choosing amongst three possible values, lower & upper bounds and mid point of the bound, for each parameter. This means that we will carry out a total of $3^2 = 9$ simulations. The results of the experiments are discussed in Section 4.1.5. We have carried out this experiment for all spring specimens.

4.1.5 Results

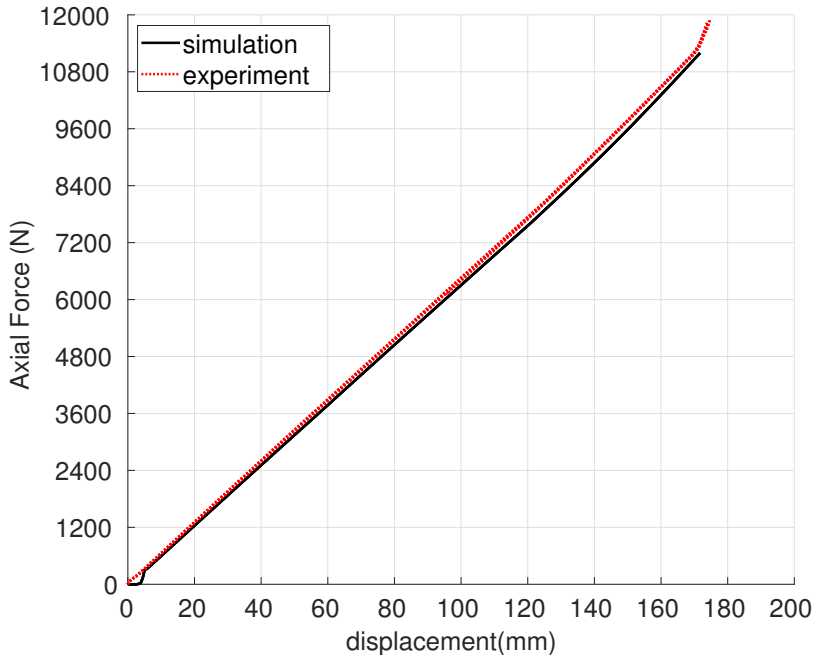
In this section we will be presenting the results for placement variation for the two sets of experiments and will draw inferences from the results. Also,the order of magnitude of bounds for the results presented here are within ±5(mm) for translation and ±2 (degrees) for rotation.

Experiment 1

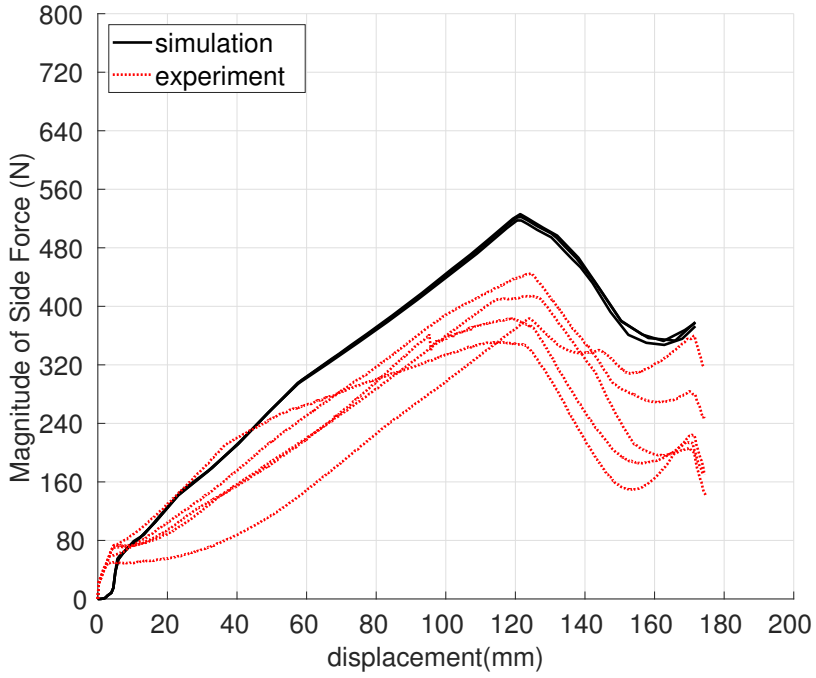
For this experiment we have both translational and rotational variations.

Translation Variation The plots for spread of axial-force and side-force response for translation variation are placed in Fig 4.2, Fig 4.3 & Fig 4.4.

From Fig 4.2, Fig 4.3 & Fig 4.4 for translation variation we can observe that both the axial force and side force are unaffected by the translation variations. This was expected as a pure translation variation will only translate the contact regions and therefore will not affect either of the forces. Thus, for the next experiment we can discard these parameters.

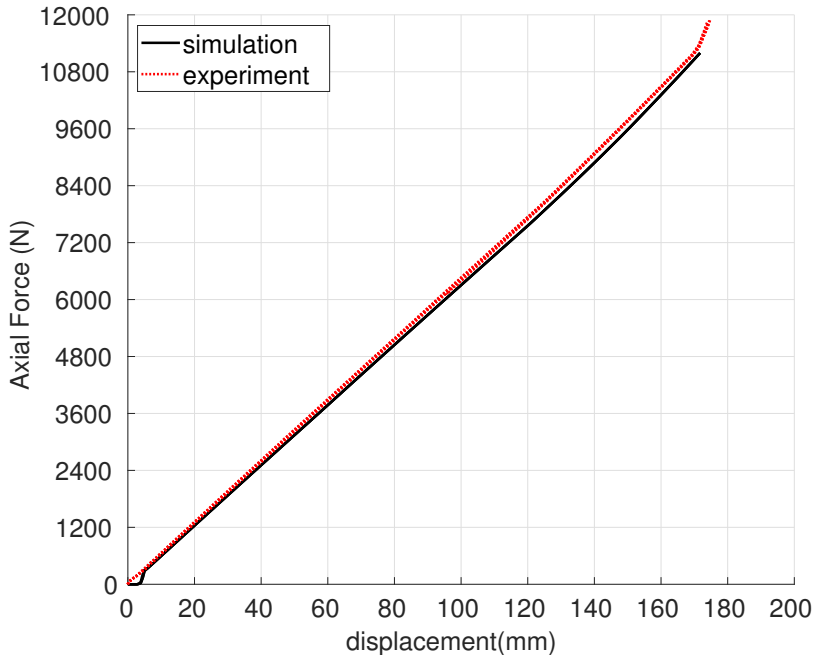


(a) axial force

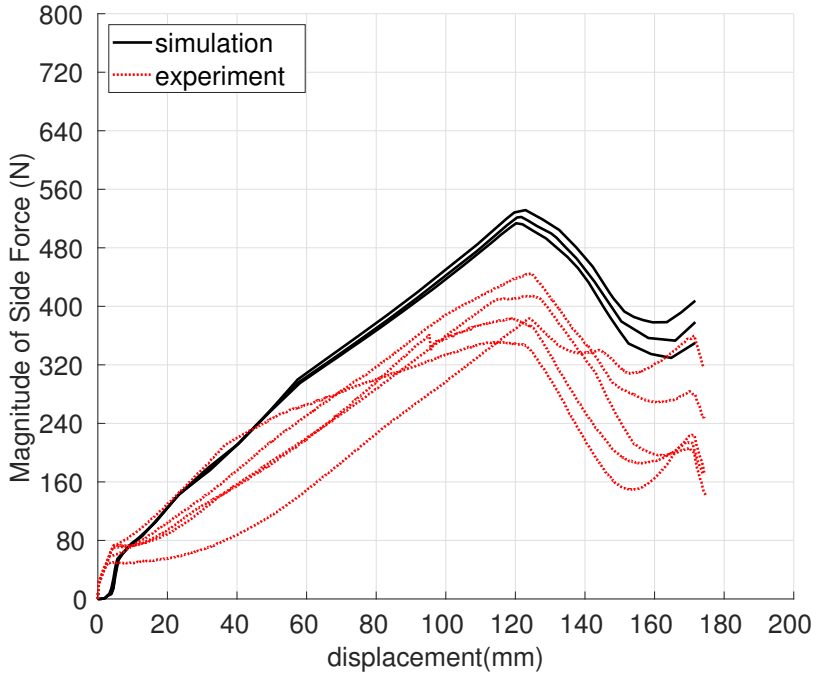


(b) side force

Figure 4.2. spread for only translation in X with zero rotations for specimen 5.

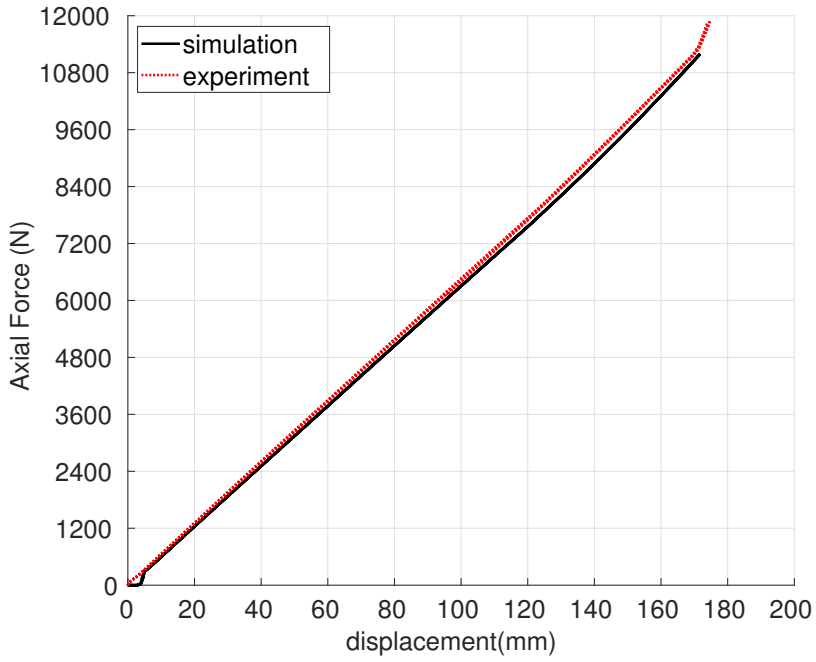


(a) axial force

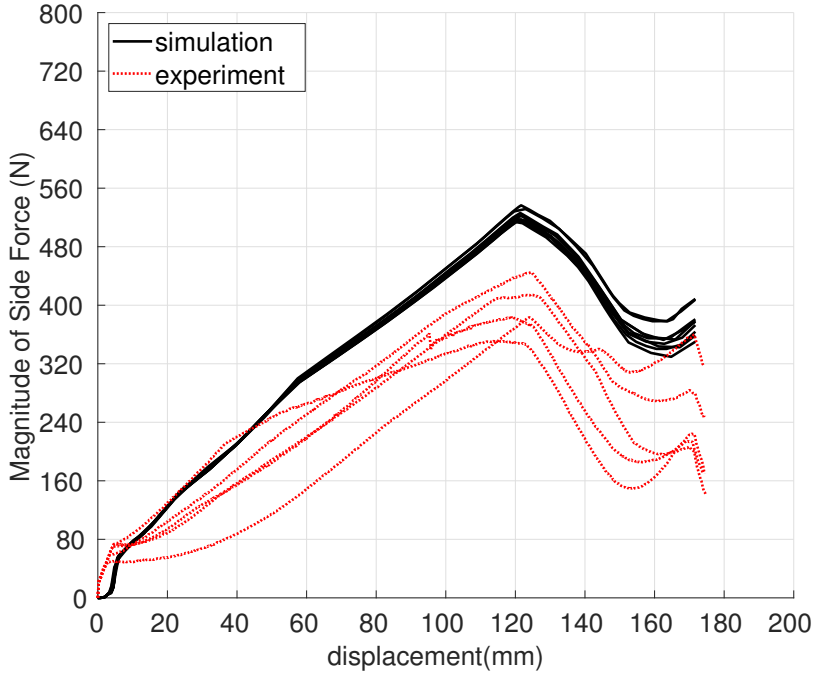


(b) side force

Figure 4.3. spread for only translation in Z with zero rotations for specimen 5.



(a) axial force



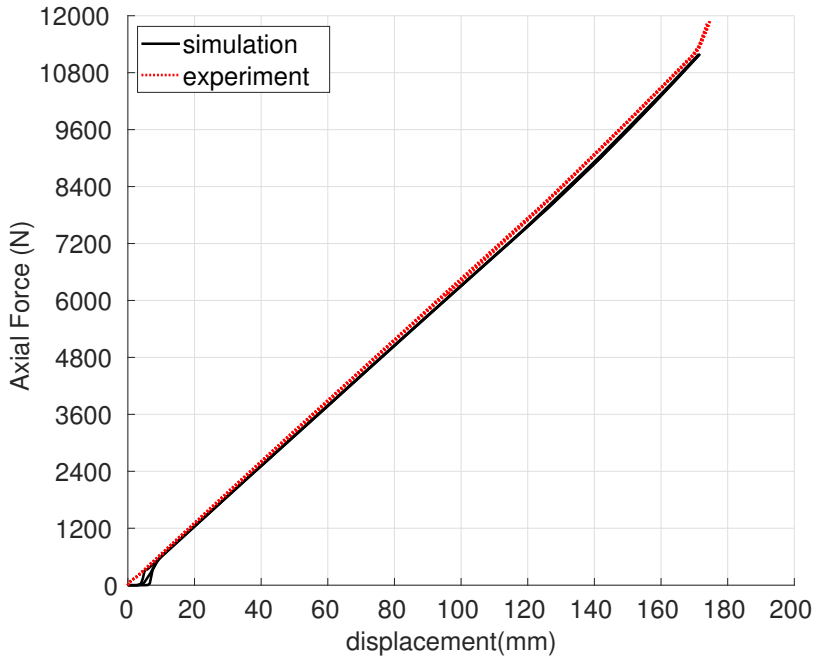
(b) side force

Figure 4.4. spread for both translation in X & Z with zero rotations for specimen 5.

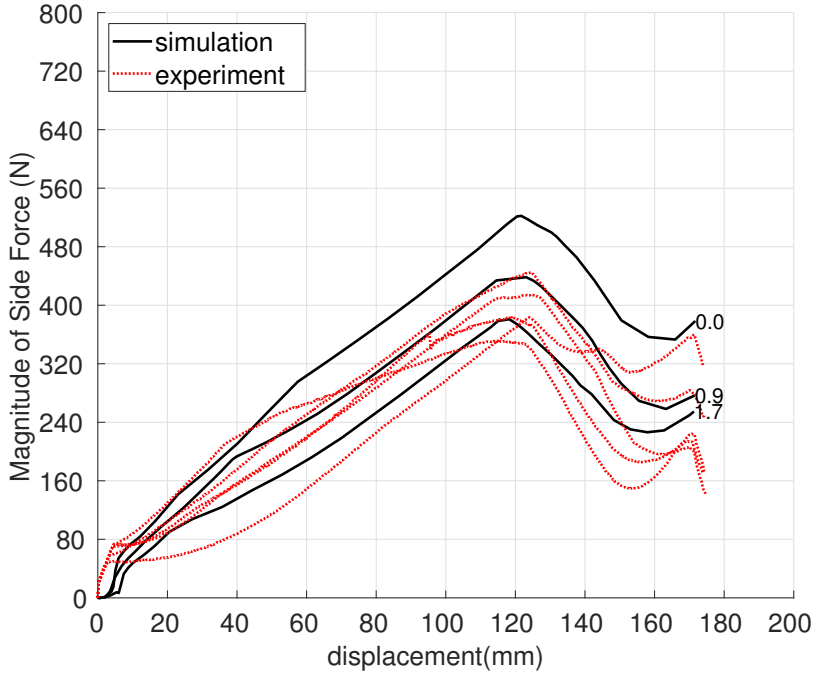
Rotation Variation The plots for spread of axial-force and side-force response for translation variation are placed in Fig 4.5, Fig 4.6 & Fig 4.7.

From Fig 4.5, Fig 4.6 & Fig 4.7 for rotation variation we can observe that the axial force is unaffected by these variations, however, the side-forces do get affected due to variations in rotation. This was expected as a rotation variation will alter the contact regions and therefore will affect the side-forces. Thus, for the next experiment we can focus on these parameters. Also, as our bounds for rotation depends on the geometry of the spring thus we can expect different type of spread for different specimens.

We can also observe that the spread of side-forces from our simulations matches closely with the spread of experimental data. Therefore with these results we can also comment on the possible rotation value of the experimental data.

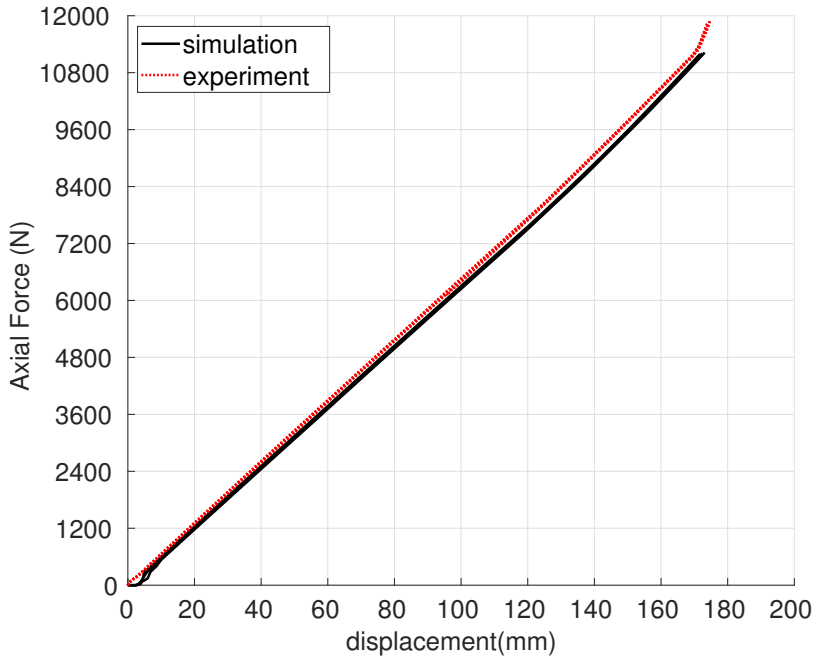


(a) axial force

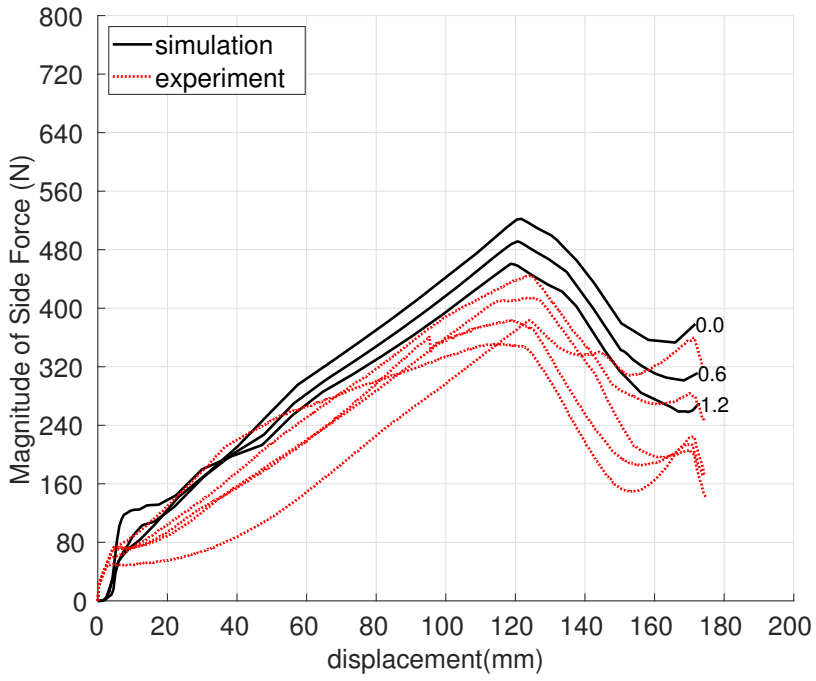


(b) side force

Figure 4.5. spread for only rotation in X with zero translations for specimen 5 with annotation indicating X-angle in centi-radians.

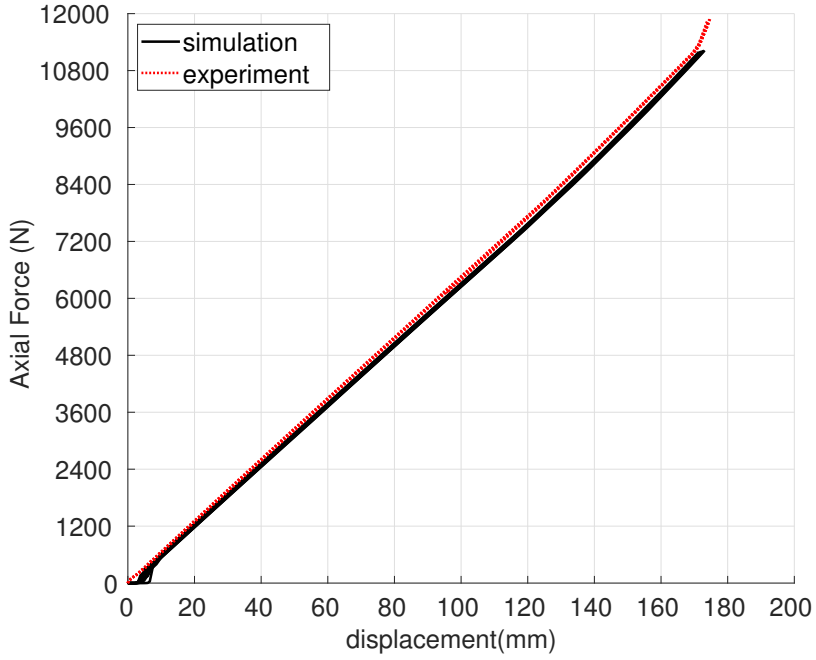


(a) axial force

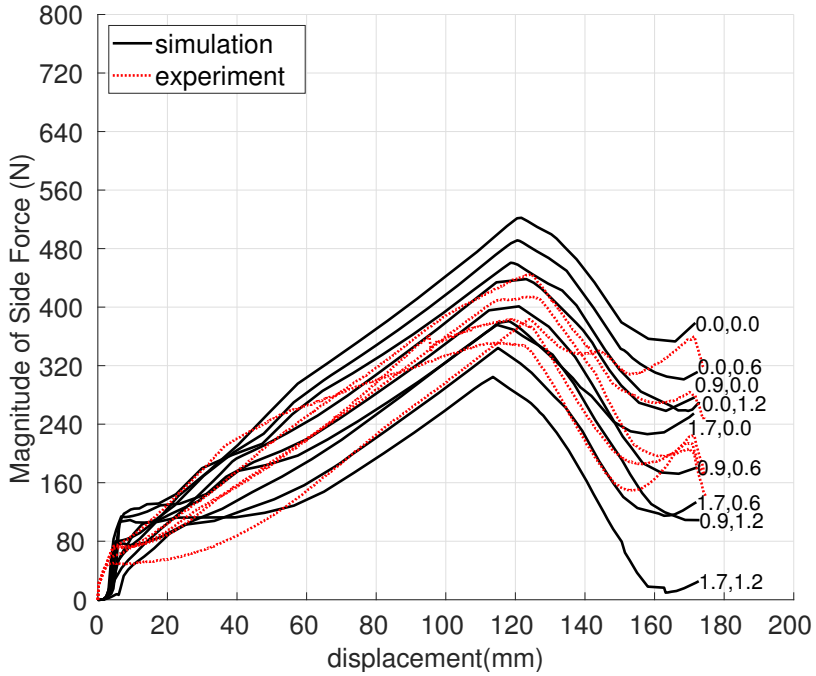


(b) side force

Figure 4.6. spread for only rotation in Z with zero translations for specimen 5 with annotation indicating Z-angle in centi-radians.



(a) axial force



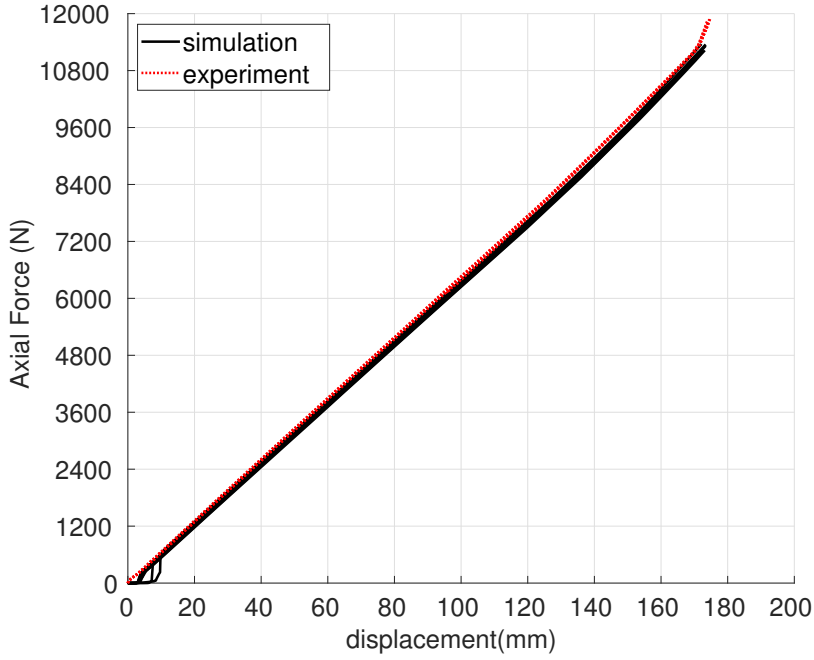
(b) side force

Figure 4.7. spread for both rotation in X & Z with zero translations for specimen 5 with annotation indicating angle pair X,Z respectively in centi-radians.

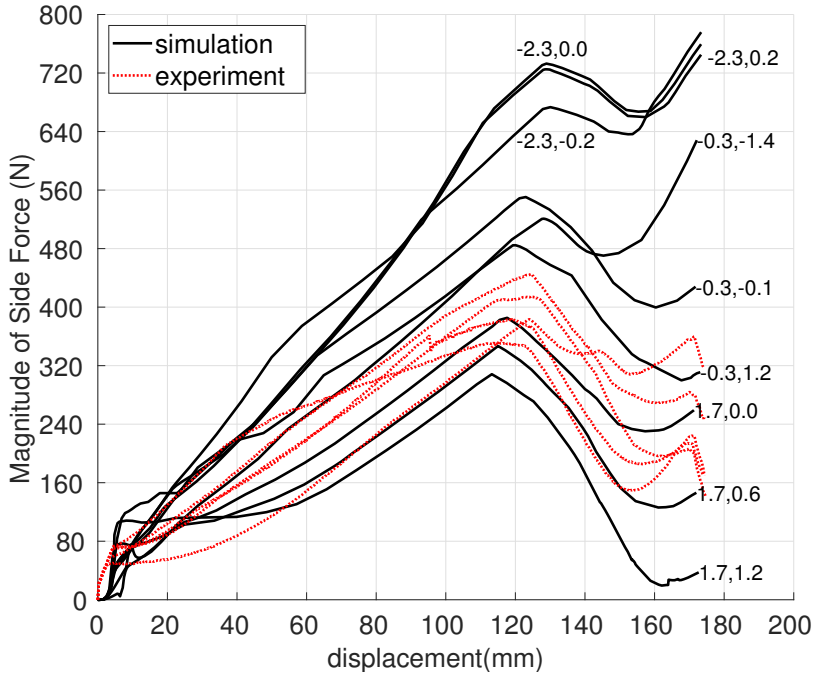
Experiment 2

For this experiment we have only rotational variations. From plots in Fig 4.8, Fig 4.9, Fig 4.10, Fig 4.11 & Fig 4.12 for the five specimens we can observe that the side-forces are quite sensitive to rotational variations. Also, we can observe that the spread of side- forces changes for different specimens, this is because the spread of variation depends on the amount of rotation which is dictated by geometry of the spring.

When we have a closer look at the results for specimen we can identify certain patterns, for specimen 5 we can observe from Fig 4.5,Fig 4.6,Fig 4.7 and Fig 4.8 that as the rotational angle increases for both the directions X and Z, the side forces decrease. Also, a rotation in negative direction increases the side forces. Whereas, for specimen 1 the trend is different as shown in Fig 4.9 which indicates that as the rotation along X increases the side forces increase and as rotation along Z increases the side forces decrease. This inconsistent pattern indicates that side-forces are not purely dependent on the placement variation but rather are also affected by the geometry of the barrel spring. Therefore in the next section we will investigate the effect of profile variation on the side-forces.

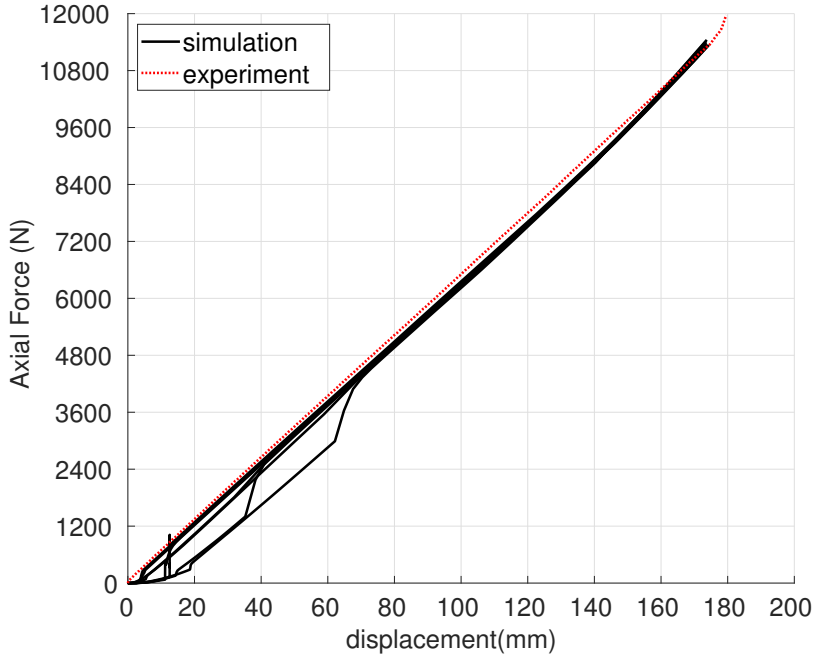


(a) axial force

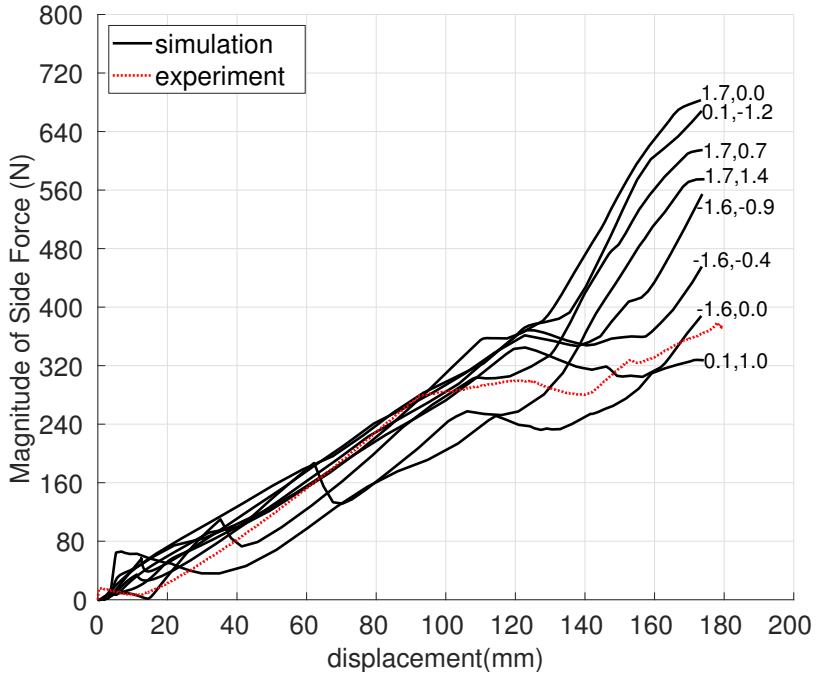


(b) side force

Figure 4.8. spread for both rotation in X & Z with zero translations for specimen 5 with annotation indicating angle pair X,Z respectively in centi-radians.

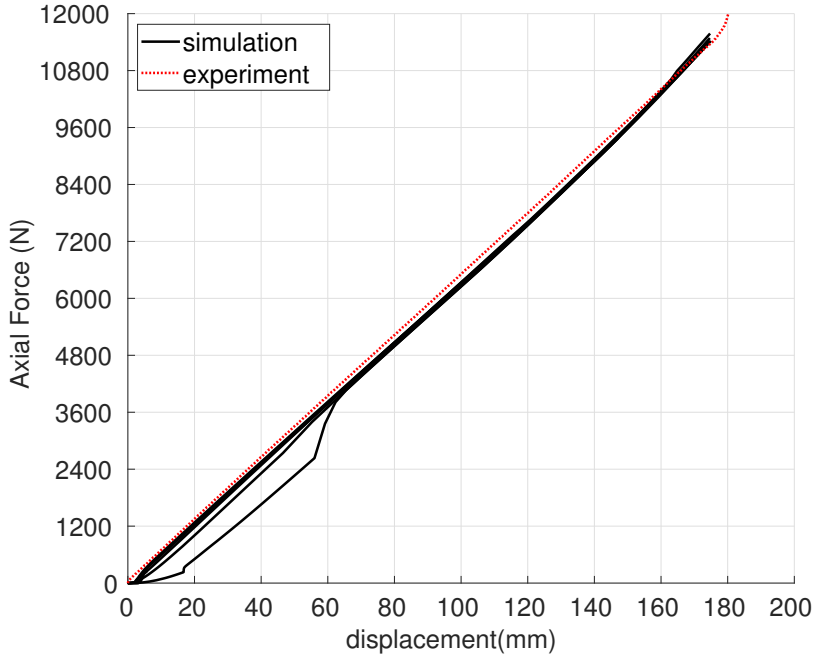


(a) axial force

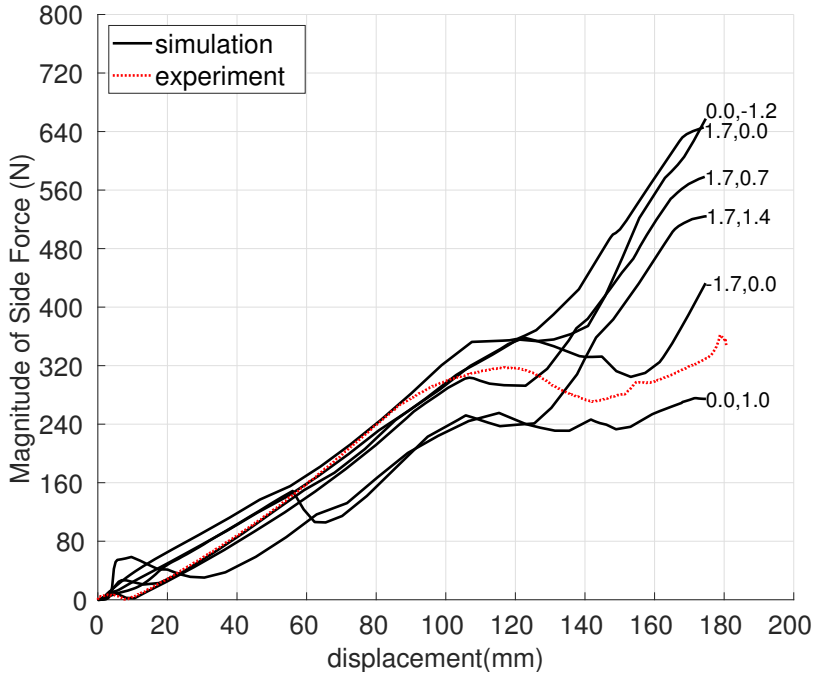


(b) side force

Figure 4.9. spread for both rotation in X & Z with zero translations for specimen 1 with annotation indicating angle pair X, Z respectively in centi-radians.

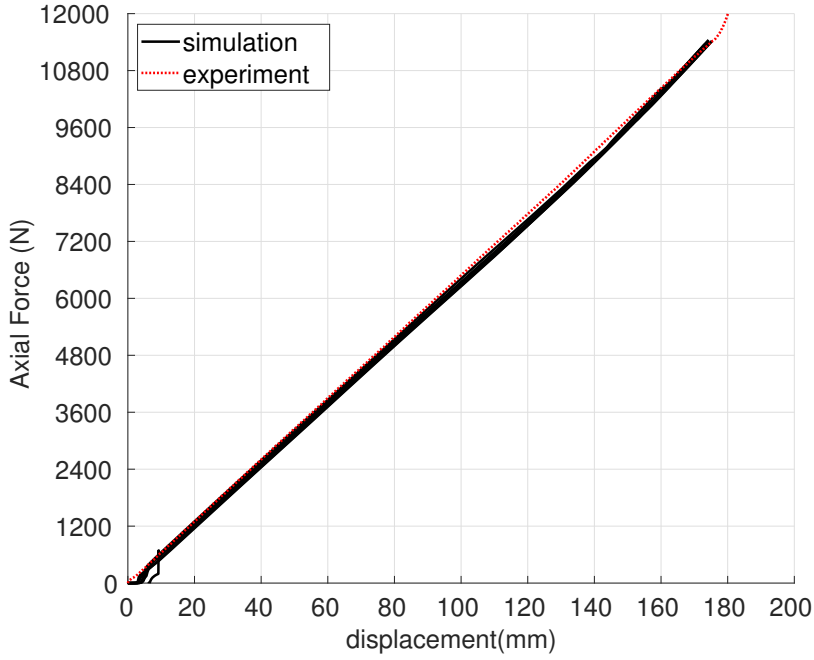


(a) axial force

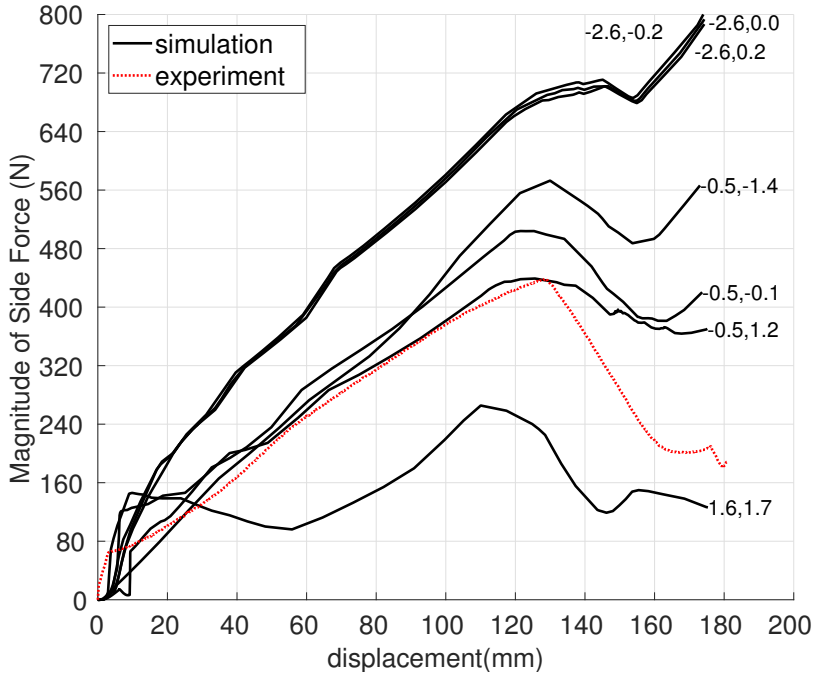


(b) side force

Figure 4.10. spread for both rotation in X & Z with zero translations for specimen 2 with annotation indicating angle pair X,Z respectively in centi-radians.

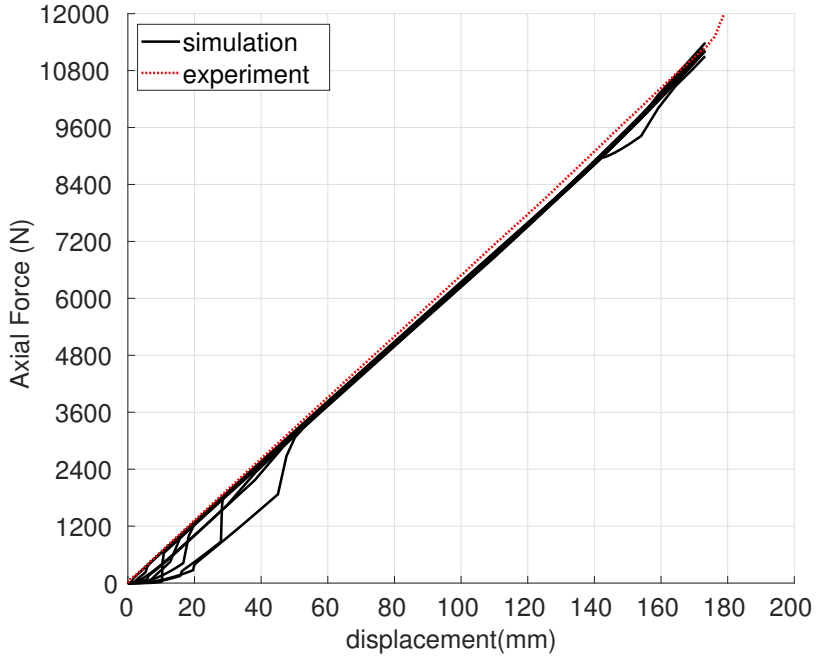


(a) axial force

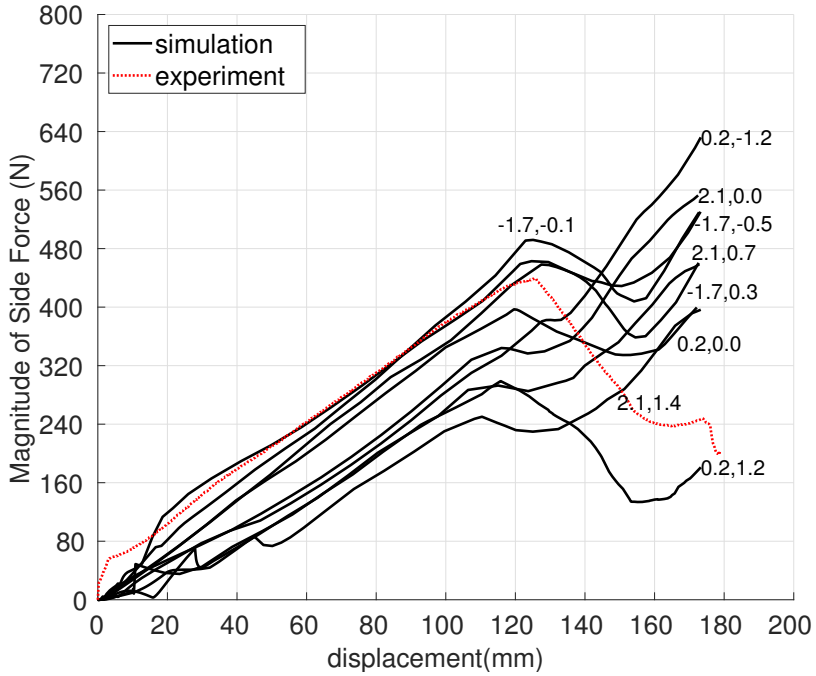


(b) side force

Figure 4.11. spread for both rotation in X & Z with zero translations for specimen 3 with annotation indicating angle pair X,Z respectively in centi-radians.



(a) axial force



(b) side force

Figure 4.12. spread for both rotation in X & Z with zero translations for specimen 4 with annotation indicating angle pair X,Z respectively in centi-radians.

4.2 Effect of Variations in Spring Geometry

As discussed earlier the springs are produced with a manufacturing tolerance which results in subtle changes in profiles which is also evident for our specimens as shown in Fig 2.2. Up until now we have observed that the side-force response of springs can change with respect to placement variation as discussed in Section 4.1.5. Further, these results vary across different springs as the bounds for placement variation were geometry dependent. Also we have noticed that the spring response varies even when all specimens are placed in a similar placement setting as shown in Section 3.2.6. These observations lead us to investigate effects of variation in profiles on the side-force response.

While modeling the profile variations, we need to keep in mind that our variations are within the space of profiles observed for the five specimens. As we don't have the exact manufacturing tolerances for the whole profile we will be choosing the magnitude of variations based on profiles of the five specimens. The idea here is to first identify an 'average spring' with the help of the five specimens and then carry out variations on this average spring with the magnitude of variation calculated from difference of 'average spring' and rest of the five springs.

Also, as our profile is a continuous function there can be infinite number of variations that we can carry out, to make the problem manageable we would need to define the profile with the help of countable number of points and carry out variations for small number of groups/regions of these points. We also need to ensure that the variations across these groups/regions of points have smooth transition at the boundary of the two groups/regions because while manufacturing of the spring it is highly unlikely that we would observe a sudden kink of variation in profile, rather variations across the length of the spring will be smoother.

Then once we have a manageable framework for generating any spring from different types of variations, we need to check that the final profile generated for all variations is a valid profile, i.e. is not a self-intersecting volume.

4.2.1 Average Spring Profile

When we say average spring it means that the profile of such a spring will be computed by taking average of the profiles of five specimens. As discussed earlier, we have about 3000 points for each specimen which define its profile. Also, these profile measurements might have different maximum angle values across all specimens as the starting position of the measurement probe might be different for different specimens due to human error. Therefore to calculate the average profile, we first find out the minimum out of the maximum angle ($min(\theta_{max})$) for the five springs. Then we generate a list of equally spaced 3000 points for the average spring ranging from zero to $min(\theta_{max})$. Then for each point we find the value of diameter and height for the five specimens using linear interpolation and the take their average. Thus with this process we obtain an 'average spring' profile.

4.2.2 Approximating the Profile

After calculating an 'average spring' profile we down-sample the 3000 3D points to 200 3D points. As we saw in Section 3.2.6 we need only a minimum of 200 3D points to have a converging beam connector model. We obtain these 200 points for equally spaced angles using linear interpolation. We can then use these a total of 400 points, ie 200 data points for diameter and 200 data points for height values respectively, to define our approximate profile.

We also investigated the option of down-sampling the points to a mere 13 number of data points for each diameter, height and angle by using cubic spline to define the profile from these points. When we have a closer look at the profiles for the five specimens in Fig 2.2(a) we can observe that only the peaks and valleys in the plots are moving for different profiles. Therefore, we used these points as our control points to define a spring profile. Also, choosing these points as our control points lent us a direct control over the possible shape of profile when carrying out the profile variations. We chose two default control points at the two ends of the spring and then eleven control

points corresponding to the peaks and valleys. These control points would dictate the angular position for corresponding points in Fig 2.2(b). Therefore, we have all three coordinates defined for our thirteen control points. We can easily obtain control points for each specimen using the experimental measurements of profiles. One example of computed control points (in red circles) for specimen 5 is shown in Fig A.11.

When we compared the loading results of the the two methods of approximating the profile, we observed that the force response for the spline approximation as shown in Fig A.14 was nowhere closer to the response of the five specimens. Whereas, the first method with 200 3-D points had a response which was within the space of response of the five specimens. This was because the profile generated using the cubic splines was not closer to the actual profile than the first method and thus the response was significantly different. Therefore, going forward we chose the first method for approximating the profile.

4.2.3 Self-Intersection Checks

After approximating the profile we can generate any number of 3D points along the central axis of the spring. When generating the points, $\{\boldsymbol{x}_i\}_{i=1}^{N}$, we will ensure that the points are in increasing order of equally spaced angles, we will use this information to our advantage when checking for self-intersection. If there is a self-intersection in the spring then that means that the euclidean distance between any two points is less than the wire diameter, d. Also, we need to ensure that these two points are not in fact adjacent/neighboring center-line points and rather are on different turns. We can check this just by finding pair-wise distance of any i^{th} point with every other point and store only those indices which have a distance smaller than d. We can then find the indices of "allowable points" which are neighbors of this i^{th} point such they are within a ball of radius d around the i^{th} point. After finding the indices of allowable points we then find the difference of two sets of indices and check whether it is empty or not. If it is empty then we don't have any self intersections.

To find the indices of the "allowable points" we first start with finding the angle (θ_i) of the i^{th} point and also calculate the projected radius in XZ plane of the point. We can then find the δ angle, which is the change in azimuth angle for the two diametric ends of the ball. Therefore we get two values of angles ($\theta_i \pm \delta$) which correspond to the two ends of the ball. Now, as the points are ordered in such a way that they are at equal angles, we can find the indices of the points which are closest to the calculated angles and then the set indices of allowable points is just an ordered list of numbers starting from the index corresponding to $\theta_i - \delta$ and ending at $\theta_i + \delta$. Algorithm 2 concisely describes this process.

4.2.4 Realistic Bounds

Now that we have devised a way of approximating the profile, the next step is to formulate the bounds of variation for the points. There are two ways to approach it, one way is to simply pose a constant value for the bound throughout the length of the spring another way is to calculate the bounds of variation at each point using the data of profiles for the five specimens this way we stay within the space of manufactured springs. Since we are interested in estimating the behavior of manufactured springs with respect to feasible variations in profile, we will be taking the second route.

To calculate the bounds for each of the 400 points, ie 200 diameter and 200 height points, we take the absolute difference of the average spring profile with the five specimen profiles and then take the maximum difference as our value for the bound. This ways we will have a bound envelope which varies across the length of the spring and can be considered as a bound which is a consequence of manufacturing process. It should be noted that the bounds obtained from this method can have kinks in it and may not be always be smooth. To make these bounds smooth, we have used a convolution filter of width 7 and with weights as per a Gaussian distribution with a variance of $\sigma = 2$, with these parameters we were able to smooth out the local kinks in the bounds without over-smoothing the whole curve. Also, since we are estimating

Algorithm 2: Self Intersection check

Result: notinterfering

Function `Check_4_self_intersection(`*X,Y,Z,max_angle,d*`):`

notinterfering = 1;

Num_pts = length(X);

for *i=1:Num_pts* **do**

i_dist = [X;Y;Z]-[X(i);Y(i);Z(i)];

i_3D_distance = $\sqrt{\text{i\_dist}(1,:).^2 + \text{i\_dist}(2,:).^2 + \text{i\_dist}(3,:).^2}$;

all_idx = find(i_3D_distance < d);

allowed_idx = allowable_idx(i,X(i),Z(i),max_angle,Num_pts,d);

actual_idx = setdiff(all_idx,allowed_idx);

if *actual_idx is not empty* **then**

notinterfering=0;

break;

end

end

return notinterfering;

Function `allowable_idx(`*i,x,z,max_angle,Num_pts,d*`):`

p = (max_angle/Num_pts) ; `// precision`

r = $\sqrt{x^2 + z^2}$; `// projected radius`

theta = (i-1)*max_angle/(Num_pts-1) ; `// angle for the` i^{th} `point`

delta = arcsin(d/r);

theta_min = theta -detla;

theta_max = theta +detla;

i_min = max(floor(theta_min/p),1);

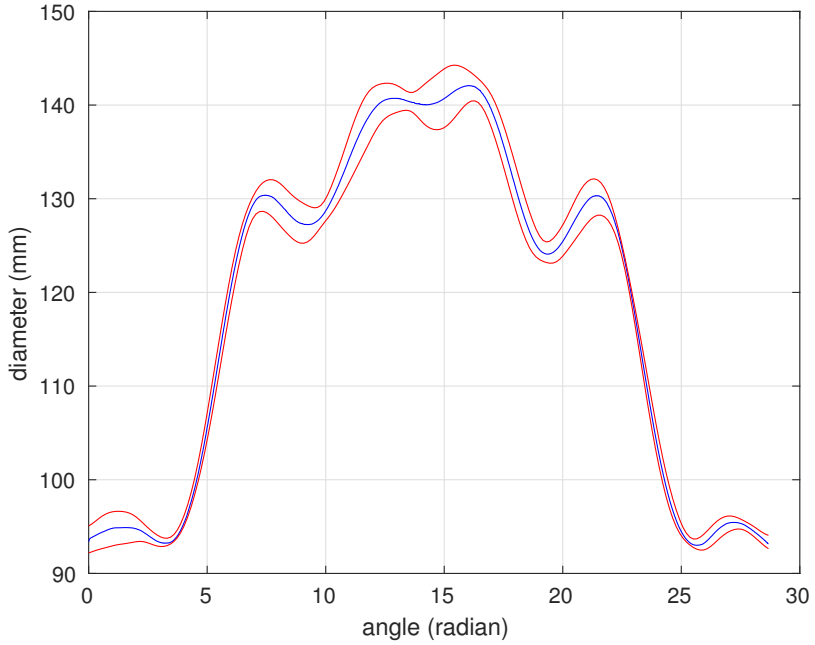
i_max = min(ceil(theta_max/p),Num_pts);

allowed_idx = i_min:1:i_max;

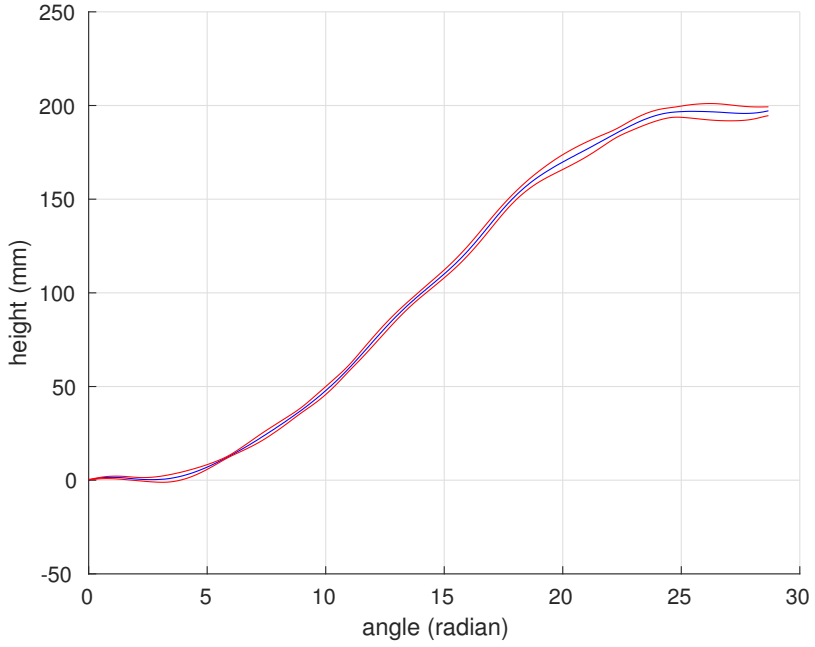
return allowed_idx;

these bounds only from a population size of five springs, we can multiply the bounds by a factor of 2 or 4 to observe the effect of variations and still possibly remain in the space of manufactured springs. The final bounds after smoothing along with the average profile can been seen in Fig 4.13.

When carrying out variations for the 200 3-D points we can have a total of 400 number of random variables corresponding to the diameter and height points. However, when we look at the profiles of the five specimens in Fig 2.2 we can spot certain trends which have manufacturing reasoning associated to them. We will use these trends to limit our number of random variables. We can easily reduce the number of random variables by identifying localized regions of variations when we plot the absolute difference of the average spring profile with the calculated bound. The plots are included in Fig A.8, we can define regions by simply placing a threshold to be acceptable as a region. For our problem we have identified the regions as shown in Fig 4.14.

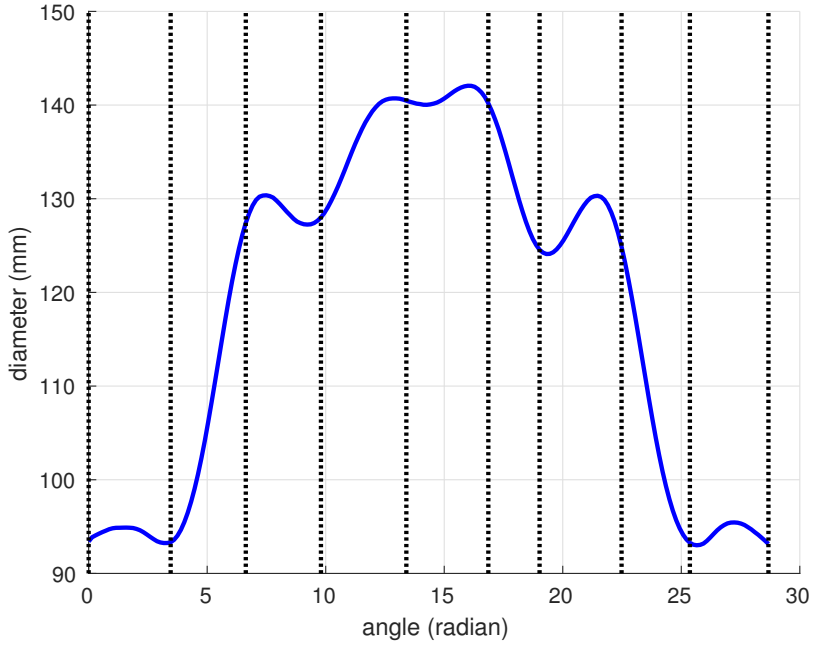


(a) Diameter bounds

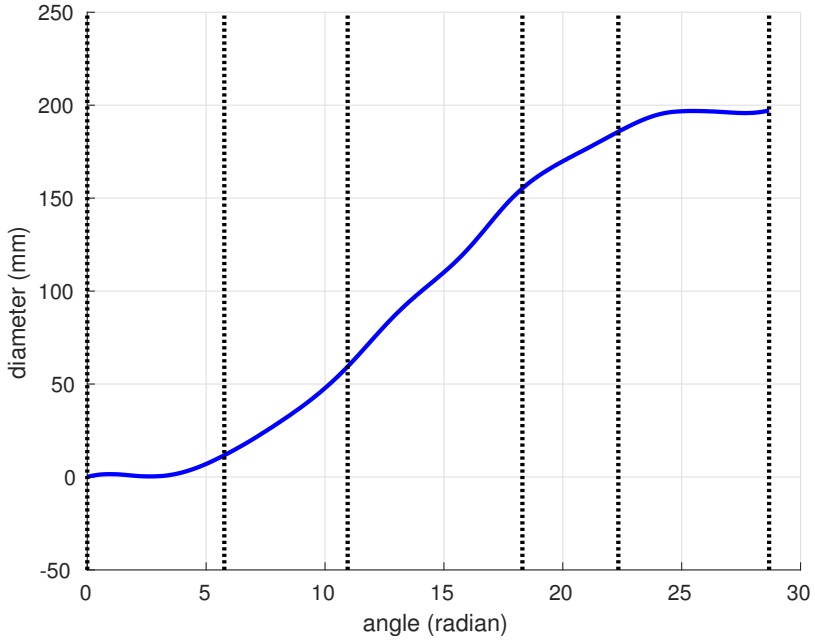


(b) Height bounds

Figure 4.13. Bounds for diameter and Height profiles for a factor of 2.



(a) Diameter regions



(b) Height regions

Figure 4.14. Regions for random variables (separated by black lines).

4.2.5 Smooth Transition of Random Variables

Now that we have defined the bounds and regions of variation for carrying out a design of experiments study, we need to ensure that the values of the random variables across the regions transition in a smooth way. This is because without smoothing each random variable acts essentially as a step response on its region and the transition of the random variable value across the regions can lead to abrupt jumps which is unrealistic for an actual spring being manufactured. When a spring is being manufactured, it is highly unlikely that the variation at one point jumps suddenly, this is because the springs are manufactured using coiler and forming tools which are operated using controllers which have a continuous response. Therefore we need to smooth out the random variable values for all the 400 points which define the spring profile.

When choosing a scheme for smoothing we should also ensure that the random variable value for a region also get contributions from adjacent regions which would mean that the variations are essentially dependent variations. We can introduce these ideas by weighing the random variable value of all regions using weight functions which span across several regions and have a higher value for the region of interest than the adjacent regions. Also, the weighing functions need to be at-least C_0 continuous, however a higher degree of continuity would correspond accurately with the manufacturing process. Thus we have chosen a piece-wise cubic polynomial weighting function spanning 2 or 3 regions.

Piece-wise Cubic Weighing Functions

We have leveraged the idea of mesh-less shape functions to construct our weight functions which helps us achieves C_1 continuity. We have defined three different types of weight functions, two for ends and one for the middle regions. The weight functions are constructed in such a way that they peak over the central region and contribute some small amount to adjacent regions. The weight functions are shown in Fig 4.15.

The end weighing functions, W_1 and W_3, are defined over two regions,$[\xi_0, \xi_2]$ and $[\xi_1, \xi_3]$ respectively, whereas the middle weight function, W_2 , is defined over three regions, $[\xi_0, \xi_3]$. As the weight functions are composed of piece-wise cubic functions, to explicitly find the functions we have defined the function value and its derivative at the nodes for each type of weighting function. These definitions are as follows:

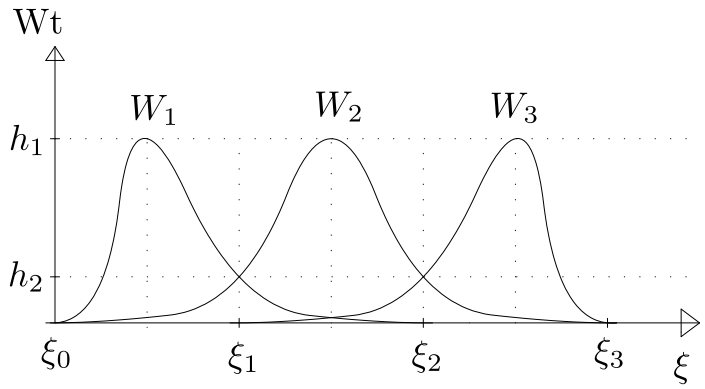


Figure 4.15. Weight functions.

- For W_1:

$$\xi_{mid} = \frac{(\xi_0 + \xi_1)}{2} \qquad \beta = \frac{h_1}{(\xi_{mid} - \xi_2)} \tag{4.10}$$

$$W_1(\xi_0) = 0 \qquad W_1(\xi_{mid}) = h_1 \tag{4.11}$$

$$W_1(\xi_1) = h_2 \qquad W_1(\xi_2) = 0 \tag{4.12}$$

$$\frac{\mathrm{d}W_1}{\mathrm{d}\xi}(\xi_0) = 0 \qquad \frac{\mathrm{d}W_1}{\mathrm{d}\xi}(\xi_{mid}) = 0 \tag{4.13}$$

$$\frac{\mathrm{d}W_1}{\mathrm{d}\xi}(\xi_1) = \beta \qquad \frac{\mathrm{d}W_1}{\mathrm{d}\xi}(\xi_2) = 0 \tag{4.14}$$

- For W_2:

$$\xi_{mid} = \frac{(\xi_1 + \xi_2)}{2} \qquad \beta = \frac{h_1}{(\xi_{mid} - \xi_0)} \qquad \gamma = \frac{h_1}{(\xi_{mid} - \xi_3)} \tag{4.15}$$

$$W_2(\xi_0) = 0 \qquad W_2(\xi_1) = h_2 \qquad W_2(\xi_{mid}) = h_1 \tag{4.16}$$

$$W_2(\xi_2) = h_2 \qquad W_2(\xi_3) = 0 \tag{4.17}$$

$$\frac{\mathrm{d}W_2}{\mathrm{d}\xi}(\xi_0) = 0 \qquad \frac{\mathrm{d}W_2}{\mathrm{d}\xi}(\xi_1) = \beta \qquad \frac{\mathrm{d}W_2}{\mathrm{d}\xi}(\xi_{mid}) = 0 \tag{4.18}$$

$$\frac{\mathrm{d}W_2}{\mathrm{d}\xi}(\xi_2) = \gamma \qquad \frac{\mathrm{d}W_2}{\mathrm{d}\xi}(\xi_3) = 0 \tag{4.19}$$

- For W_3:

$$\xi_{mid} = \frac{(\xi_2 + \xi_3)}{2} \qquad \beta = \frac{h_1}{(\xi_{mid} - \xi_1)} \tag{4.20}$$

$$W_3(\xi_1) = 0 \qquad W_3(\xi_2) = h_2 \tag{4.21}$$

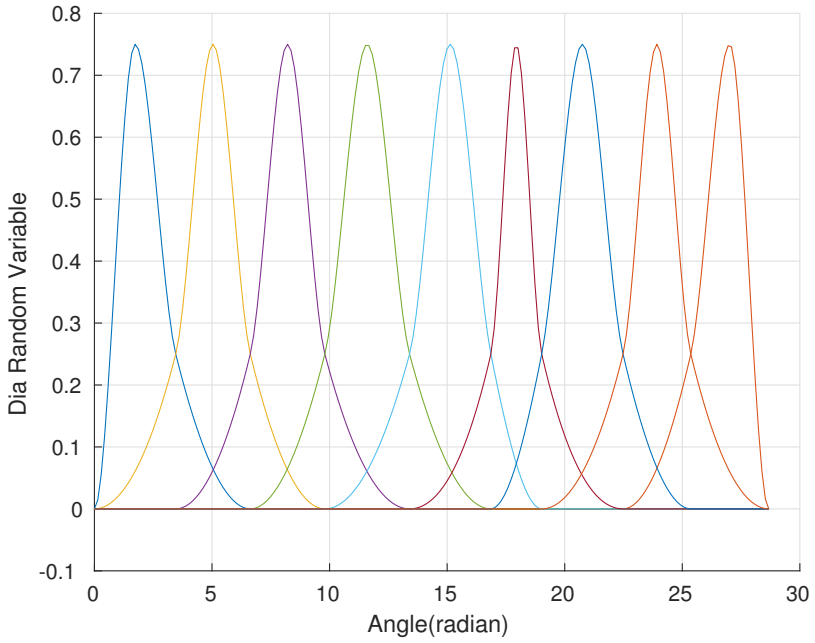
$$W_3(\xi_{mid}) = h_1 \qquad W_3(\xi_3) = 0 \tag{4.22}$$

$$\frac{\mathrm{d}W_3}{\mathrm{d}\xi}(\xi_1) = 0 \qquad \frac{\mathrm{d}W_3}{\mathrm{d}\xi}(\xi_2) = \beta \tag{4.23}$$

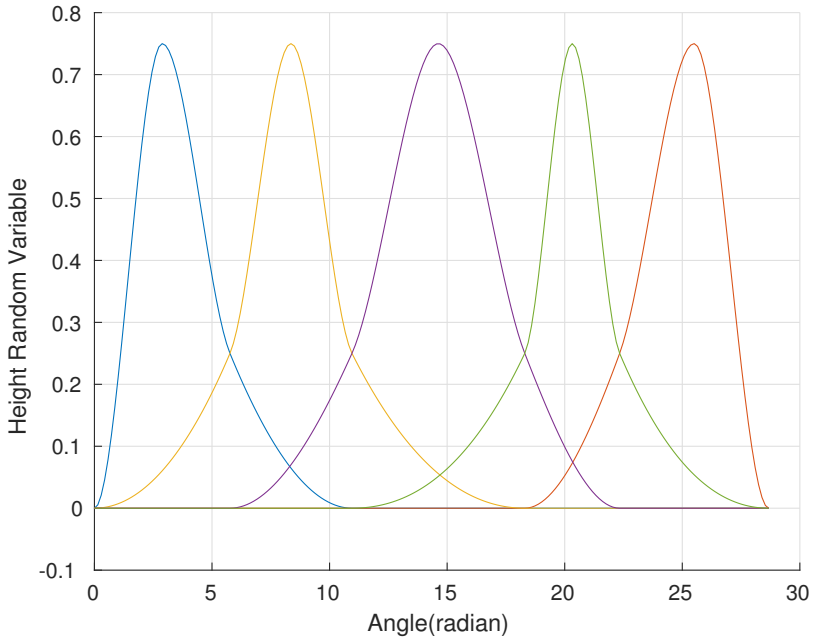
$$\frac{\mathrm{d}W_3}{\mathrm{d}\xi}(\xi_{mid}) = 0 \qquad \frac{\mathrm{d}W_3}{\mathrm{d}\xi}(\xi_3) = 0 \tag{4.24}$$

Using the above information we can construct all weighing functions over any arbitrary domain , $[\xi_0, \xi_1, ..., \xi_n]$, such that W_1 and W_n are defined over the starting and ending regions, $[\xi_0, \xi_2]$ and $[\xi_{n-2}, \xi_n]$ respectively, and all other weight functions, W_m, will be defined over three regions, $[\xi_{m-2}, \xi_{m+1}]$ for all $m = (2, 3, ..., n-1)$.

The plots for computed weight function defined for diameter and height regions are shown in Fig 4.16 with $h_2 = 0.25$ and $h_1 = 1 - h_2 = 0.75$.



(a) Weight functions over Diameter regions



(b) Weight functions over Height regions

Figure 4.16. Weight functions over the computed regions.

4.2.6 Design of Experiments

We will be following a two-step strategy for our experiments similar to how we approached design of experiments for placement variation. In the first experiment we aim to identify the critical profile parameters which affect the side-force the most. After identifying the critical parameters we hone onto them and carry out more number of experiments to draw more informed inferences. The details of each set of experiments are as follows:

Experiment 1

In this experiment we would consider variations over all regions as defined in Section 4.2.4. Moreover, it should be noted that the variation on certain region is very small in comparison to other regions therefore to reduce the number of random variables we can assume no variations on the region which have tight bounds. Using this simplification we can reduce our number of random variables to a total of 7 variables $(q_1, q_2, ..., q_7)$, 5 for diameter variations $(q_1, q_2, ..., q_5)$ and 2 for height variations (q_6, q_7). The choice of random variables for regions is shown in Fig A.9

Further, with 7 random variables, our aim for this experiment is to categorically identify which type of variation affects the side force the most. Therefore we are changing just one parameter to its lower and upper bound and see its effect on the side-force response. Thus we have a total of $7 * 2 + 1 = 15$ simulations. The results of the experiment are discussed in Section 4.2.7.

Experiment 2

After analyzing the results from experiment 1, we observed that side-forces were most sensitive to variations in made in the height vs angle plots. Therefore for the second experiment we focus only on the height variations by drawing more samples to draw more informed inferences. Further, we now increase the number of random

variables for height variations by assigning separate random variable $(q_1, q_2, ..., q_5)$ for each of the five regions as shown in Fig A.10. For this experiment we are carrying out a full factorial design for 5 parameters with two levels (1:on,0:off) such that the levels indicate whether the parameter is contributing or not and then for each combination we draw 5 random samples each. Therefore, we have a total of $31 * 5 + 1 = 156$ simulations. The results of the experiment are discussed in section 4.2.7

Note: For both the experiments carried out above, none of the parameter combination ever produce a non-valid spring i.e. none of them had self-intersections.

4.2.7 Results

In this section we will be presenting the results for profile variation of the two sets of experiments and will draw qualitative inferences from the results . The results presented here are for a profile variation with the bounds of variation as twice the bounds computed from the five specimens. Further, we have also carried experiments for four times the computed bounds and the figures are placed in Section A.

Experiment 1

In this experiment we are mainly concerned in identifying the critical parameter out of diameter and height variations. The results of the simulations carried out using a one factor at a time for for the seven random variables can be seen in Fig 4.18 and Fig 4.20. We can observe from these plots that the side-forces are more responsive towards changes in height variations that diameter variations. It should also be noted that the profiles for diameter variations in Fig 4.17 are visibly more different than the height variations in Fig 4.19 but it does not affect the side-force response. Thus, we have identified that height variations are the critical parameters.

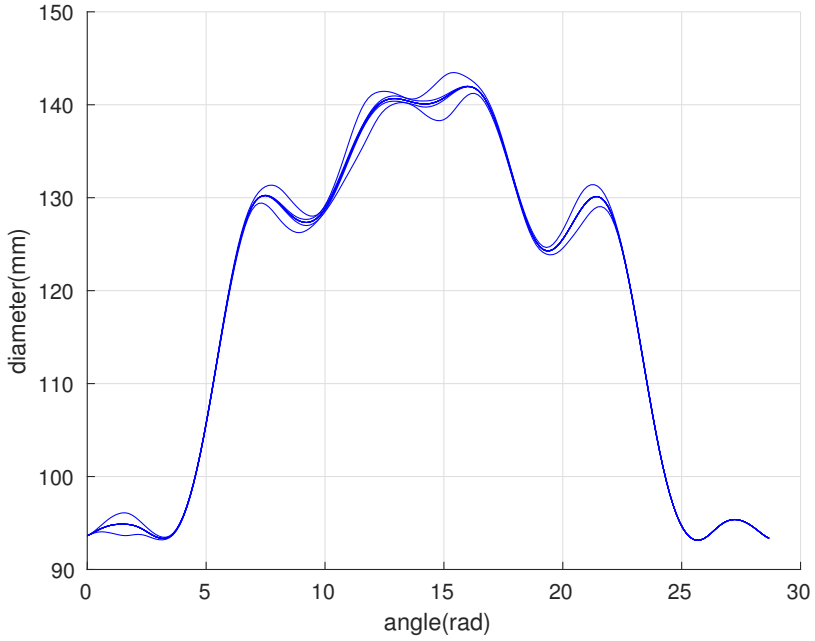
We observed a similar finding when we carried out design of experiments based variation for spline-based approximations of the profiles the spread of forces is shown

in Fig A.14. The results there indicated that height variations were indeed the one which are the critical parameters.

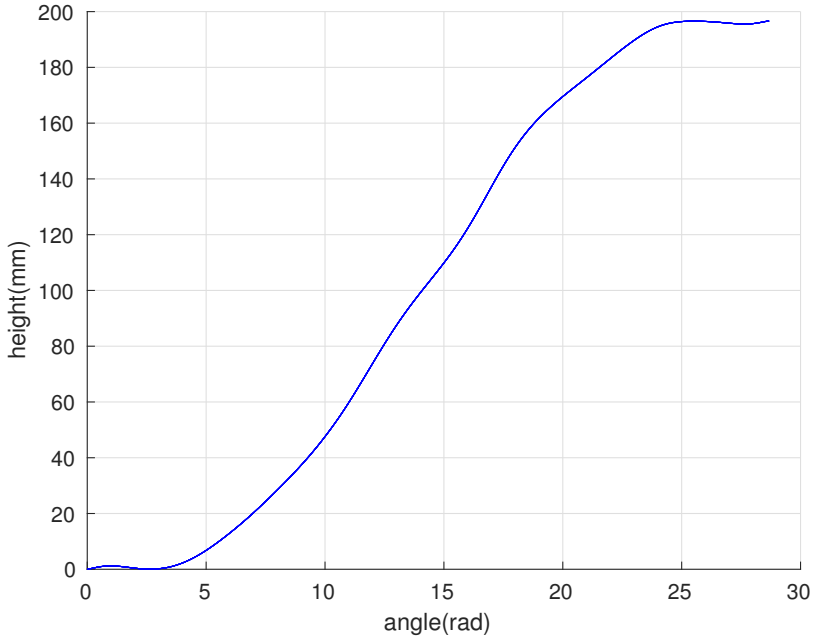
Experiment 2

After identifying that the height variations were the most critical parameter in the previous section, we will now hone onto the height variations only and draw a larger set of samples for this experiment. We have carried out a total of 156 simulations for different profiles as shown in Fig 4.21 for this experiment. The plots for axial and side-forces for this experiment can be seen at Fig 4.22. These results are for a twice the maximum variations, we have also carried out simulations for four times the maximum variations which can be seen at Fig A.2 and Fig A.3.

From the plots in Fig 4.22 and Fig A.3 we can draw statistics for the side-force response. We can calculate the mean and variance of the side-forces and plot them to observe the average behavior of side-forces. These plots are placed in Fig 4.23 and Fig A.4. We can observe that as the bounds of variations are increased we get a wider spread of side-forces, however, we cannot draw any additional inferences on the data with only visual inspection. We would like to investigate that out of the five parameters for the height variations is the most sensitive to the side-force. We will be using regularized linear regression for this process which will be discussed in Section 4.2.8.

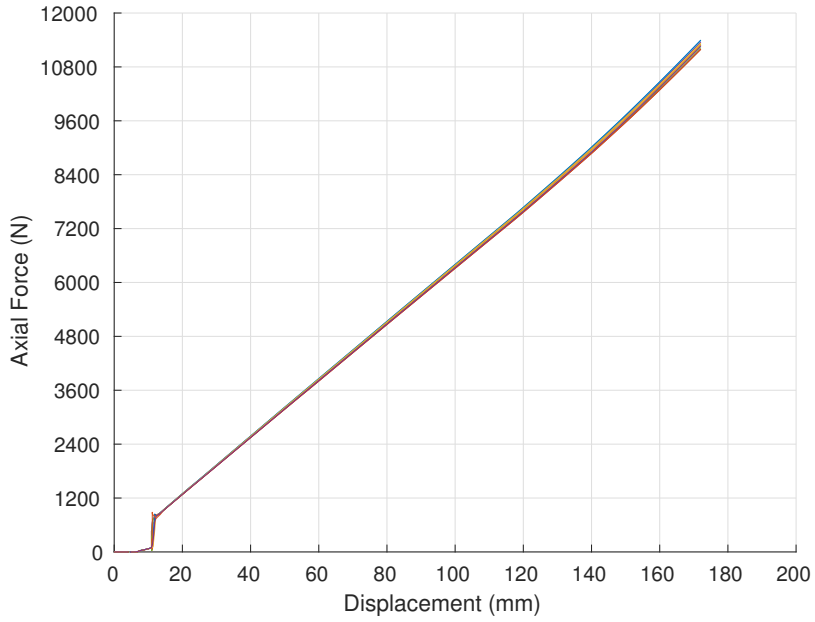


(a) Diameter profiles

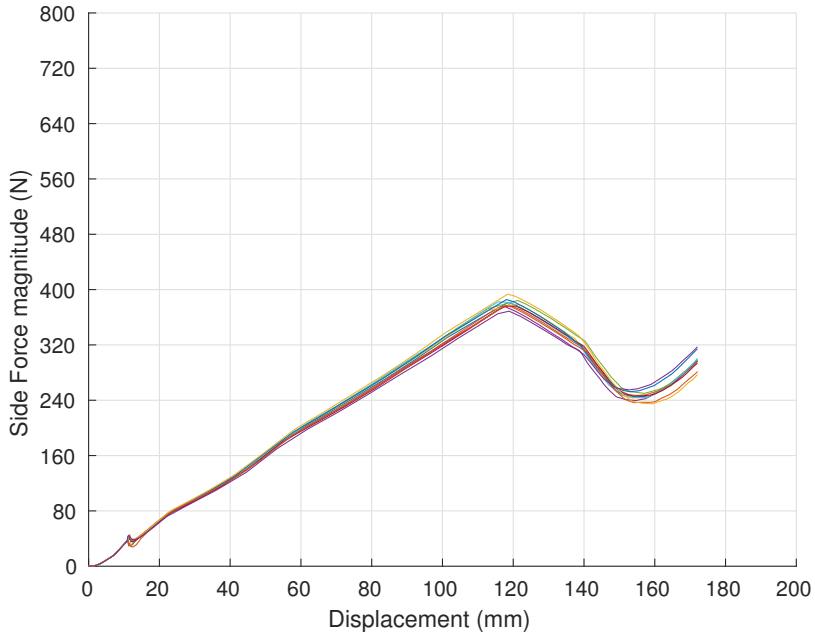


(b) Height profiles

Figure 4.17. Plots of profiles for diameter variations only for experiment 1.

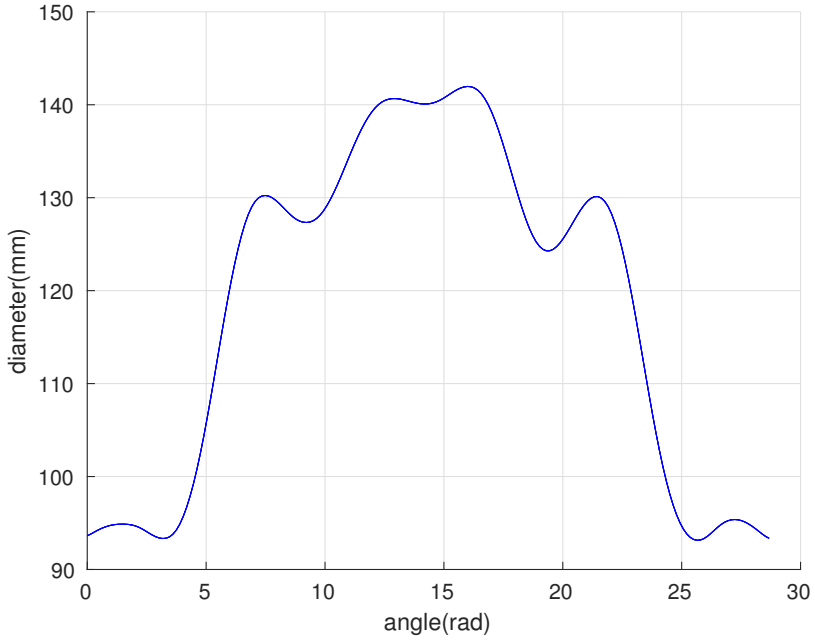


(a) Axial Force

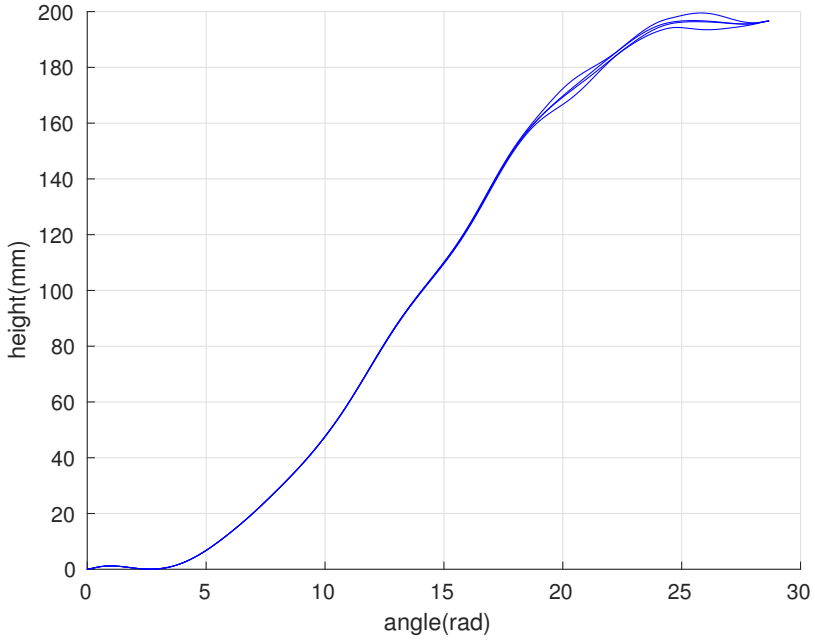


(b) Side Force Magnitude

Figure 4.18. Plots of forces for diameter variations only for experiment 1.

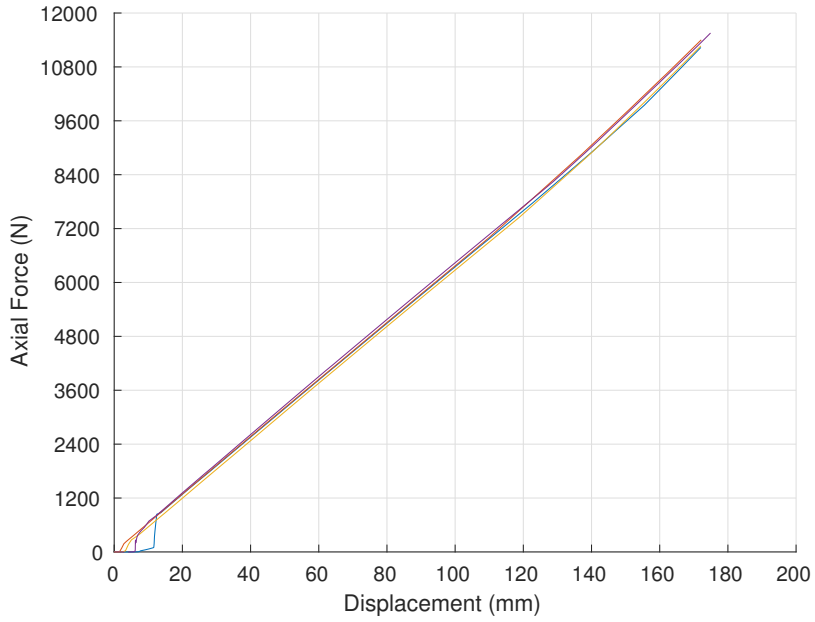


(a) Diameter profiles

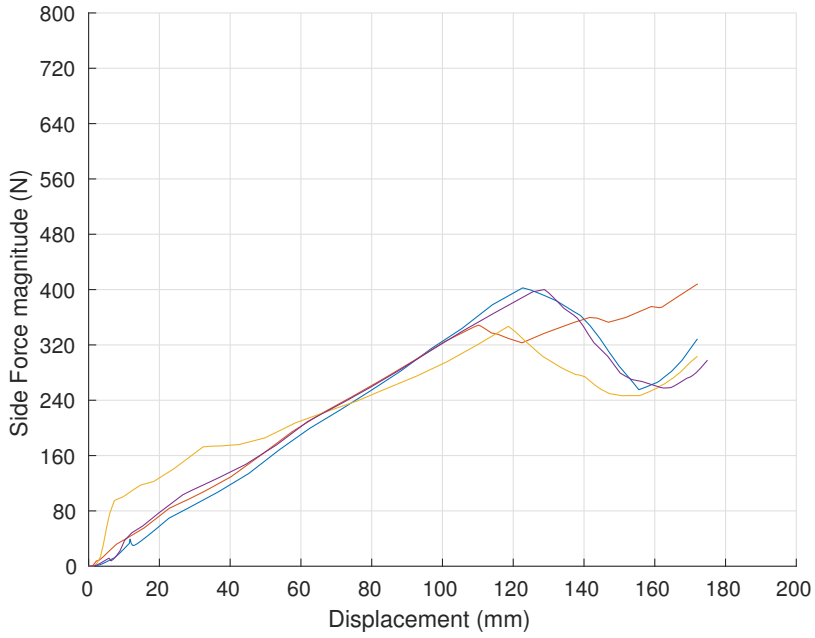


(b) Height profiles

Figure 4.19. Plots of profiles for height variations only for experiment 1.

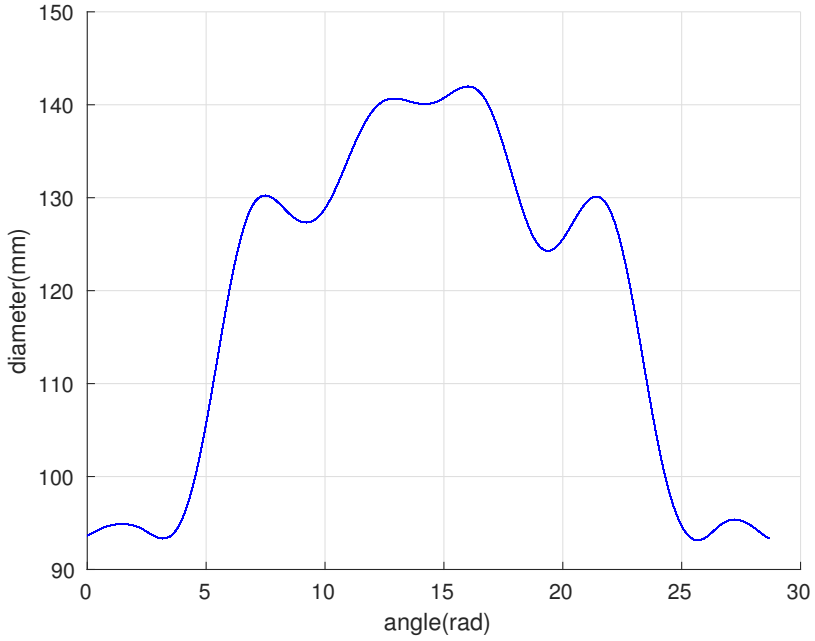


(a) Axial Force

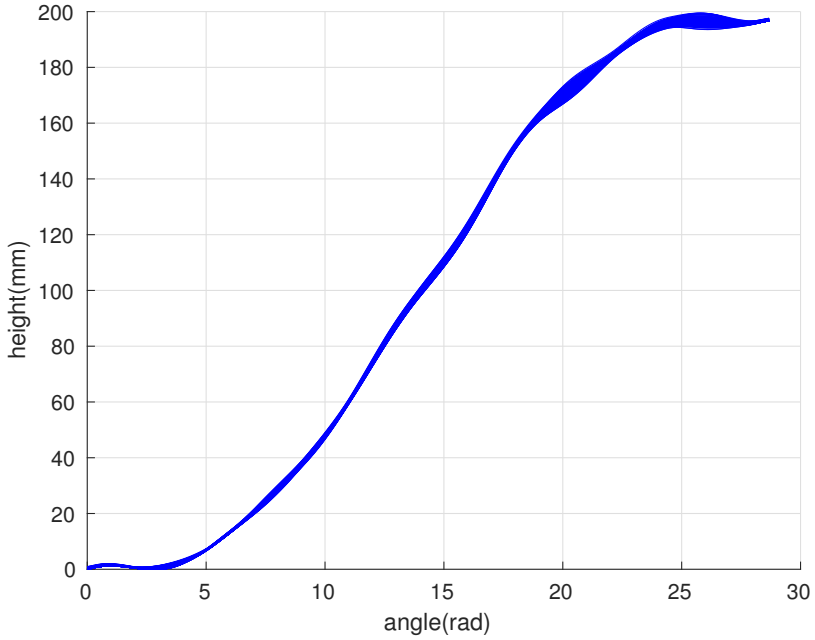


(b) Side Force Magnitude

Figure 4.20. Plots of forces for height variations only for experiment 1.

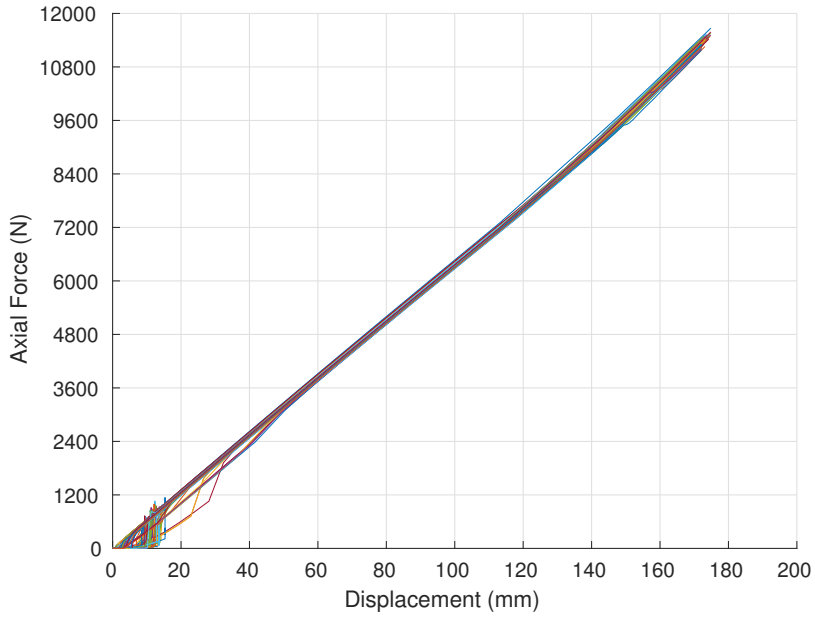


(a) Diameter profiles

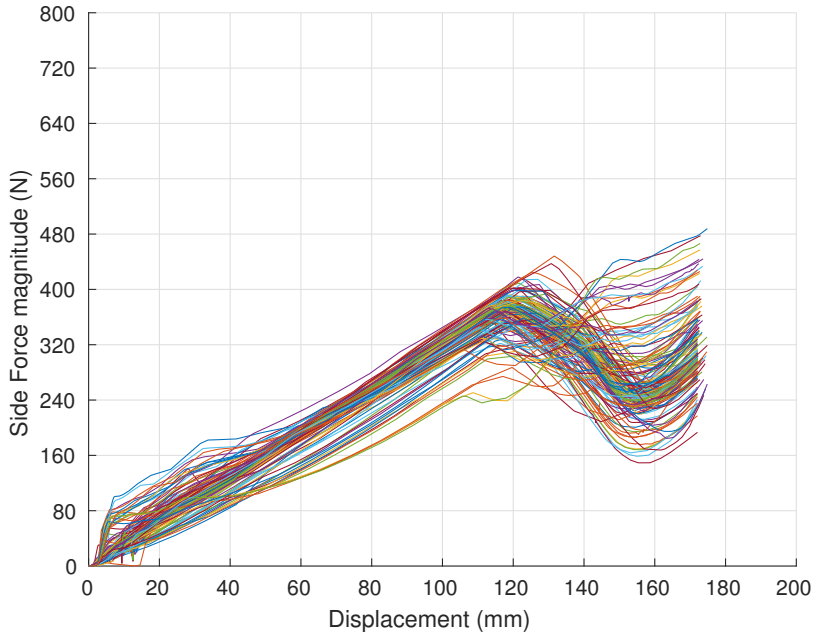


(b) Height profiles

Figure 4.21. Profiles generated for height variations only for experiment 2 for 2x the max variations.



(a) Axial Force



(b) Side Force

Figure 4.22. Plot of forces for height variations only for experiment 2 for 2x the max variations.

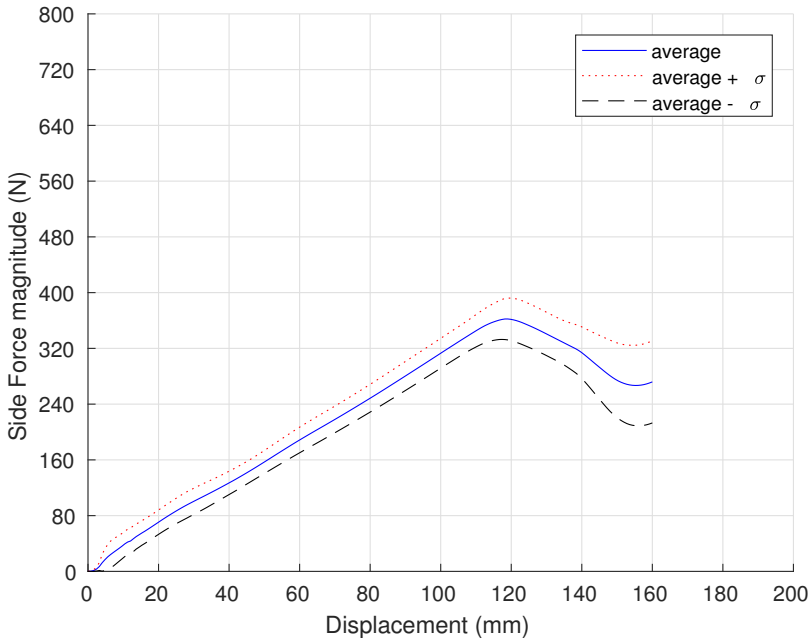


Figure 4.23. Plot of averaged side force for height variations only for experiment 2 for 2x the max variations.

4.2.8 Analyzing Results

In order to study the data as shown in Fig 4.22 and Fig A.3, we will use the idea of L1-regularized linear regression as discussed in Tibshirani et al. [52]. We propose to pose the problem as a linear regression problem. The data generated for experiment 2 of profile variations which has variations for five parameters (height regions), we assume the side-force response, at any given displacement, linearly depends on the parameters. The first step would be to verify this assumption, with the help of linear regression we can calculate the averaged residual for the best-fit hyper-plane and compare this value with the variance in the side-force if it is smaller than the variance then our assumption will hold. After verifying this assumption, using standard linear regression we can find the best fitting hyper-plane and the coefficients of the parameters would represent how sensitive the side-force is to that particular parameter. Furthermore, we want to carry out a sensitivity analysis over the parameters therefore we can assume that the response of the parameters to be sparse by introducing L1-norm regularizing term in our loss function for linear regression we will develop this idea in detail over the next sections. We can carry out this process for different values of displacement throughout the loading history and identify repeating pattern of most critical parameter. Thus with this analysis we will be able to comment on the sensitivity of each parameter.

Linear Regression

As we have carried out a total of $M = 156$ number of simulations, at any given displacement value, α, we can get the value of side-force response for all the simulations. These force values will be our response for the linear regression problem. We can stack them together to make a vector $\boldsymbol{y}$ out of them. Furthermore, assuming that the response $\boldsymbol{y}$ is a linear function of the parameters $\boldsymbol{q} = [q_1, q_2, q_3, q_4, q_5]$ (which are

the random variables in the original problem) , where the numbering convention is same as shown in Fig A.10. The linear relation between $\boldsymbol{y}$ and $\boldsymbol{q}$ will be:

$$y_j = \boldsymbol{w}^T \boldsymbol{Q}_j \quad \forall \quad j = \{1, 2, ..., M\} \tag{4.25}$$

Where $\boldsymbol{Q}_j = [1, q_1, q_2, q_3, q_4, q_5]_j^T$ and $\boldsymbol{w} = [w_0, w_1, w_2, w_3, w_4, w_5]^T$. We can then write this relation for all the M samples as:

$$\boldsymbol{y} = A\boldsymbol{w} \tag{4.26}$$

$$A = \begin{bmatrix} \boldsymbol{Q}_1^T \\ \boldsymbol{Q}_2^T \\ .. \\ .. \\ \boldsymbol{Q}_M^T \end{bmatrix} \tag{4.27}$$

To find the best-fitting hyper-plane using linear regression we define the residual function($J(\boldsymbol{w})$) as:

$$J(\boldsymbol{w}) = \frac{1}{M} \|A\boldsymbol{w} - \boldsymbol{y}\|_2^2 \tag{4.28}$$

We then solve the following optimization problem to find the best-fitting hyper plane.

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, J(\boldsymbol{w}) \tag{4.29}$$

We can solve the above problem by taking the gradient of the objective function. The solution is as follows:

$$\begin{aligned} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) &= \boldsymbol{0} \\ \Rightarrow \nabla_{\boldsymbol{w}^*} \frac{1}{M} (A\boldsymbol{w} - \boldsymbol{y})^T (A\boldsymbol{w} - \boldsymbol{y}) &= \boldsymbol{0} \\ \nabla_{\boldsymbol{w}} (\boldsymbol{w}^T A^T A\boldsymbol{w} - \boldsymbol{w}^T A^T \boldsymbol{y} - \boldsymbol{y}^T A\boldsymbol{w} + \boldsymbol{y}^T \boldsymbol{y}) &= \boldsymbol{0} \\ 2(A^T A)\boldsymbol{w} - 2(A^T)\boldsymbol{y} &= \boldsymbol{0} \\ \boldsymbol{w}^* = (A^T A)^{-1} (A^T)\boldsymbol{y} \end{aligned} \tag{4.30}$$

Therefore, using the above expression for $\boldsymbol{w}^*$ we can find the best-fitting hyper plane. The optimum value of the residual will be:

$$J(\boldsymbol{w}^*) = \frac{1}{M}\|A\boldsymbol{w}^* - \boldsymbol{y}\|_2^2 \tag{4.31}$$

This residual is essentially the average of the sum of normal distance of each sample from the hyper plane. We can assume this as the average prediction error per sample for the linear regression. If this error is smaller than the variance of $\boldsymbol{y}$ then our linear model will hold. We verify this assumption for our data and the comparison is shown in Fig 4.24 and Fig A.5 for twice and four times the max variations respectively. It is evident from the plot that the average error is always below the variance of the samples and therefore our assumption of a linear model will hold.

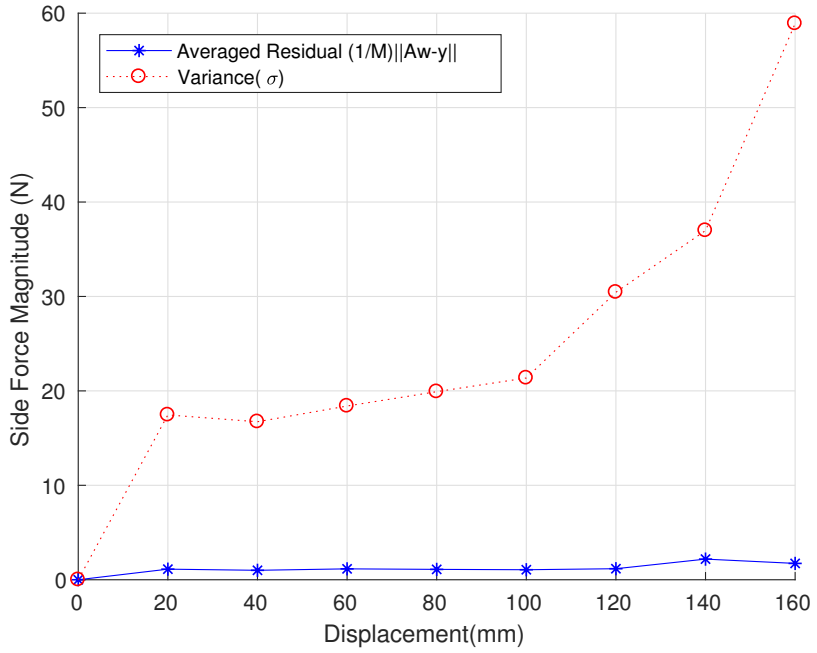


Figure 4.24. average error vs variance for 2x max variations.

We can now carry out a sensitivity analysis using L1-regularization of linear regression problem. We will discuss this in the next section

L1-Regularized Linear Regression

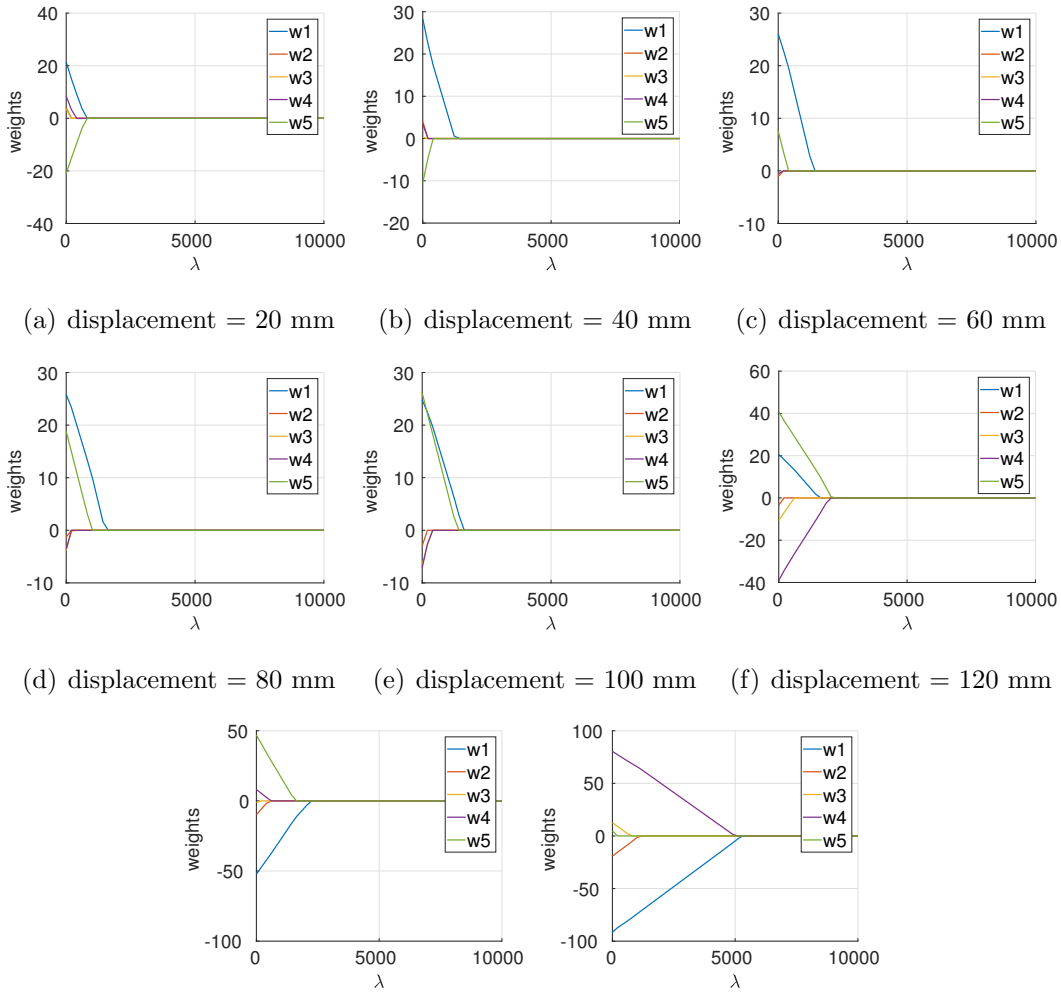
As discussed earlier, to introduce sparsity of factors in the response we need to add a L1-norm regularization term to our residual function. The L1 norm is added as geometrically it promotes sparsity in solution. This method is also called as **LASSO** (Least Absolute Shrinkage and Selection Operator). The new optimization problem with the regularization term then becomes:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{M} \|A\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_1 \tag{4.32}$$

where λ is a user specified scalar. The above objective function is still convex but is non-differentiable. The L1-regularized linear regression problem can be solved using ISTA algorithm [53], FISTA algorithm [54], ADMM algorithm [55] or other solvers. We have used CVX, a package for specifying and solving convex programs [56], [57], to solve this problem.

We can solve the above problem for several values of λ ranging from very high to zero. When we use a very high value for λ we essentially would be just minimizing the regularization term only and thus would get a solution as $\boldsymbol{w}^*_{\lambda=\infty} = \mathbf{0}$ and when $\lambda = 0$ then the L1-regularized problem becomes a standard linear regression problem. We can think of λ as a 'control knob' and as we move from a very high value to zero we would see that the weights (w^*_k) corresponding to the factors (q_k) would start becoming non-zero. Further, due to sparsity introduced by the L1 norm we would see that some factors would start earlier than others. The factor which starts the first would be the one which is most sensitive to the side-force. Thus we would be able to distinguish between the sensitivity of all the factors.

The results for the L1-regularized linear regression for twice and four times the maximum variations are placed at Fig 4.25 and Fig A.6 respectively.



(a) displacement = 20 mm (b) displacement = 40 mm (c) displacement = 60 mm

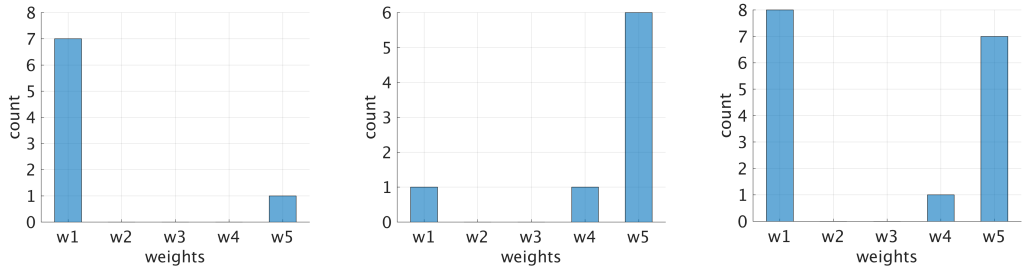
(d) displacement = 80 mm (e) displacement = 100 mm (f) displacement = 120 mm

(g) displacement = 140 mm (h) displacement = 160 mm

Figure 4.25. Regression results for 2x max variations.

The figures Fig 4.25 and Fig A.6 show the plots for the computed weights vs λ using the above process at different values of displacements. We can count the top and second factor which becomes non-zero to identify the two most critical parameters. The histograms in Fig 4.26 and Fig A.7 show the counts of the five parameters for the top, second and top 2 factors over all the computed displacements. We can see that w_1 is always the top factor, w_5 is the second factor and both w_1 & w_5 are always the top two factors which means that the side-forces are sensitive to the random

variables q_1 and q_5. These random variables correspond to the two ends of the spring. Therefore, side-forces are most sensitive to the height variation in the two ends of the spring and more strongly to the bottom end.



(a) Top factor (b) Second factor (c) Top two factors

Figure 4.26. Histogram for Top, second and top two factors for 2x max variations.

Further, from Fig 4.25 and Fig A.6 we can observe that during the initial phase of displacement primarily w_1 and w_5 are the top factors however towards the later phase of displacement, w_4 also becomes important. This indicates that at larger displacement values even the height variations corresponding to the q_4 random variable region become important. We observed a similar behavior when using spline-based approximation for profiles.

Therefore, with the help of the sensitivity analysis we can now conclude that for barrel springs the side forces are very sensitive to variations in height of the coils of the spring specially for the first and last full turn. This is because the side-forces are generated as a consequence of frictional forces between the plate and the barrel springs, and these forces are concentrated at specific regions as brought out by discussion in Section 3.1.6 and Section 3.2.6. The location of the regions of concentration are primarily dictated by the height profile of the barrel spring as any point which is closer to the plate will come into contact earlier and therefore will dictate the location of the region of concentration. Further, as the height of any point on the barrel spring with respect the loading plates can be altered due to rigid body rotations of

the spring, therefore they cause changes in side-forces. Moreover, the non-linearity of side-forces in barrel spring which is evident in the later part of the loading is a consequence of changing status of the regions of concentration such a behavior will not be evident in cylindrical spring as evident from previous studies carried out for side-forces in the existing literature. This is because for cylindrical springs only the first and last coil will be in contact and none of the other coils will ever come into contact and thus the regions of concentration will not change and correspondingly the side-forces will show a linear behavior.

5. SUMMARY AND CONCLUSIONS

In this study the cause of side-forces along with its behavior with respect to changes in geometry and placement of barrel springs was investigated. In order to study the behavior of side-forces, two different finite element models, continuum model and beam & connector model , were built as discussed in Chapter 3. These models were calibrated using experimental data for geometry of the spring and load-testing data as discussed in Section 2.2 and Section 2.3 respectively. Results from the continuum model as discussed in Section 3.1.6 revealed that the side-forces manifest as frictional forces acting over localized regions where the ends of the spring make contact with the loading plates. These contact regions evolve as the spring is progressively loaded leading to the highly nonlinear and unpredictable behavior. The beam & connector model which was built as a computationally economical alternative to the high-fidelity continuum model also exhibited similar behavior as discussed in Section 3.2.6. To gain more insight into the behavior of side forces, their sensitivity to variations in geometric profile and placement of the spring were studied using the beam & connector model because of its lower computational cost.

Placement variations were generated by imparting small rigid body motions onto the spring within the clearances of the housing. To estimate the amount of permissible variation, an optimization problem was formulated in Section 4.1.1 and its solving procedure was discussed in Section 4.1.3. A DOE-based framework was developed to investigate the effect of all six types of rigid body motions on the side-forces. Results of placement variations revealed that only the rotational rigid body motions affected side-forces appreciably. However, the trend of how side-forces were affected as a function of the rigid body rotation was different for different specimens leading to the conclusion that placement parameters are not independent of the parameters describing the geometric profile of the spring.

To ascertain the effect of geometric variations of spring profile on the side-forces, a reference spring profile was first generated by computing the mean of the diameter and height profiles of the five specimens. Then the DOE-based framework was utilized again to carry out variations in the diameter and height about the profile of this reference spring. The spring geometry was partitioned into several different regions along the length of the spring coil where the five specimens were observed to have maximum variations from the reference spring. Diameter and height variations were introduced about the reference spring geometry by specifying a different magnitude of geometric variation for each of the profile regions. Piece-wise cubic spline weighting function were used to smoothen the transition of profile variations across two adjacent regions with different magnitudes of variation as discussed in Section 4.2.4. From the first set of experiments on profile variation, it was found that variations in the diameter profile had very little effect on the side-forces. However, even small variations in the height profile of the spring affected side-force response significantly.

Further, to identify the regions of the spring where variations in the height profile affected side-forces the most, sensitivity analysis was carried out using an L1-regularized linear regression as discussed in Section 4.2.8. The regularized linear regression problem was solved for several displacements along the loading history of the spring with changing regularization parameter as discussed in Section 4.2.8. Results from this analysis indicated that side-forces are most sensitive to variations in height profile of the first and last turns at the two ends of the barrel spring.

The parametric study confirmed that side-forces are highly sensitive to rigid body rotations in the placement of the spring and to variations in the height profile of the spring. It is worth noting that the turns at the ends of the spring make first contact with the loading plates and even small variations in height profile of the spring can change the way this contact evolves upon further loading. Similarly, rigid body rotations also affect the height of different points along the length of the coils of the spring and consequently have a strong bearing on the contact mechanism which is responsible for side-forces.

Therefore, when a barrel spring is designed, the side-forces can be reduced by altering the height profile of the spring or by placing the spring in a rotated orientation but finding the optimal height profile and placement of the spring for minimum side-forces will require additional work which can be carried out in a future study. Further, to reduce the variations in side forces of different specimens, tighter design tolerances on the geometry of the barrel springs may be chosen for the height profile of the springs especially at the two ends of the spring coils while relatively loose tolerances may be allowed on the diameter profile. Manufacturing processes of springs may also be adapted to ensure these tight tolerances on the height profile of the spring.

5.1 Future Directions

Results of this study may be further refined by expanding the beam-connector model to account for self-contact within the coils of the spring. Modeling of self-contact can be achieved with the help of non-linear springs which provide large resistance when the deformed distance between adjacent coils of the spring is less than the wire diameter (d).

The model may also be refined to address the assumptions underlying the current model. Throughout the modeling process, the cross-section of spring is assumed to be circular. However, due to the forming process, the cross-section is bound to become slightly elliptical or irregular. The effect of shape of the cross-section on the side-forces can be investigated. As exhibited by previous studies by Chaudhury et al. [9] and Gzal et al. [41], the shape of the cross-section affects the axial force response which will also change the side-forces.

Also, when calibrating the models with the experiments it was observed that the Young's modulus was the most critical parameter which needed calibration. However, the models built here assumed a constant Young's modulus which may be a weak assumption as the material properties across cross-section of the spring will change due to different temperatures of the core versus the crux of the spring during the heat

treatment process. Therefore, the continuum model may be modified to model varying Young's modulus across the cross-section and then the experimental measurements can be used to calibrate such a model. As this model will have changing stiffness across the cross-section of the spring, the deformation shape of the spring can be different from that of a perfectly homogeneous model. The change in deformation shape can alter the contact regions and thus change the side forces.

Furthermore, this study has been primarily focused on identifying critical parameters that affect the side-forces. However, after identifying and controlling these parameters to have minimum variation in side-forces, one can identify ways to reduce the side-forces by writing out an optimization problem to minimize the side-forces by changing the geometry & placement of the spring taking into account the permissible variations in these parameters.

REFERENCES

[1] Spring Manufacturers Institute. *Handbook of Spring Design*. Spring Manufacturers Institute, 2002.

[2] Air brakes-what you need to know. Accessed: 2019-07-08.

[3] Han-Chung Wang and Will J Worley. Load-deflection behavior of conical spiral compression springs. *Journal of Engineering for Industry*, 84(3):329–337, 1962.

[4] MH Wu and WY Hsu. Modelling the static and dynamic behavior of a conical spring by considering the coil close and damping effects. *Journal of Sound and Vibration*, 214(1):17–28, 1998.

[5] Emmanuel Rodriguez, Manuel Paredes, and Marc Sartor. Analytical behavior law for a constant pitch conical compression spring. *Journal of Mechanical Design*, 128(6):1352–1356, 2006.

[6] Vebil Yıldırım. Exact determination of the global tip deflection of both close-coiled and open-coiled cylindrical helical compression springs having arbitrary doubly-symmetric cross-sections. *International Journal of Mechanical Sciences*, 115:280–298, 2016.

[7] Sanjukta Chakraborty, Koushik Roy, and Samit Ray-Chaudhuri. Design of re-centering spring for flat sliding base isolation system: Theory and a numerical study. *Engineering Structures*, 126:66–77, 2016.

[8] Umit N Aribas, Merve Ermis, Nihal Eratli, and Mehmet H Omurtag. The static and dynamic analyses of warping included composite exact conical helix by mixed fem. *Composites Part B: Engineering*, 160:285–297, 2019.

[9] Arkadeep Narayan Chaudhury and Debasis Datta. Analysis of prismatic springs of non-circular coil shape and non-prismatic springs of circular coil shape by analytical and finite element methods. *Journal of Computational Design and Engineering*, 4(3):178–191, 2017.

[10] Thomas Wünsche, Karl-Heinz Muhr, Karl Biecker, and Leo Schnaubelt. Side load springs as a solution to minimize adverse side loads acting on the mcpherson strut. Technical report, SAE Technical Paper, 1994.

[11] Satoshi Suzuki, Syuji Kamiya, Toshiyuki Imaizumi, and Yukitomo Sanada. Approaches to minimizing side force of helical coil springs for riding comfort. Technical report, SAE Technical Paper, 1996.

[12] JP Hastey, Jacques Baudelet, Eric Gerard, Colin Jones, and Christelle Viel. Optimization on mac pherson suspensions with a spring. Technical report, SAE Technical Paper, 1997.

[13] Takashi Gotoh and Toshiyuki Imaizumi. Optimization of force action line with new spring design on the macpherson strut suspension for riding comfort. Technical report, SAE Technical Paper, 2000.

[14] Toshio Hamano, Takahiro Nakamura, Hideto Enomoto, Naoshi Sato, Shinichi Nishizawa, and Maiko Ikeda. Development of l-shape coil spring to reduce a friction on the mcpherson strut suspension system. Technical report, SAE Technical Paper, 2001.

[15] Shinichi Nishizawa, Tadashi Sakai, Maiko Ikeda, and Winda Ruiz. Spring force line based damper friction control for coil-over-shock applications. Technical report, SAE Technical Paper, 2006.

[16] Jiang Liu, DJ Zhuang, Fan Yu, and LM Lou. Optimized design for a macpherson strut suspension with side load springs. *International Journal of Automotive Technology*, 9(1):29–35, 2008.

[17] YI Ryu, DO Kang, SJ Heo, HJ Yim, and JI Jeon. Development of analytical process to reduce side load in strut-type suspension. *Journal of mechanical science and technology*, 24(1):351–356, 2010.

[18] Jooho Choi, Dawn An, and Junho Won. Bayesian approach for structural reliability analysis and optimization using the kriging dimension reduction method. *Journal of Mechanical Design*, 132(5):051003, 2010.

[19] Shinichi Nishizawa, Winda Ruiz, Tadashi Sakai, and Maiko Ikeda. Parametric study of the spring force line effect on vehicle self steer for macpherson strut suspension system. Technical report, SAE Technical Paper, 2006.

[20] Masayoshi Shimoseki, Toshio Kuwabara, Toshio Hamano, and Toshiyuki Imaizumi. *FEM for Springs*. Springer Science & Business Media, 2003.

[21] William Henry Wittrick. On elastic wave propagation in helical springs. *International Journal of Mechanical Sciences*, 8(1):25–47, 1966.

[22] Leif Eric Becker and WL Cleghorn. On the buckling of helical compression springs. *International journal of mechanical sciences*, 34(4):275–282, 1992.

[23] LE Becker, GG Chassie, and WL Cleghorn. On the natural frequencies of helical compression springs. *International Journal of Mechanical Sciences*, 44(4):825–841, 2002.

[24] V Yildirim. Expressions for predicting fundamental natural frequencies of non-cylindrical helical springs. *Journal of Sound and Vibration*, 252(3):479–491, 2002.

[25] V Yıldırım. Numerical buckling analysis of cylindrical helical coil springs in a dynamic manner. *International Journal Of Engineering & Applied Sciences*, 1(1):20–32, 2009.

[26] Aimin Yu and Changjin Yang. Formulation and evaluation of an analytical study for cylindrical helical springs. *Acta Mechanica Solida Sinica*, 23(1):85–94, 2010.

[27] AM Yu and Y Hao. Free vibration analysis of cylindrical helical springs with noncircular cross-sections. *Journal of Sound and Vibration*, 330(11):2628–2639, 2011.

[28] Jamil M Renno and Brian R Mace. Vibration modelling of helical springs with non-uniform ends. *Journal of Sound and Vibration*, 331(12):2809–2823, 2012.

[29] Krzysztof Michalczyk. Dynamic stresses in helical springs locally coated with highly-damping material in resonant longitudinal vibration conditions. *International Journal of Mechanical Sciences*, 90:53–60, 2015.

[30] Merve Ermis and Mehmet H Omurtag. Static and dynamic analysis of conical helices based on exact geometry via mixed fem. *International Journal of Mechanical Sciences*, 131:296–304, 2017.

[31] R Champion and WL Champion. Departure from linear mechanical behaviour of a helical spring. *Mathematical and Computer Modelling*, 53(5-6):915–926, 2011.

[32] L Del Llano-Vizcaya, C Rubio-González, Gérard Mesmacque, and T Cervantes-Hernández. Multiaxial fatigue and failure analysis of helical compression springs. *Engineering failure analysis*, 13(8):1303–1313, 2006.

[33] Riccardo Bartolozzi and Francesco Frendo. Stiffness and strength aspects in the design of automotive coil springs for mcpherson front suspensions: a case study. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 225(10):1377–1391, 2011.

[34] B Ravi Kumar, Swapan K Das, and DK Bhattacharya. Fatigue failure of helical compression spring in coke oven batteries. *Engineering Failure Analysis*, 10(3):291–296, 2003.

[35] Christina Berger and B Kaiser. Results of very high cycle fatigue tests on helical compression springs. *international journal of fatigue*, 28(11):1658–1663, 2006.

[36] Youli Zhu, Yanli Wang, and Yuanlin Huang. Failure analysis of a helical compression spring for a heavy vehicle's suspension system. *Case Studies in Engineering Failure Analysis*, 2(2):169–173, 2014.

[37] Goran Vukelic and Marino Brcic. Failure analysis of a motor vehicle coil spring. *Procedia Structural Integrity*, 2:2944–2950, 2016.

[38] Reza Mirzaeifar, Reginald DesRoches, and Arash Yavari. A combined analytical, numerical, and experimental study of shape-memory-alloy helical springs. *International Journal of Solids and Structures*, 48(3-4):611–624, 2011.

[39] Donghai Qiu, Sébastien Seguy, and Manuel Paredes. Tuned nonlinear energy sink with conical spring: design theory and sensitivity analysis. *Journal of Mechanical Design*, 140(1):011404, 2018.

[40] Sid Ali Kaoua, Kamel Taibi, Nacera Benghanem, Krimo Azouaoui, and Mohammed Azzaz. Numerical modelling of twin helical spring under tensile loading. *Applied Mathematical Modelling*, 35(3):1378–1387, 2011.

[41] Majdi Gzal, Morel Groper, and Oleg Gendelman. Analytical, experimental and finite element analysis of elliptical cross-section helical spring with small helix angle under static load. *International Journal of Mechanical Sciences*, 130:476–486, 2017.

[42] M Paredes, M Sartor, and C Masclet. Obtaining an optimal compression spring design directly from a user specification. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 216(3):419–428, 2002.

[43] Mohamed Taktak, Khalifa Omheni, Abdessattar Aloui, Fakhreddine Dammak, and Mohamed Haddar. Dynamic optimization design of a cylindrical helical spring. *Applied Acoustics*, 77:178–183, 2014.

[44] Mohamed Taktak, Fakhreddine Dammak, Said Abid, and Mohamed Haddar. A finite element for dynamic analysis of a cylindrical isotropic helical spring. *Journal of Mechanics of Materials and Structures*, 3(4):641–658, 2008.

[45] Byung Chul Choi, Seunghyeon Cho, and Chang-Wan Kim. Kriging model based optimization of macpherson strut suspension for minimizing side load using flexible multi-body dynamics. *International Journal of Precision Engineering and Manufacturing*, 19(6):873–879, 2018.

[46] Max MJ Opgenoord, Douglas L Allaire, and Karen E Willcox. Variance-based sensitivity analysis to support simulation-based design under uncertainty. *Journal of Mechanical Design*, 138(11):111410, 2016.

[47] Abaqus Users Manual. Version 6.16. *Dassault Systemes Simulia Corp, Providence*, 2016.

[48] Changchang Wu et al. VisualSFM: A visual structure from motion system. http://ccwu.me/vsfm/, 2011.

[49] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *CVPR 2011*, pages 3057–3064. IEEE, 2011.

[50] Joseph Edward Shigley. Mechanical engineering design. 1972.

[51] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[52] Robert Tibshirani, Martin Wainwright, and Trevor Hastie. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2015.

[53] Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

[54] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

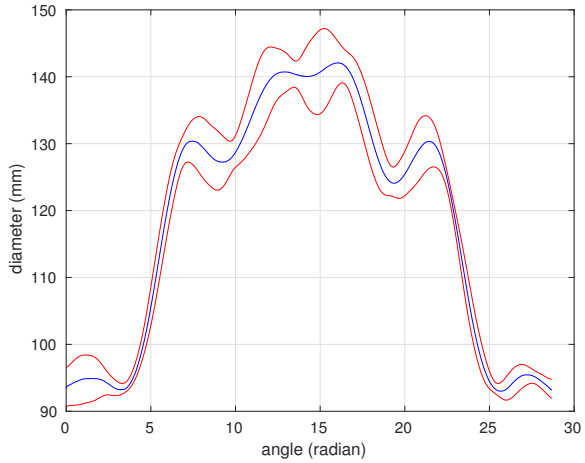
[55] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[56] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, March 2014.

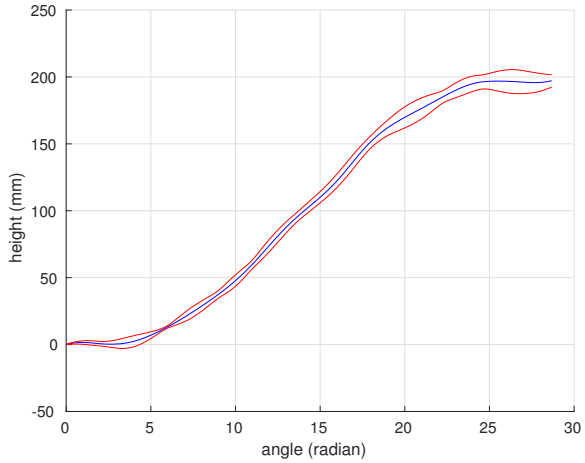
[57] Michael Grant. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. http://stanford.edu/~boyd/graph_dcp.html.

A. ADDITIONAL FIGURES

Additional Plots for Profile Variations

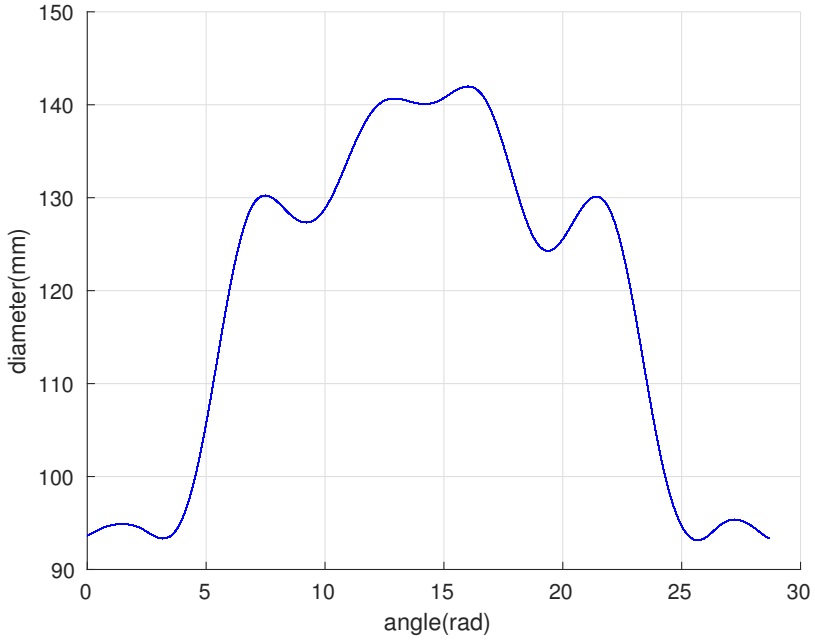


(a) Diameter profiles

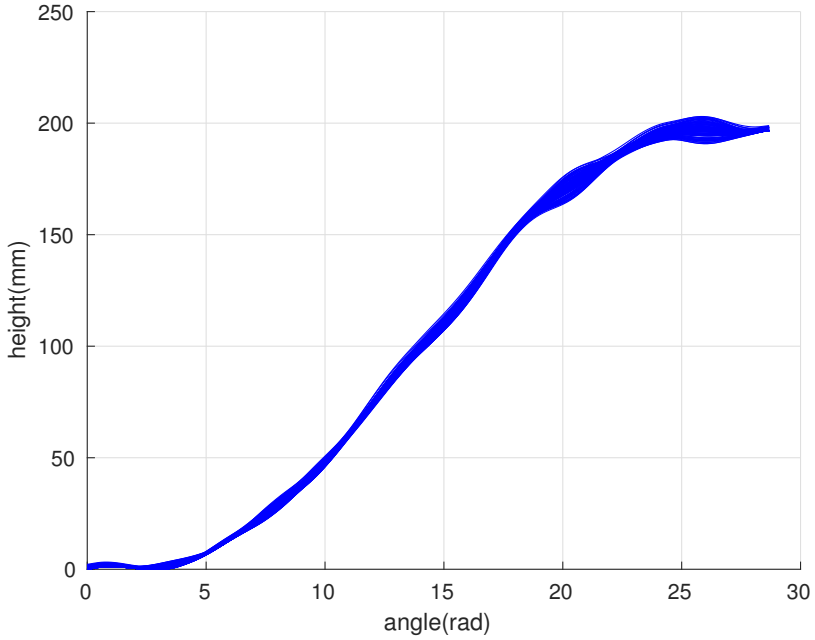


(b) Height profiles

Figure A.1. Bounds for height variations only for experiment 2 for 4x the max variations.

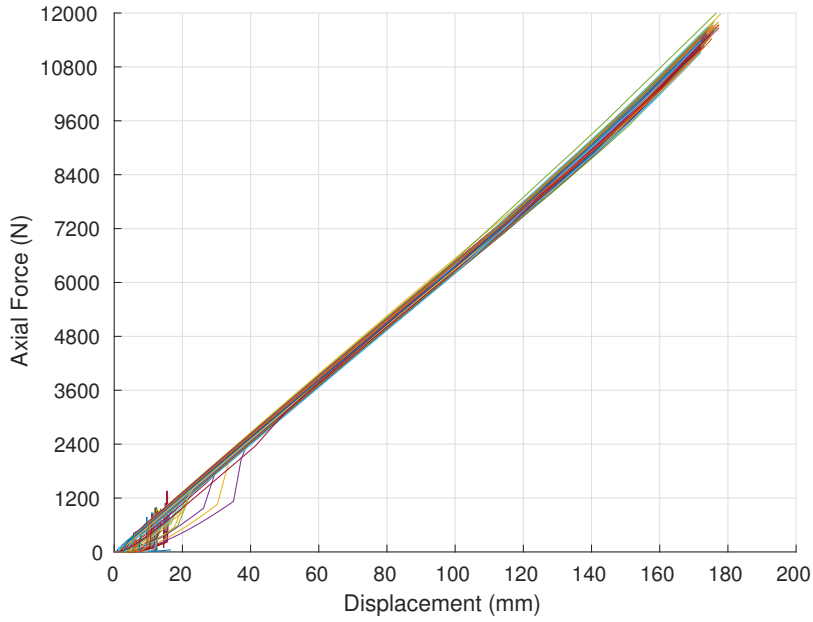


(a) Diameter profiles

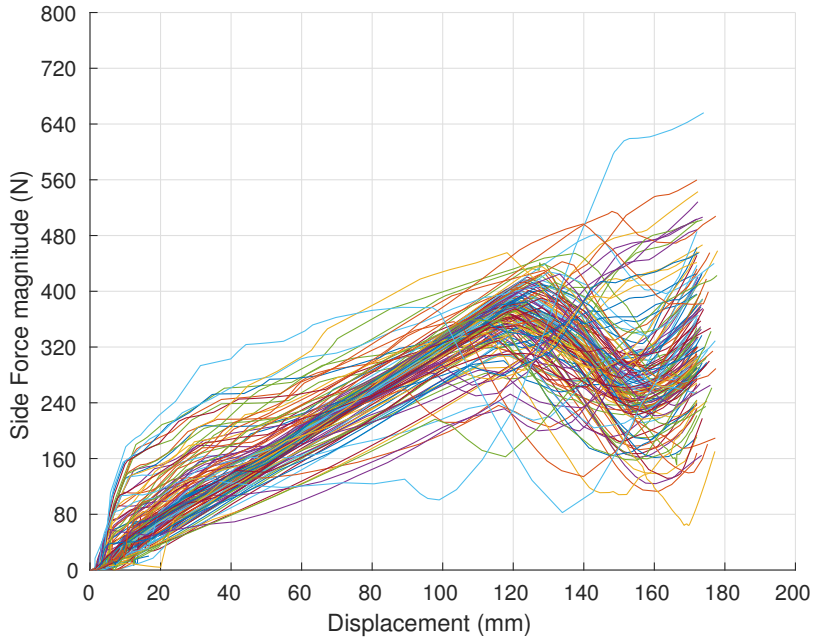


(b) Height profiles

Figure A.2. Profiles generated for height variations only for experiment 2 for 4x the max variations.



(a) Axial Force



(b) Side Force

Figure A.3. Plot of forces for height variations only for experiment 2 for 4x the max variations.

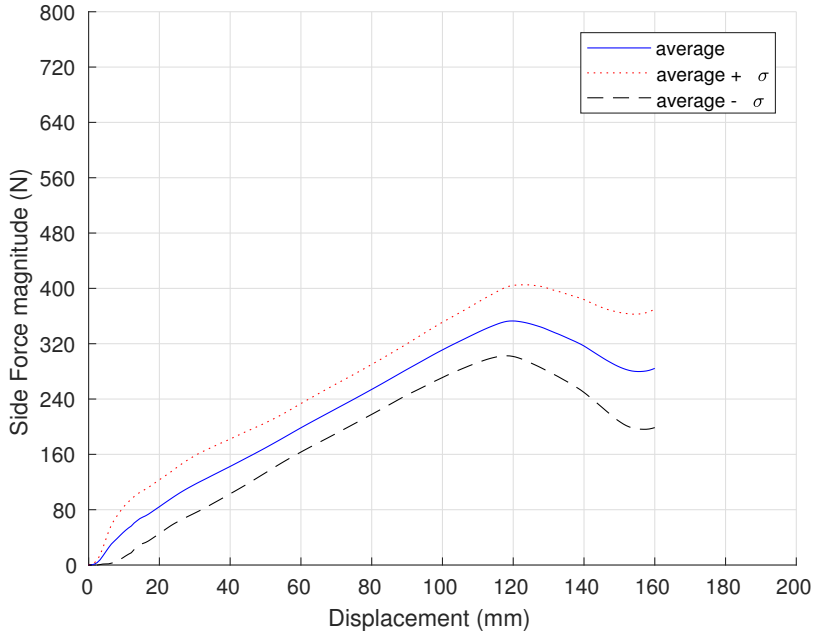


Figure A.4. Plot of averaged side force for height variations only for experiment 2 for 4x the max variations.

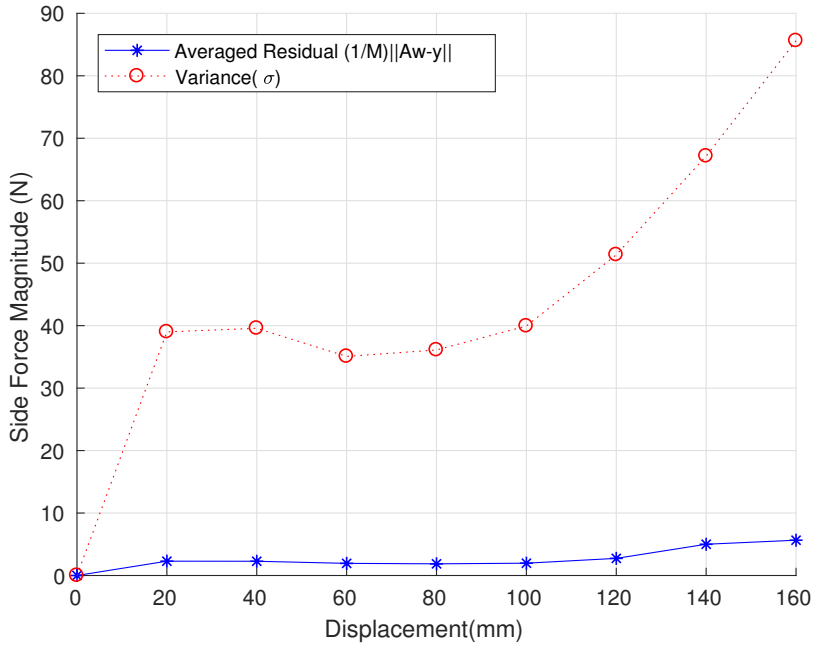
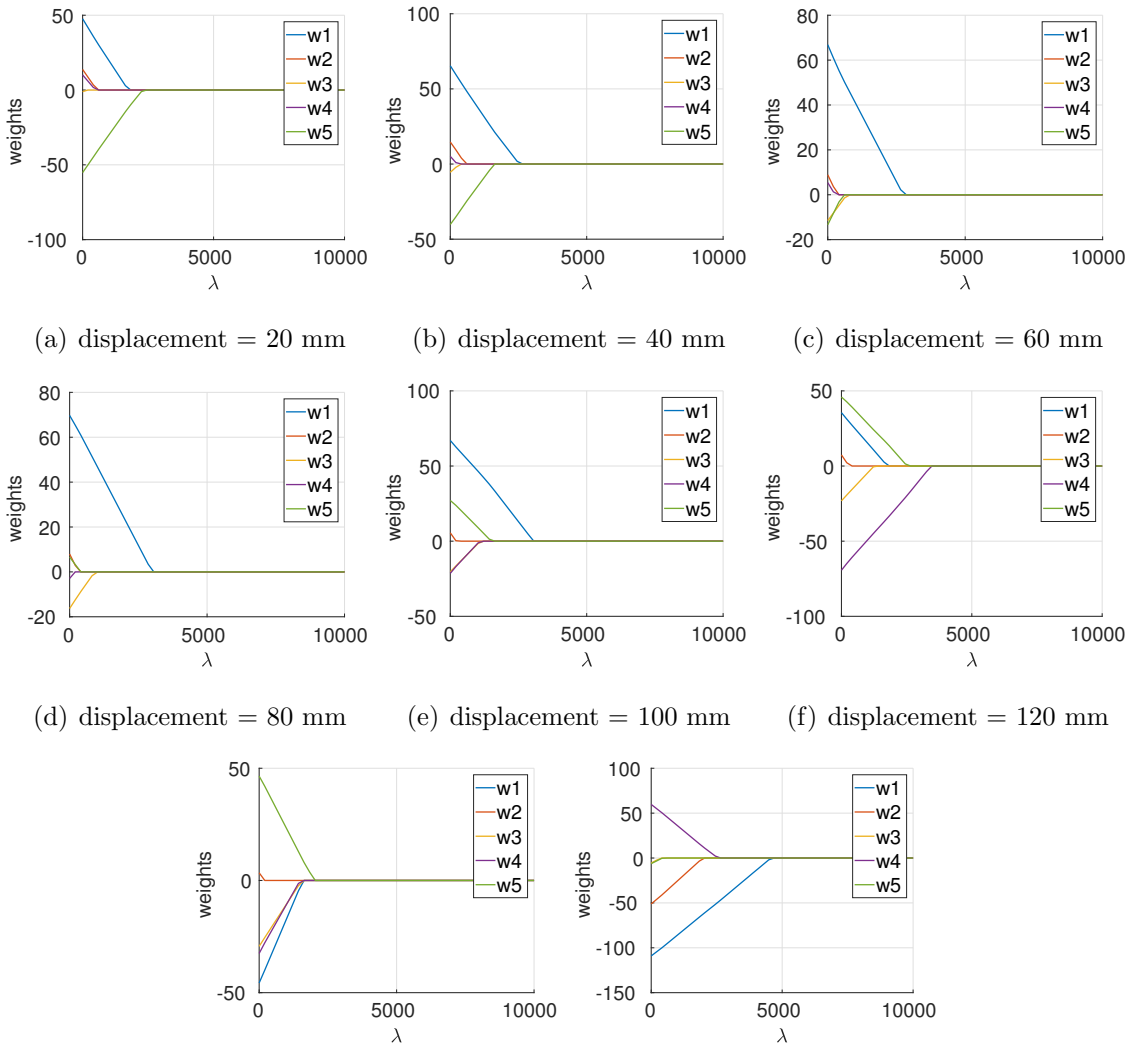


Figure A.5. average error vs variance for 4x max variations.

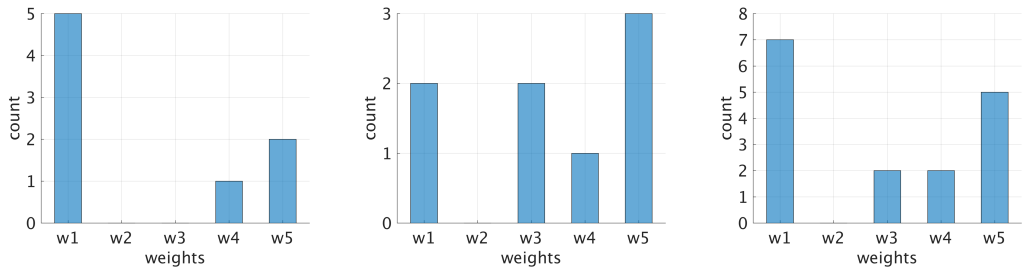


(a) displacement = 20 mm (b) displacement = 40 mm (c) displacement = 60 mm

(d) displacement = 80 mm (e) displacement = 100 mm (f) displacement = 120 mm

(g) displacement = 140 mm (h) displacement = 160 mm

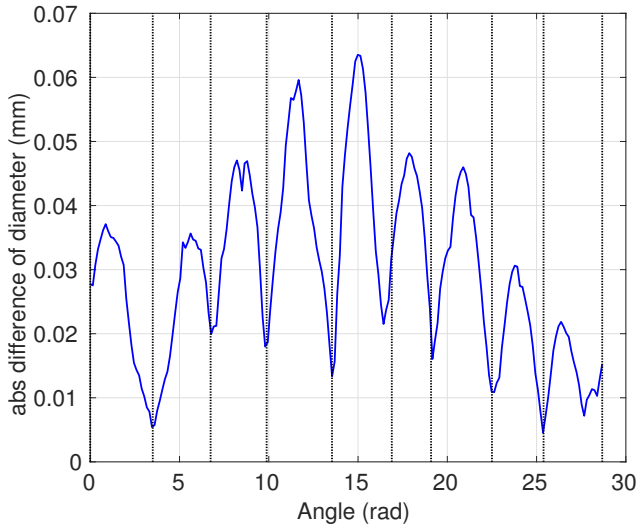
Figure A.6. Regression results for 4x max variations.



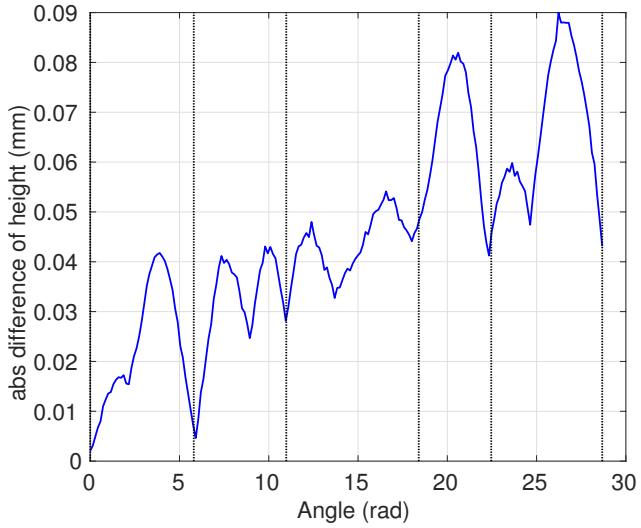
(a) Top factor (b) Second factor (c) Top two factors

Figure A.7. Histogram for Top, second and top two factors for 4x max variations.

Figures for Region-based Profile Approximation

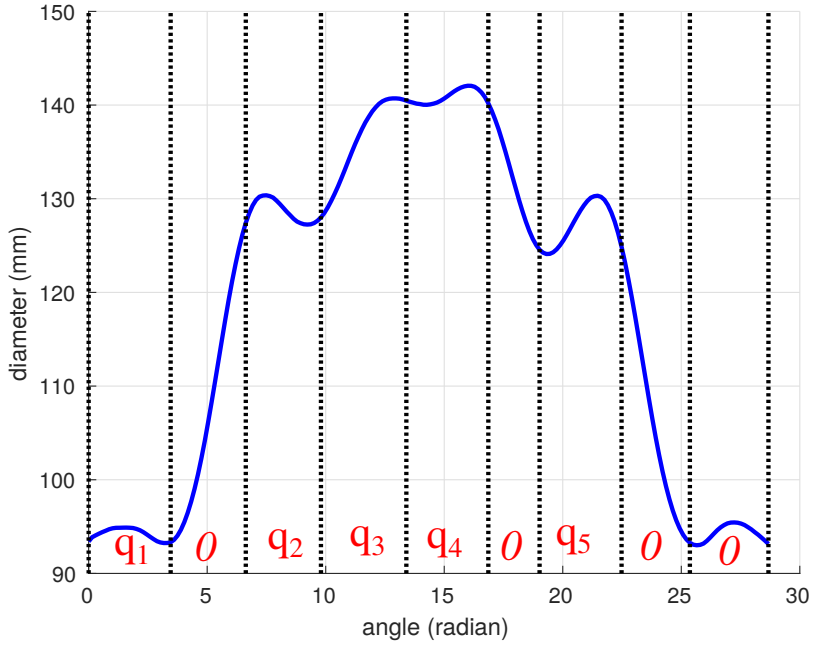


(a) Diameter regions

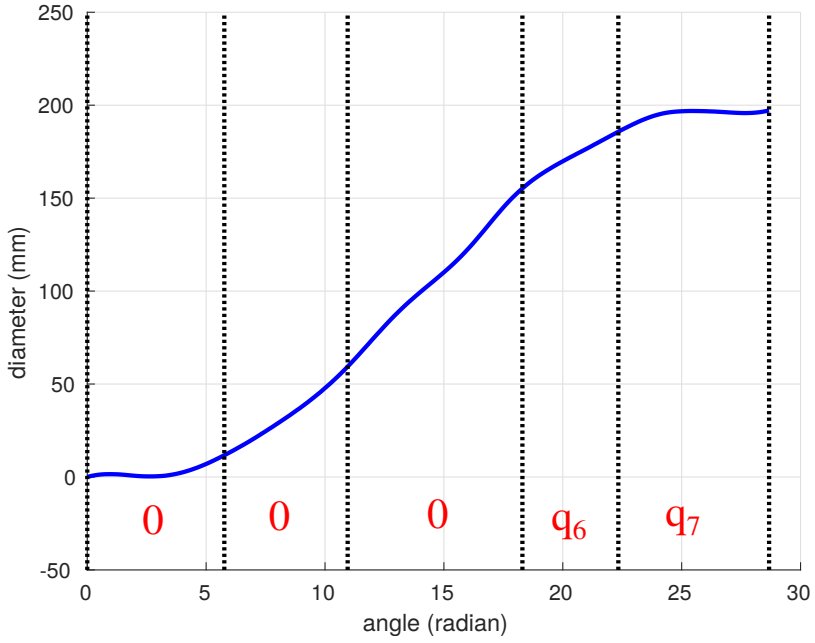


(b) Height regions

Figure A.8. Plot of absolute difference of average profile with bounds where the vertical black lines mark the boundaries of regions selected for variation.

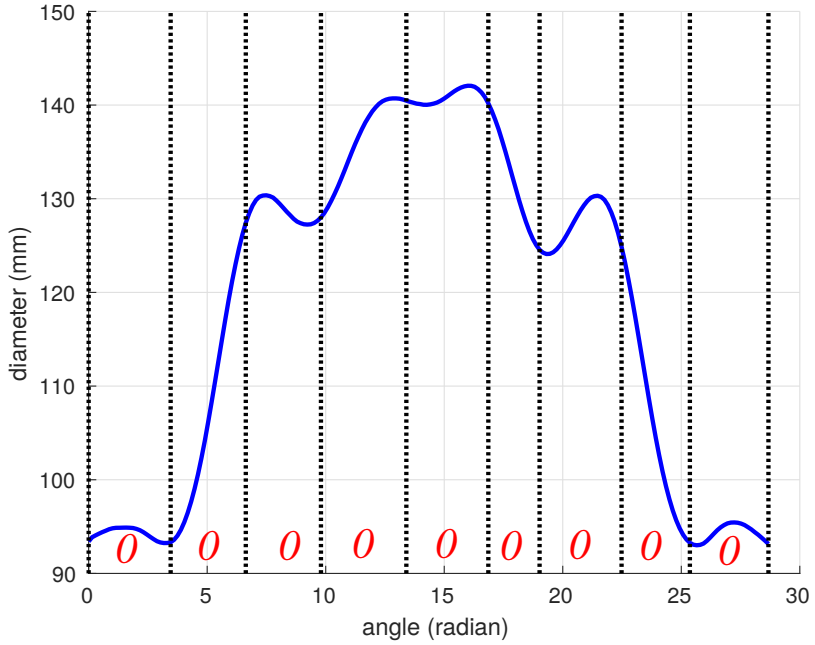


(a) Random variables for diameter regions

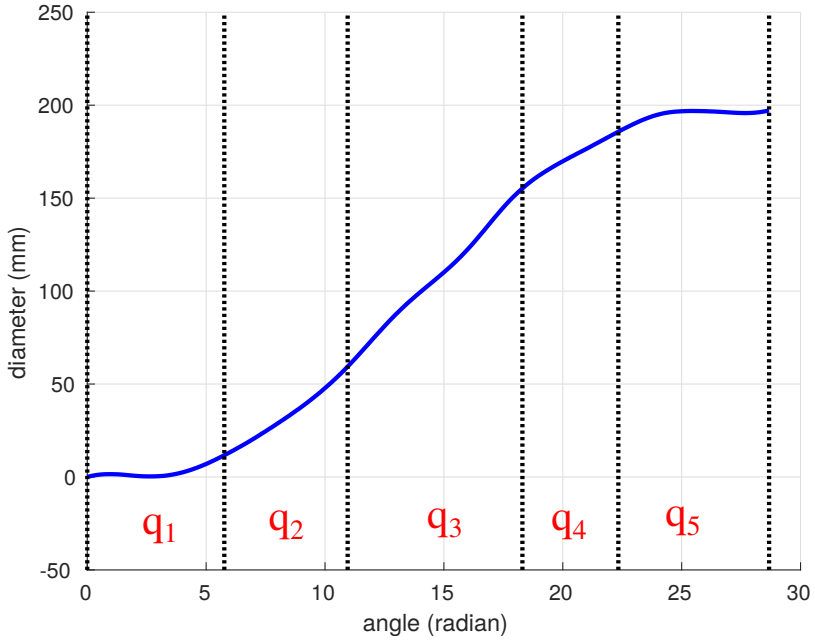


(b) Random variables for height regions

Figure A.9. Choice of random variables experiment 1.

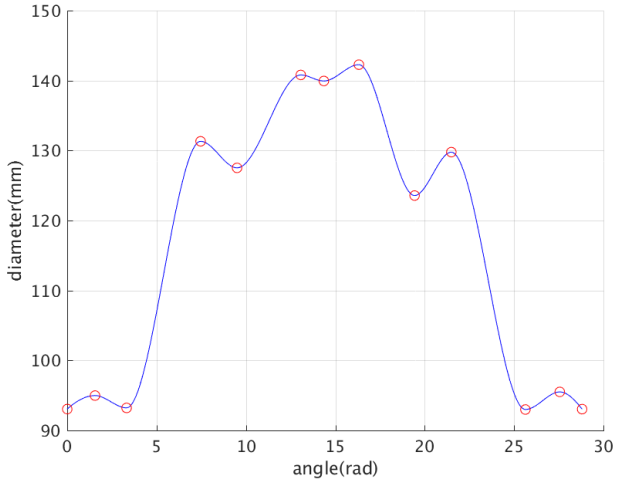


(a) Random variables for diameter regions

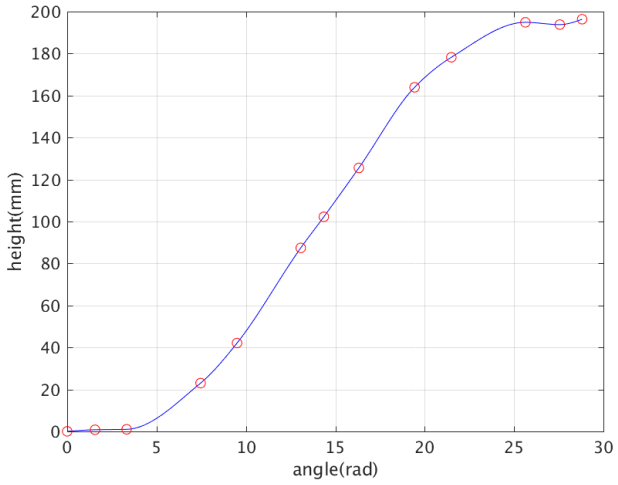


(b) Random variables for height regions

Figure A.10. Choice of random variables for experiment 2.

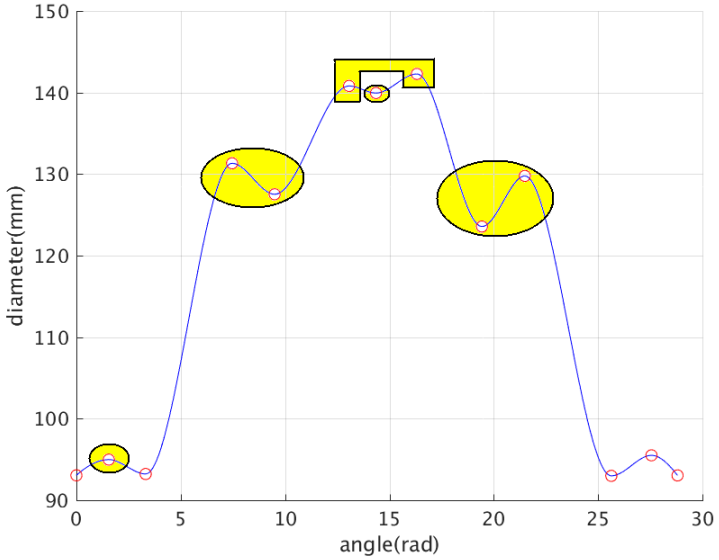


(a) Diameter profile

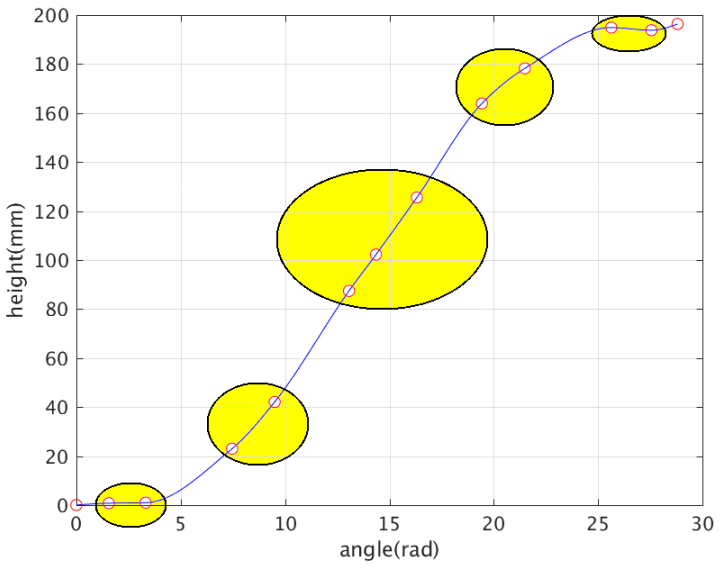


(b) Height profile

Figure A.11. Control points for specimen 5 in red circles for spline based approximation.

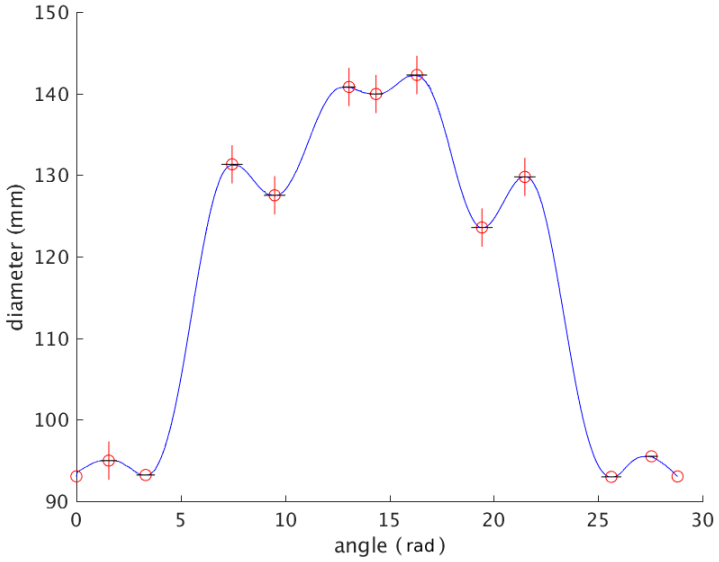


(a) Diameter and angle groups

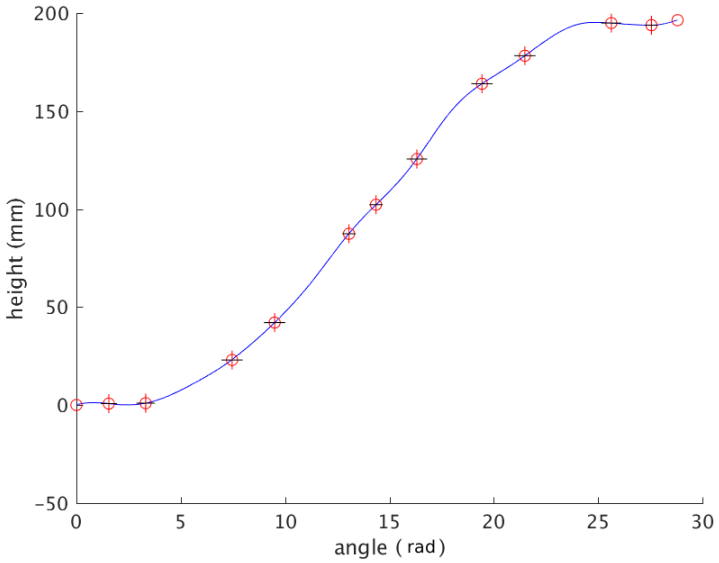


(b) Height groups

Figure A.12. Groups of random variables for spline based approximation (highlighted in yellow).

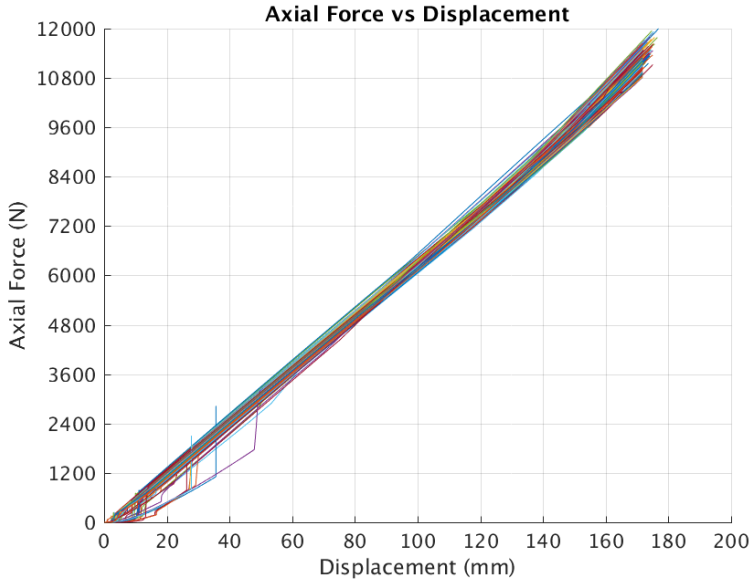


(a) Bounds on diameter and angle

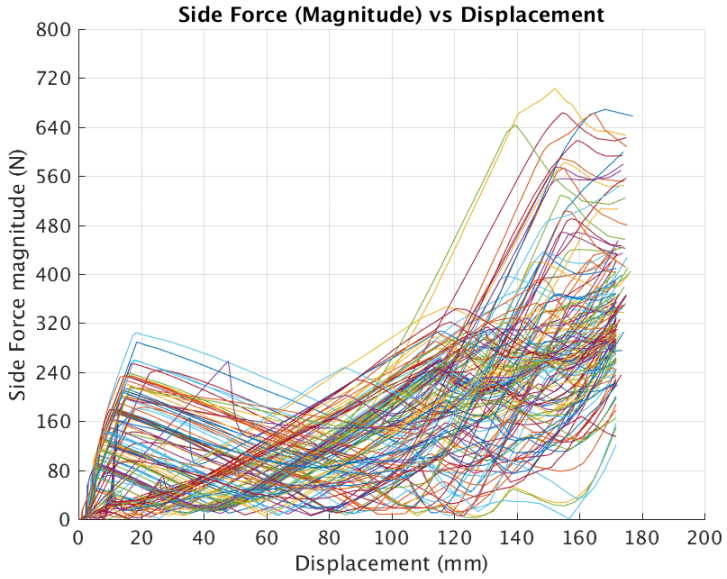


(b) Bounds on height and angles

Figure A.13. Bounds on control points for spline based approximation.



(a) Axial Force



(b) Side Force

Figure A.14. Plot of forces for height variations only for experiment 2.

B. MODEL SCRIPTS

In this section we are attaching scripts for the two models discussed in section 3. In order to run the scripts one needs to have access to `ABAQUS` [47] where you can execute the python scripts. The scripts written out here were made for `ABAQUS CAE 2017` with `python 2.7`. In order to execute the scripts follow these instructions:

1. Create a `'New folder'` where you want to execute the program and copy its path
2. Copy the text written out in section B to a text file and save the file in the `'New folder'` with the specific name of `coordinatefile.txt`.
3. Copy the contents of the script in section B or B into another file and save it in the `'New folder'` with `*.py` extension.
4. Edit the `*.py` file by pasting the path of the `'New folder'` inside the quotes in line 8 of the code.
5. Open `ABAQUS CAE 2017`, go to `File -> Run Script` and select the script to execute it.

Coordinate File

```
2 1 39.784908 0.100000 0.000000
2 2 39.544808 5.902103 0.472336
2 3 38.408358 11.618066 0.937341
2 4 36.409094 17.107197 1.286106
2 5 33.631749 22.259785 1.539973
2 6 30.116271 26.952339 1.697142
2 7 25.982204 31.127997 1.760289
2 8 21.235102 34.605921 1.804744
2 9 16.025562 37.349729 1.728804
2 10 10.468131 39.297962 1.663882
2 11 4.687208 40.421188 1.601790
2 12 -1.195275 40.694184 1.433389
2 13 -7.053892 40.107094 1.215151
2 14 -12.762370 38.672036 0.985273
2 15 -18.178405 36.381834 0.757424
2 16 -23.202682 33.344372 0.557278
2 17 -27.709739 29.597843 0.413107
2 18 -31.601675 25.234100 0.329422
2 19 -34.808057 20.363593 0.270015
2 20 -37.275997 15.098157 0.233439
2 21 -38.955778 9.546777 0.238133
2 22 -39.871827 3.830967 0.265867
2 23 -39.948810 -1.950949 0.241097
2 24 -39.262037 -7.691166 0.430703
2 25 -37.791207 -13.289981 0.645801
2 26 -35.598171 -18.667049 0.911792
2 27 -32.712616 -23.744146 1.232024
2 28 -29.157549 -28.435069 1.624506
2 29 -24.941194 -32.615067 2.068109
2 30 -20.232858 -36.372324 2.624158
2 31 -14.964939 -39.476518 3.194548
2 32 -9.233632 -41.889322 3.839107
2 33 -3.119238 -43.556598 4.533399
```

```
2 34 3.293813 -44.402337 5.301997
2 35 9.909144 -44.387755 6.099754
2 36 16.611008 -43.434793 6.949193
2 37 23.261693 -41.495315 7.848821
2 38 29.706268 -38.545312 8.801488
2 39 35.771564 -34.583523 9.800194
2 40 41.360380 -29.705749 10.764008
2 41 46.123518 -23.848648 11.812902
2 42 50.109139 -17.252434 12.890833
2 43 53.018931 -9.976939 14.018307
2 44 54.780142 -2.217515 15.128207
3 45 55.343396 5.834704 16.244581
3 46 54.613903 13.973599 17.383049
3 47 52.584477 21.977676 18.531093
3 48 49.277842 29.620765 19.706598
3 49 44.769054 36.687668 20.918628
3 50 39.185058 42.995962 22.052649
3 51 32.628880 48.324468 23.290893
3 52 25.319623 52.579160 24.489722
3 53 17.460796 55.708168 25.667602
3 54 9.226721 57.632224 26.825839
3 55 0.805249 58.265267 27.963758
3 56 -7.596984 57.634944 29.124541
3 57 -15.792363 55.783297 30.289727
3 58 -23.604930 52.763705 31.492948
3 59 -30.881033 48.668973 32.672870
3 60 -37.480046 43.594279 33.866632
3 61 -43.269343 37.643135 35.130545
3 62 -48.189945 30.976057 36.279320
3 63 -52.098334 23.679688 37.575586
3 64 -54.983921 15.924612 38.704112
3 65 -56.673269 7.824423 40.256841
3 66 -57.242629 -0.432242 41.523965
3 67 -56.685065 -8.702646 42.961504
3 68 -54.973810 -16.835097 44.559890
```

```
3 69 −52.133418 −24.674607 46.133340
3 70 −48.237588 −32.089110 47.695164
3 71 −43.311131 −38.917781 49.346016
3 72 −37.441217 −45.024747 51.027467
3 73 −30.722061 −50.272251 52.779760
3 74 −23.271581 −54.533730 54.622008
3 75 −15.231936 −57.706571 56.487119
3 76 −6.759497 −59.703152 58.404316
3 77 1.974056 −60.448511 60.373088
3 78 10.777490 −59.881101 62.322380
3 79 19.466877 −58.052868 64.468779
3 80 27.833003 −54.922127 66.544159
3 81 35.681379 −50.557515 68.674104
3 82 42.838425 −45.060167 70.766235
3 83 49.128261 −38.532728 72.881757
3 84 54.397026 −31.113473 75.005924
3 85 58.509082 −22.963387 77.054714
3 86 61.400348 −14.277229 79.126494
3 87 62.964459 −5.235124 81.149232
3 88 63.168966 3.955151 83.145953
3 89 62.016746 13.087356 85.101372
3 90 59.509264 21.951024 87.058504
3 91 55.737826 30.358609 88.901789
3 92 50.780122 38.128237 90.715556
3 93 44.750692 45.100608 92.486608
3 94 37.770415 51.121897 94.215273
3 95 29.992595 56.070337 95.874349
3 96 21.581458 59.843697 97.489189
3 97 12.712187 62.358618 99.116847
3 98 3.572456 63.564788 100.679137
3 99 −5.645989 63.439719 102.236966
3 100 −14.759121 62.021384 103.746600
3 101 −23.583884 59.314545 105.269233
3 102 −31.942336 55.373242 106.811231
3 103 −39.659989 50.267196 108.365522
```

```
3 104 −46.576458 44.097529 109.928189
3 105 −52.526689 36.973000 111.513450
3 106 −57.385418 29.048685 113.160075
3 107 −61.037266 20.488652 114.866974
3 108 −63.379660 11.474167 116.613884
3 109 −64.387440 2.210513 118.398790
3 110 −64.026995 −7.103497 120.268343
3 111 −62.300428 −16.262901 122.191136
3 112 −59.227110 −25.057049 124.161319
3 113 −54.862388 −33.272825 126.223602
3 114 −49.326667 −40.724035 128.332718
3 115 −42.747848 −47.232439 130.484828
3 116 −35.279730 −52.633803 132.709414
3 117 −27.130868 −56.838247 134.888615
3 118 −18.484675 −59.725900 137.127136
3 119 −9.563130 −61.252690 139.353083
3 120 −0.586876 −61.436305 141.521185
3 121 8.238692 −60.283243 143.629609
3 122 16.714682 −57.854009 145.689390
3 123 24.655180 −54.227032 147.733026
3 124 31.905003 −49.531363 149.689201
3 125 38.338811 −43.907576 151.543242
3 126 43.820529 −37.469389 153.397056
3 127 48.326821 −30.422045 155.048864
3 128 51.752566 −22.877824 156.711311
3 129 54.090796 −15.007141 158.311404
3 130 55.325886 −6.955130 159.834388
3 131 55.477157 1.137251 161.279655
3 132 54.537028 9.137837 162.678278
3 133 52.564717 16.923473 163.980020
3 134 49.572389 24.368388 165.245198
3 135 45.566882 31.323230 166.465739
3 136 40.673785 37.702285 167.658799
3 137 34.935265 43.370610 168.816476
3 138 28.441829 48.215111 169.923657
```

```
3 139 21.298477 52.126354 170.960740
3 140 13.632009 55.014828 172.022643
3 141 5.580204 56.781781 173.046148
3 142 -2.693812 57.370890 173.995420
3 143 -11.011454 56.720075 175.093367
3 144 -19.190798 54.854437 175.975448
3 145 -27.036602 51.768551 176.873306
3 146 -34.353948 47.512935 177.765894
3 147 -40.951529 42.170409 178.680661
3 148 -46.634588 35.847955 179.604715
3 149 -51.408198 28.800549 180.327675
3 150 -54.876788 21.030396 181.273535
3 151 -57.063860 12.835906 182.205896
3 152 -57.982386 4.446062 183.071618
3 153 -57.565020 -3.933228 183.934296
3 154 -55.850414 -12.079447 184.781407
3 155 -52.895407 -19.778319 185.661384
3 156 -48.804019 -26.831610 186.524413
4 157 -43.701214 -33.055385 187.322762
4 158 -37.810523 -38.354705 188.164415
4 159 -31.271396 -42.599543 188.980642
4 160 -24.278512 -45.721193 189.793702
4 161 -17.027920 -47.682406 190.609379
4 162 -9.722281 -48.517096 191.394286
4 163 -2.535611 -48.285275 192.084500
4 164 4.380446 -47.008598 192.760051
4 165 10.898126 -44.848869 193.458364
4 166 16.903130 -41.844125 194.047174
4 167 22.315236 -38.128388 194.587869
4 168 27.072212 -33.809814 195.077379
4 169 31.127988 -28.992979 195.508225
4 170 34.445297 -23.775756 195.885240
4 171 36.993684 -18.253823 196.139398
4 172 38.786262 -12.535734 196.340519
4 173 39.796467 -6.702028 196.594926
```

```
4 174 40.037720 −0.845426 196.816674
4 175 39.514322 4.947865 197.010743
4 176 38.246874 10.592795 197.129117
4 177 36.249204 16.002482 197.202288
4 178 33.552479 21.090997 197.249655
4 179 30.188527 25.767010 197.298943
4 180 26.223466 29.959949 197.327778
4 181 21.712520 33.590661 197.337388
4 182 16.719874 36.575686 197.325382
4 183 11.327746 38.837141 197.302211
4 184 5.632083 40.281744 197.264770
4 185 −0.240609 40.863429 197.208945
4 186 −6.156027 40.561289 197.096525
4 187 −11.968486 39.345752 196.965607
4 188 −17.530228 37.244853 196.828388
4 189 −22.714432 34.332464 196.678277
4 190 −27.390399 30.666537 196.538808
4 191 −31.436934 26.331633 196.469494
4 192 −34.781581 21.456580 196.428120
4 193 −37.370121 16.164465 196.411970
4 194 −39.178090 10.579054 196.387705
4 195 −40.125466 4.805621 196.477168
4 196 −40.243294 −1.021561 196.602311
4 197 −39.531955 −6.786193 196.769553
4 198 −37.961809 −12.356891 197.063770
4 199 −35.615426 −17.621139 197.355275
4 200 −32.456762 −22.409020 197.602627
```

Continuum Model

```
#--Inputs --#
"""
Variables and inputs
"""
#set working directory
import os, time
time_in= time.time()
mainfilepath =''# to be given by user
os.chdir(mainfilepath)
notinterferingfile  = open(mainfilepath+'/notinterfering.txt','w')
notinterferingfile.write("1")
notinterferingfile.close()
#----spring parameters-----#
#note dimensions are in mm we are using SI 'mm' units so for stress we have MPa, Mass-tonne, density-tonne/mm**3
wire_dia=13.5#wire diameter (mm)
spring_initial_radius=38.5 #initial radius of the spring (mm)
spring_max_radius=64.2819#maximum radius of the spring (mm)
spring_height=197.6026 #height of the spring (mm)
spring_Np=200#number of points in the spring
#--------parameters of the plates--------#
plate_thickness=10 #plate thickness
bottom_center= (0.0,0.0,0.0) #bottom plate center coordinate(tuple)#xyz to be in accordance with abaqus (loading in Y direction)
top_center= (0.0,spring_height+2*plate_thickness+wire_dia,0.0) #top plate center coordinates(tuple)#xyz to be in accordance with abaqus (loading in Y direction)
plate_radius=spring_max_radius*2
#-----------------#
#Material properties variables (independent definition)
springyoungmodulus=250000#Mpa
poissonratio=0.3
springDisplacement=-173.0026#mm
```

```
30 plateyoungmodulus=207.0E3#MPa
31 bottomplatemeshSize=3
32 topplatemeshSize=3
33 springmeshSize=3
34 eps=10**-6
35 friction_coeff=0.7
36 #----Read Coordinate Text File----#
37 """
38 Read xyz coordinates of the spring from the file
39 (This file is assumed to be present in mainfilepath)
40 """
41 ##NOTE: coordinatefile.txt to give xyz coordinates such that the Z
       coordinates are in axial direction,
42 #we are doing a manipulation by interchangin Y and Z coordinates here
43 filename='coordinatefile'
44 filepath= mainfilepath + '/' + filename + '.txt'
45 fid=open(filepath,'r')
46 spring_coord = []; spring_identifier=[]
47 for i in range(spring_Np):
48   temp= fid.readline().split()
49   temp=[float(j) for j in temp]#convert the coordinate strings to float
         and rotate using matrix [1 0 0, 0 0 1, 0 -1 0]
50   spring_identifier.append(temp[0])
51   a=temp[-2]
52   temp[-2]=temp[-1]+plate_thickness+(wire_dia/2.0)#displacing to correct
         the height
53   temp[-1]=-a
54   spring_coord.append(temp[2:])
55 fid.close
56 #--------#
57
58 #-------------------------#
59 #---ABAQUS Main File------#
60 #-------------------------#
61 """
```

```
62 Abaqus scripting begins
63 """
64 from abaqus import *
65 from abaqusConstants import *
66 #creating a model
67 springModel=mdb.Model(name='Spring')
68 #--Parts-------#
69 """
70 Create all geometric parts and prepare them for adequate meshing by
       partitioning the surfaces
71 """
72 import part
73 #----part for bottom plate-----#
74 #sketch of bottom plate
75 bottomplateSketch= springModel.ConstrainedSketch(name = 'Bottom Plate',
       sheetSize=2*plate_radius)
76 temp = (bottom_center[0:2],(bottom_center[0],bottom_center[1]+
       plate_thickness+wire_dia),(bottom_center[0]+spring_initial_radius-
       wire_dia/2.0,bottom_center[1]+plate_thickness+wire_dia),
77      (bottom_center[0]+spring_initial_radius-wire_dia/2.0,bottom_center
           [1]+plate_thickness),(bottom_center[0]+plate_radius,
           bottom_center[1]+plate_thickness),
78      (bottom_center[0]+plate_radius,bottom_center[1]),bottom_center[0:2])
79 for i in range(len(temp)-1):
80   bottomplateSketch.Line(temp[i],temp[i+1])
81 bottomplateSketch.ConstructionLine(bottom_center[0:2],(bottom_center[0],
       bottom_center[1]+plate_thickness+wire_dia))
82 #create part for bottomplate
83 bottomplatePart = springModel.Part(name='Bottom plate',dimensionality=
       THREE_D, type=DEFORMABLE_BODY)
84 #revolve extrude bottom plate
85 bottomplatePart.BaseSolidRevolve(sketch=bottomplateSketch,angle=360)
86 #-------#
87 ##code for partitioning the Bottom Plate to get desired elements and
       node locations
```

```
88  bottomplatePart.DatumPointByCoordinate((bottom_center[0],bottom_center
        [1]+plate_thickness,bottom_center[2]))#center+plate thickness in y
        direction
89  bottomplatePart.DatumPointByCoordinate((bottom_center[0],bottom_center
        [1]+plate_thickness+wire_dia,bottom_center[2]))#y center of top face
         of bottom pug
90  bottomplatePart.DatumPointByCoordinate((bottom_center[0]+plate_radius,
        bottom_center[1]+plate_thickness,bottom_center[2]))#x+plate
        thickness in y direction
91  ptz=bottomplatePart.DatumPointByCoordinate((bottom_center[0],
        bottom_center[1]+plate_thickness,bottom_center[2]+plate_radius))#z+
        plate thickness in y direction
92  ptc=bottomplatePart.datums[2]## index 1 is reserved for the datum axis
93  pty=bottomplatePart.datums[3]
94  ptx=bottomplatePart.datums[4]
95  ptz=bottomplatePart.datums[5]
96  bottomplatePart.PartitionCellByPlaneThreePoints(cells=bottomplatePart.
        cells,point1=ptc,point2=ptx,point3=pty)
97  bottomplatePart.PartitionCellByPlaneThreePoints(cells=bottomplatePart.
        cells,point1=ptc,point2=ptz,point3=pty)
98  bottomplatePart.PartitionCellByPlaneThreePoints(cells=bottomplatePart.
        cells,point1=ptc,point2=ptz,point3=ptx)
99  #----part for top plate-------#
100 #sketch of Top plate
101 topplateSketch= springModel.ConstrainedSketch(name = 'Top Plate',
        sheetSize=2*plate_radius)
102 temp = (top_center[0:2],(top_center[0],top_center[1]-plate_thickness-
        wire_dia),(top_center[0]+spring_initial_radius-wire_dia/2.0,
        top_center[1]-plate_thickness-wire_dia),\
103   (top_center[0]+spring_initial_radius-wire_dia/2.0,top_center[1]-
          plate_thickness),(top_center[0]+plate_radius,top_center[1]-
          plate_thickness),\
104     (top_center[0]+plate_radius,top_center[1]),top_center[0:2])
105 for i in range(len(temp)-1):
106   topplateSketch.Line(temp[i],temp[i+1])
```

```
107 topplateSketch.ConstructionLine(top_center[0:2],(top_center[0],
        top_center[1]-plate_thickness-wire_dia))
108 #create part for topplate
109 topplatePart = springModel.Part(name='top plate',dimensionality= THREE_D
        , type=DEFORMABLE_BODY)
110 #revolve extrude top plate
111 topplatePart.BaseSolidRevolve(sketch =topplateSketch,angle=360)
112 #-----------------#
113 ##code for partitioning top Plate
114 topplatePart.DatumPointByCoordinate((top_center[0],top_center[1]-
        plate_thickness,top_center[2]))#center-plate thickness in y
        direction
115 topplatePart.DatumPointByCoordinate((top_center[0],top_center[1]-
        plate_thickness-wire_dia,top_center[2]))#pty center of top face of
        bottom pug
116 topplatePart.DatumPointByCoordinate((top_center[0]+plate_radius,
        top_center[1]-plate_thickness,top_center[2]))#ptx-plate thickness in
         y direction
117 ptz=topplatePart.DatumPointByCoordinate((top_center[0],top_center[1]-
        plate_thickness,top_center[2]+plate_radius))#ptz-plate thickness in
        y direction
118 ptc=topplatePart.datums[2]## index 1 is reserved for the datum axis
119 pty=topplatePart.datums[3]
120 ptx=topplatePart.datums[4]
121 ptz=topplatePart.datums[5]
122 topplatePart.PartitionCellByPlaneThreePoints(cells=topplatePart.cells,
        point1=ptc,point2=ptx,point3=pty)
123 topplatePart.PartitionCellByPlaneThreePoints(cells=topplatePart.cells,
        point1=ptc,point2=ptz,point3=pty)
124 topplatePart.PartitionCellByPlaneThreePoints(cells=topplatePart.cells,
        point1=ptc,point2=ptz,point3=ptx)
125 #---part for spring-------#
126 #create a part for the spring
127 springPart = springModel.Part(name='Spring',dimensionality= THREE_D,
        type=DEFORMABLE_BODY)
```

```
128 #--Solid Spring----#
129 springPart.WireSpline(points=spring_coord,mergeType=SEPARATE, meshable=
    ON, smoothClosedSpline=OFF)
130 springPart.DatumPointByCoordinate((bottom_center[0],bottom_center[1],
    bottom_center[2]))#center
131 springPart.DatumPointByCoordinate((bottom_center[0],bottom_center[1]+
    plate_thickness+wire_dia,bottom_center[2]))#y center of top face of
    bottom pug
132 p1=springPart.datums[2]
133 p2=springPart.datums[3]
134 springPart.DatumAxisByTwoPoint(point1=p1,point2=p2)
135 springSketch= springModel.ConstrainedSketch(name = 'Spring', sheetSize
    =2*wire_dia)
136 springSketch.CircleByCenterPerimeter(center=(0.0,0.0),point1=(wire_dia
    /2.0,0.0))# in local sketch coordinate system the center is at such
    a position got this from macro recording
137 wireregion = springPart.edges
138 temp=wireregion.getMask()#to convert edge array object to a sequence got
     this from the macro recording
139 wireregion=wireregion.getSequenceFromMask(mask=temp)
140
141 notinterfering = 1
142 try:
143   springPart.SolidSweep(path=wireregion,profile=springSketch,
        sketchUpEdge=springPart.datums[4],sketchOrientation=RIGHT)
144 except AbaqusException:
145   notinterfering = 0
146   #-------------------------#
147   #--Parts and mesh creadted--#
148 if notinterfering:
149   #------Material--#
150   """
151   Create Materials with material properties for plate and spring
152   """
153   import material
```

```
154 #create material
155 springMaterial = springModel.Material(name='Spring Material',
        description='spring steel')
156 plateMaterial = springModel.Material(name='Plate Material',
        description='mild steel')
157 #assign youngs modulus and poissons ratio to both the materials
158 springelasticProperties= (springyoungmodulus,poissonratio)
159 plateelasticProperties= (plateyoungmodulus,poissonratio)
160 springMaterial.Elastic(table=(springelasticProperties,))
161 plateMaterial.Elastic(table=(plateelasticProperties,))
162 #-----Materials Created------#
163
164 #-----Section -----#
165 """
166 Create Sections for the different Properties and assign them to the
        respective parts
167 """
168 import section
169 #creating solid section for plates
170 plateSection = springModel.HomogeneousSolidSection(name='Plate Section
        ',material=plateMaterial.name)
171 #section assignment to plates
172 topplatePart.SectionAssignment(region=(topplatePart.cells,),
        sectionName=plateSection.name)
173 bottomplatePart.SectionAssignment(region=(bottomplatePart.cells,),
        sectionName=plateSection.name)
174 #creating solid section for spring
175 springSection = springModel.HomogeneousSolidSection(name='Spring
        Section',material=springMaterial.name)
176 #section assignements to spring
177 wirecell = springPart.cells
178 wirecell = wirecell.getSequenceFromMask(('[#1]',),)#got this from
        macro reading
179 import regionToolset
180 wireregion = regionToolset.Region(cells = wirecell)
```

```
181 springPart.SectionAssignment(region=wireregion,sectionName =
        springSection.name)
182 #---- Sections created -------#
183
184 #--- Assembly ---#
185 """
186 Assemble instances of the parts into a global coordinate system
187 """
188 import assembly
189 #create part instances
190 springAssembly =springModel.rootAssembly
191 springInstance=springAssembly.Instance(name='Spring Instance',part=
        springPart,dependent=OFF)
192 bottomplateInstance=springAssembly.Instance(name='Bottom Plate
        Instance',part=bottomplatePart,dependent=OFF)
193 topplateInstance=springAssembly.Instance(name='Top Plate Instance',
        part=topplatePart,dependent=OFF)
194 #--- Rotate and Translate spring ---#
195 springAssembly.rotate(instanceList=('Spring Instance',),axisPoint
        =(0.0,115.5526,0.0),axisDirection=(1.0,115.5526,0.0),angle=0)
196 springAssembly.rotate(instanceList=('Spring Instance',),axisPoint
        =(0.0,115.5526,0.0),axisDirection=(0.0,115.5526,-1.0),angle=0)
197 springAssembly.translate(instanceList=('Spring Instance',),vector=(0.0
        ,-0.0024892 , 0.0))
198 springAssembly.translate(instanceList=('Spring Instance',),vector
        =(-0.15, 0.0 , 0.0))
199 springAssembly.translate(instanceList=('Spring Instance',),vector=(
        0.0 , 0.0 , -0.1))
200 #-------------#
201 #-- Assembly Created --#
202
203 #-- Mesh --#
204 """
205 Create Mesh for the instances in the assembly and create sets of nodes
        for application of BC
```

```
206 """
207 ##code for generating mesh on the bottom plate
208 import mesh
209 #select type of element and assign to the instance
210 elemType1 = mesh.ElemType(elemCode=C3D8, elemLibrary=STANDARD,
        secondOrderAccuracy=OFF, distortionControl=DEFAULT)
211 elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=STANDARD)
212 elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=STANDARD)
213 springAssembly.setElementType(regions=(bottomplateInstance.cells,),
        elemTypes=(elemType1,elemType2,elemType3))
214 #seed the instance
215 springAssembly.seedPartInstance(regions=(bottomplateInstance,),size=
        bottomplatemeshSize)
216 ###mesh the instance
217 springAssembly.generateMesh(regions=(bottomplateInstance,))
218 ##create sets for boundary conditions
219 bottomplatecenterSet=bottomplateInstance.nodes.getByBoundingSphere((
        bottom_center[0],bottom_center[1],bottom_center[2]),eps)
220 bottomplateSet=bottomplateInstance.nodes.getByBoundingBox(xMin=
        bottom_center[0]-1.1*plate_radius,yMin=bottom_center[1]-eps,zMin=
        bottom_center[2]-1.1*plate_radius,\
221                                 xMax=bottom_center[0]+1.1*plate_radius,yMax=
                                        bottom_center[1]+eps,zMax=bottom_center
                                        [2]+1.1*plate_radius)
222 bottomplateZSet=bottomplateInstance.nodes.getByBoundingSphere((
        bottom_center[0],bottom_center[1],bottom_center[2]+plate_radius),
        eps)
223 #creating sets
224 bottomplatecenterSet=springAssembly.Set(name='Bottom Plate Center Set'
        ,nodes = bottomplatecenterSet)
225 springAssembly.Set(name='Bottom Plate all Set',nodes = bottomplateSet)
226 bottomplateZSet=springAssembly.Set(name='Bottom Plate Z-Node Set',
        nodes=bottomplateZSet)
```

```
227 springAssembly.SetByBoolean(name='Bottom Plate Set-Z', operation=
        DIFFERENCE, sets=(springAssembly.sets['Bottom Plate all Set'],
        springAssembly.sets['Bottom Plate Z-Node Set'], ))
228 bottomplateSet=springAssembly.SetByBoolean(name='Bottom Plate Set',
        operation=DIFFERENCE, sets=(springAssembly.sets['Bottom Plate Set-
        Z'], springAssembly.sets['Bottom Plate Center Set'], ))
229 del springAssembly.sets['Bottom Plate all Set']
230 del springAssembly.sets['Bottom Plate Set-Z']
231 ##code for generating mesh on the top plate
232 #select type of element and assign to the instance
233 elemType1 = mesh.ElemType(elemCode=C3D8, elemLibrary=STANDARD,
        secondOrderAccuracy=OFF, distortionControl=DEFAULT)
234 elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=STANDARD)
235 elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=STANDARD)
236 springAssembly.setElementType(regions=(topplateInstance.cells,),
        elemTypes=(elemType1,elemType2,elemType3))
237 #seed the instance
238 springAssembly.seedPartInstance(regions=(topplateInstance,),size=
        topplatemeshSize)
239 #mesh the instance
240 springAssembly.generateMesh(regions=(topplateInstance,))
241 ##create sets for boundary conditions
242 topplatecenterSet=topplateInstance.nodes.getByBoundingSphere((
        top_center[0],top_center[1],top_center[2]),eps)
243 topplateSet=topplateInstance.nodes.getByBoundingBox(xMin=top_center
        [0]-1.1*plate_radius,yMin=top_center[1]-10*eps,zMin=top_center
        [2]-1.1*plate_radius,\
244                             xMax=top_center[0]+1.1*plate_radius,yMax=
                                top_center[1]+10*eps,zMax=top_center
                                [2]+1.1*plate_radius)
245 topplateZSet=topplateInstance.nodes.getByBoundingSphere((top_center
        [0],top_center[1],top_center[2]+plate_radius),eps)
246 #creating sets
247 topplatecenterSet=springAssembly.Set(name='Top Plate Center Set',nodes
        = topplatecenterSet)
```

```
248 springAssembly.Set(name='Top Plate all Set',nodes = topplateSet)
249 topplateZSet=springAssembly.Set(name='Top Plate Z-Node Set',nodes=
        topplateZSet)
250 springAssembly.SetByBoolean(name='Top Plate Set-Z', operation=
        DIFFERENCE, sets=(springAssembly.sets['Top Plate all Set'],
        springAssembly.sets['Top Plate Z-Node Set'], ))
251 topplateSet=springAssembly.SetByBoolean(name='Top Plate Set',
        operation=DIFFERENCE, sets=(springAssembly.sets['Top Plate Set-Z'
        ], springAssembly.sets['Top Plate Center Set'], ))
252 del springAssembly.sets['Top Plate all Set']
253 del springAssembly.sets['Top Plate Set-Z']
254 ##code for meshing the spring
255 #select type of element and assign to the instance
256 elemType1 = mesh.ElemType(elemCode=C3D8I,elemLibrary=STANDARD,
        secondOrderAccuracy=OFF, distortionControl=DEFAULT)
257 elemType2 = mesh.ElemType(elemCode=C3D6,elemLibrary=STANDARD)
258 elemType3 = mesh.ElemType(elemCode=C3D4,elemLibrary=STANDARD)
259 springAssembly.setElementType(regions=(springInstance.cells,),
        elemTypes=(elemType1,elemType2,elemType3))
260 #seed the instance
261 springAssembly.seedPartInstance(regions=(springInstance,),size=
        springmeshSize)
262 #mesh the instance
263 springAssembly.generateMesh(regions=(springInstance,))
264 #create sets for boundary conditions
265 springbottomSet=springInstance.nodes.getByBoundingSphere(spring_coord
        [0],eps)
266 springtopSet=springInstance.nodes.getByBoundingSphere(spring_coord
        [-1],eps)
267 #create sets
268 springbottomSet=springAssembly.Set(name='Spring Bottom Node Set',nodes
         = springbottomSet)
269 springtopSet=springAssembly.Set(name='Spring Top Node Set',nodes =
        springtopSet)
270 #-- Mesh created --#
```

```
271
272 #-- Step --#
273 """
274 Create Load Step ie type of analysis= Static + Non-Linear Geometry
      effects
275 """
276 import step
277 #creating step for  static analysis with NonLinear Geometry option as
      ON
278 springStep=springModel.StaticStep(name='Static',previous='Initial',
      nlgeom=ON,maxNumInc=1000,initialInc=0.002, maxInc=0.05,minInc=1E
      -10,stabilizationMagnitude=0.0002,
279 stabilizationMethod=DAMPING_FACTOR, continueDampingFactors=False,
      adaptiveDampingRatio=0.05)
280 #-- Steps created --#
281
282 #-- Interaction --#
283 """
284 Create Contacts between the plates and spring
285 """
286 import interaction
287 springModel.ContactProperty('contact property')
288 springModel.interactionProperties['contact property'].NormalBehavior(
      pressureOverclosure=HARD, allowSeparation=ON,
289 constraintEnforcementMethod=PENALTY,
290 contactStiffness=DEFAULT, contactStiffnessScaleFactor=1.0,
      clearanceAtZeroContactPressure=0.0,stiffnessBehavior=LINEAR)
291 springModel.interactionProperties['contact property'].
      TangentialBehavior(formulation=PENALTY, directionality=ISOTROPIC,
      slipRateDependency=OFF,
292       pressureDependency=OFF, temperatureDependency=OFF, dependencies=0,
293       table=((friction_coeff, ), ), shearStressLimit=None,
            maximumElasticSlip=FRACTION,
294       fraction=0.005, elasticSlipStiffness=None)
295 springModel.StdInitialization(name='contact initialisation')
```

```
296 springModel.ContactStd(name='general contact interaction',
297     createStepName='Initial')
298 springModel.interactions['general contact interaction'].includedPairs.
    setValuesInStep(stepName='Initial', useAllstar=ON)
299 springModel.interactions['general contact interaction'].
    contactPropertyAssignments.appendInStep(stepName='Initial',
300 assignments=((GLOBAL, SELF, 'contact property'), ))
301 springModel.interactions['general contact interaction'].
    initializationAssignments.appendInStep(stepName='Initial',
302 assignments=((GLOBAL, SELF, 'contact initialisation'), ))
303 #-- Interaction created --#
304
305 #-- Load and BC --#
306 """
307 Create Boundary conditions
308 """
309 import load
310 #create BC at the bottom plate
311 # bottom center node restricted in xyz
312 #other nodes are free to move in x and z directions and given a fixed
    displacement of 0.0 in Y direction
313 #node on the Z axis is not allowed to move in x direction to avoid
    roataion of the plate
314 springModel.DisplacementBC(name='Bottom Plate Center BC',
    createStepName=springStep.name,region=bottomplatecenterSet,u1=0.0,
    u2=0.0,u3=0.0)
315 springModel.DisplacementBC(name='Bottom Plate BC',createStepName=
    springStep.name,region=bottomplateSet,u2=0.0)#loading is in Y
    direction
316 springModel.DisplacementBC(name='Bottom Plate Z-Node BC',
    createStepName=springStep.name,region=bottomplateZSet,u1=0.0,u2
    =0.0)#to avoid rotation
317 #create BC at the Top plate
318 # Top center node given a forced displacement of value '
    springDisplacement'
```

```
319 #other nodes are free to move in x and z directions and given a fixed
        displacement of 'springDisplacement' in Y direction
320 #node on the Z axis is not allowed to move in x direction to avoid
        roataion of the plate
321 springModel.DisplacementBC(name='Top Plate Center BC',createStepName=
        springStep.name,region=topplatecenterSet,u1=0.0,u2=
        springDisplacement,u3=0.0)
322 springModel.DisplacementBC(name='Top Plate BC',createStepName=
        springStep.name,region=topplateSet,u2=springDisplacement)#loading
        is in Y direction
323 springModel.DisplacementBC(name='Top Plate Z-Node BC',createStepName=
        springStep.name,region=topplateZSet,u1=0.0,u2=springDisplacement)#
        to avoid rotation
324 #-- Load and BC created --#
325
326 #-- Job --#
327 """
328 Create Job
329 """
330 import job
331 #create job
332 springJob=mdb.Job(name='springJob', model= springModel.name,
        description='', type=ANALYSIS,
333   atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
334   memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
335   explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF
        ,
336   modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='
        ',
337   scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus
        =32,numDomains=32,
338   numGPUs=0)
339 springModel.fieldOutputRequests['F-Output-1'].setValues(timeInterval
        =0.1,timeMarks =OFF)
340 regionDef = springAssembly.sets['Bottom Plate Center Set']
```

```
341 springModel.FieldOutputRequest(name='ReactionForces1', createStepName=
        springStep.name, variables=('RT','U', ),region=regionDef)
342 regionDef = springAssembly.sets['Bottom Plate Set']
343 springModel.FieldOutputRequest(name='ReactionForces2', createStepName=
        springStep.name, variables=('RT','U', ),region=regionDef)
344 regionDef = springAssembly.sets['Bottom Plate Z-Node Set']
345 springModel.FieldOutputRequest(name='ReactionForces3', createStepName=
        springStep.name, variables=('RT','U', ),region=regionDef)
346 regionDef = springAssembly.sets['Top Plate Center Set']
347 springModel.FieldOutputRequest(name='ReactionForces4', createStepName=
        springStep.name, variables=('RT','U', ),region=regionDef)
348 regionDef = springAssembly.sets['Top Plate Set']
349 springModel.FieldOutputRequest(name='ReactionForces5', createStepName=
        springStep.name, variables=('RT','U', ),region=regionDef)
350 regionDef = springAssembly.sets['Top Plate Z-Node Set']
351 springModel.FieldOutputRequest(name='ReactionForces6', createStepName=
        springStep.name, variables=('RT','U', ),region=regionDef)
352 #submit job
353 springJob.submit(consistencyChecking=OFF)
354 springJob.waitForCompletion()
355 #-- Job created --#
356 #save file
357 mdb.saveAs(pathName= mainfilepath+'/My_Model_solid'+'.cae')
358 #-- odb --#
359 """
360 Access the output database generated from the job and print out
        quantities of interest to txt files at current directory
361 """
362 import odb
363 from odbAccess import *
364 odb  = openOdb(path = mainfilepath+'/'+springJob.name+'.odb')
365 #define regions for bottom center and bottom plate pts
366 springOdb=odb.rootAssembly
367 odb_bottomcenter_region= springOdb.nodeSets['Bottom Plate Center Set']
368 odb_bottomznode_region= springOdb.nodeSets['Bottom Plate Z-Node Set']
```

```
369   odb_bottomplate_region= springOdb.nodeSets['Bottom Plate Set']
370   odb_topcenter_region= springOdb.nodeSets['Top Plate Center Set']
371   #loop over frames to populate the values of reaction force and
          displacement
372   sideforce = []
373   axialforce = []
374   displacement =[]
375   odbStep=odb.steps[springStep.name]
376   for i in range(len(odbStep.frames)):
377     displacement.append(abs(odbStep.frames[i].fieldOutputs['U'].
          getSubset(region =odb_topcenter_region ).values[0].data[1]))#
          appending abs value of Uy of the top node ## note vlaues[0] says
           that we have only one node in this subset
378     sideforce.append(tuple([sum(lists) for lists in zip(odbStep.frames[i
          ].fieldOutputs['RT'].getSubset(region =odb_bottomcenter_region).
          values[0].data,odbStep.frames[i].fieldOutputs['RT'].getSubset(
          region =odb_bottomznode_region).values[0].data)] ))# appending
          tuple as reaction value to side force
379     bottomplate_reactions= odbStep.frames[i].fieldOutputs['RT'].
          getSubset(region =odb_bottomplate_region)
380     tempbp=0
381     for ibp in range(len(bottomplate_reactions.values)):
382       tempbp=tempbp+bottomplate_reactions.values[ibp].data[1]# index 1
            for Fy
383     axialforce.append(tempbp)
384   #print value of the above variables to a text file
385   fsideforce = open("sideforce.txt","w")
386   fsideforce.write("Side forces [Fx Fy Fz] at bottom center node\n")
387   for i in sideforce:
388     fsideforce.write(str(i[0])+'  '+str(i[1])+' '+str(i[2])+'\n')
389   fsideforce.close()
390
391   fdisplacement  = open("displacement.txt","w")
392   fdisplacement.write("displacement abs(Uy) of Top center node\n")
393   for i in displacement:
```

```
394       fdisplacement.write(str(i)+'\n')
395     fdisplacement.close()
396
397     faxialforce  = open("axialforce.txt","w")
398     faxialforce.write("sum of axial force Fy due to bottomplate\n")
399     for i in axialforce:
400       faxialforce.write(str(i)+'\n')
401     faxialforce.close()
402
403     time_out= time.time()
404     total_time= (time_out-time_in)/60.0
405     ftime= open("total_time.txt","w")
406     ftime.write("Total time taken for computation in mins\n")
407     ftime.write(str(total_time))
408     ftime.close()
409     #-- odb created --#
410 else:
411     #print the files with all zeros and only one row of data
412     notinterferingfile  = open(mainfilepath+'/notinterfering.txt','w')
413     notinterferingfile.write("0")
414     notinterferingfile.close()
415     #save file
416     mdb.saveAs(pathName= mainfilepath+'/My_Model_solid'+'.cae')
417     #create the o/p files
418     fsideforce = open("sideforce.txt","w")
419     fsideforce.write("Side forces [Fx Fy Fz] at bottom center node\n")
420     fsideforce.write('0'+'  '+'0'+' '+'0'+'\n')
421     fsideforce.close()
422
423     fdisplacement  = open("displacement.txt","w")
424     fdisplacement.write("displacement abs(Uy) of Top center node\n")
425     fdisplacement.write('0'+'\n')
426     fdisplacement.close()
427
428     faxialforce  = open("axialforce.txt","w")
```

```
429    faxialforce.write("sum of axial force Fy due to bottomplate\n")
430    faxialforce.write('0'+'\n')
431    faxialforce.close()
432
433    time_out= time.time()
434    total_time= (time_out-time_in)/60.0
435    ftime= open("total_time.txt","w")
436    ftime.write("Total time taken for computation in mins\n")
437    ftime.write(str(total_time))
438    ftime.close()
439    import sys
440    sys.exit()
```

Beam Connector Model

```
#-------Inputs---------------#
"""
Variables and inputs
"""
#set working directory
import os
import numpy as np
mainfilepath =''# to be given by user
os.chdir(mainfilepath)
statusfile  = open(mainfilepath+'/status.txt','w')
statusfile.write("1")
statusfile.close()

#--------spring parameters-------#
#note dimensions are in mm we are using SI 'mm' units so for stress we have MPa, Mass-tonne,density-tonne/mm**3
wire_dia=13.5#wire diameter (mm)
spring_initial_radius=38.5 #initial radius of the spring (mm)
spring_max_radius=64.2819#maximum radius of the spring (mm)
spring_height=197.6001 #height of the spring (mm)
spring_Np=200#number of points in the spring
#--------parameters of the plates----------#
plate_thickness=10 #plate thickness
bottom_center= (0.0,0.0,0.0) #bottom plate center coordinate(tuple)#xyz to be in accordance with abaqus (loading in Y direction)
top_center= (0.0,spring_height+2*plate_thickness+wire_dia,0.0) #top plate center coordinates(tuple)#xyz to be in accordance with abaqus (loading in Y direction)
plate_radius=spring_max_radius*2
#-------------------------#

#Material properties variables (independent definition)
springyoungmodulus=225000#Mpa
```

```
poissonratio=0.3
springDisplacement=-173.0001#mm
springmeshSize=10.0

eps=10**-6
axialspring_stiffness=1000000#N/mm
radialspring_stiffness=10000#N/mm
radialspring_tension=100#N/mm
pos_refpt_spring_topturn=20#mm
friction_spring_stiff_top=1#N/mm
friction_spring_stiff_bottom=1#N/mm
#-------Read Coordinate Text File-------#
"""
Read xyz coordinates of the spring from the file
(This file is assumed to be present in mainfilepath)
"""
##NOTE: coordinatefile.txt to give xyz coordinates such that the Z
    coordinates are in axial direction,
#we are doing a manipulation by interchangin Y and Z coordinates here

filename='coordinatefile'
filepath= mainfilepath + '/' + filename + '.txt'
fid=open(filepath,'r')
spring_coord = []; spring_identifier=[]
for i in range(spring_Np):
  temp= fid.readline().split()
  temp=[float(j) for j in temp]#convert the coordinate strings to float
      and rotate using matrix [1 0 0, 0 0 1, 0 -1 0]
  spring_identifier.append(temp[0])
  a=temp[-2]
  temp[-2]=temp[-1]+plate_thickness+(wire_dia/2)#displacing to correct
      the height
  temp[-1]=-a
  spring_coord.append(temp[2:])

```

```
62 fid.close
63 #-------------------------------#
64
65 #-------------------------------#
66 #-------ABAQUS Main File--------#
67 #-------------------------------#
68 """
69 Abaqus scripting begins
70 """
71 from abaqus import *
72 from abaqusConstants import *
73 #creating a model
74 springModel=mdb.Model(name='Spring')
75 #--------Parts-----------------#
76 """
77 Create all geometric parts and prepare them for adequate meshing by
       partitioning them
78 """
79 import part
80
81 #-------part for spring--------#
82 #create a part for the spring
83 springPart = springModel.Part(name='Spring',dimensionality= THREE_D,
       type=DEFORMABLE_BODY)
84 #creating wire spline for the spring using sprign coordinates
85 springWire=springPart.WirePolyLine(points=spring_coord,mergeType=
       SEPARATE, meshable=ON, smoothClosedSpline=ON)
86 #-------Parts creadted----------#
87
88 #-------Material----------------#
89 """
90 Create Materials with material properties for plate and spring
91 """
92 import material
93 #create material
```

```
springMaterial = springModel.Material(name='Spring Material',description
    ='spring steel')
#assign youngs modulus and poissons ratio to both the materials
springelasticProperties= (springyoungmodulus,poissonratio)
springMaterial.Elastic(table=(springelasticProperties,))
#---Materials Created-----#

#---Beam profile -------#
"""
Create crosssectional profile for the Beam elements
"""
#creating the beam cross section for spring
springProfile = springModel.CircularProfile(name='Circular',r=wire_dia
    /2)
#-------Beam profile ----#

#-----Section and Orientation----#
"""
Create Sections for the different Properties and assign them to the
    respective parts
"""
import section

#creating beam section for spring
springSection = springModel.BeamSection(name='Spring Section',
    integration=DURING_ANALYSIS, profile=springProfile.name, material=
    springMaterial.name)
#section assignements to spring
wireregion = springPart.edges
springPart.SectionAssignment(region=(wireregion,),sectionName =
    springSection.name)

#beam orientation assignment here
"""
```

```
122 For defining we are taking a reference vector, the global Y vector. We
        find the Tangential vector
123 using coordinates of the ith edge and find the cross product with the
        reference vector. This new vector ,n1,
124 is essentially orthogonal to the tangential vector we just need to
        assign this as the orientation of the beam
125 """
126 #y=(0.0,1.0,0.0) #global y vector
127 #The cross product t x y comes out to be (-t3,0,t1)
128 for i in range(len(springPart.edges)-1):
129   xt=spring_coord[i+1][0]-spring_coord[i][0]#x component
130   yt=spring_coord[i+1][1]-spring_coord[i][1]#y component
131   zt=spring_coord[i+1][2]-spring_coord[i][2]#z component
132   lent=(xt**2+yt**2+zt**2)**(0.5)#length of edge tangential vector
133   t=(xt/lent,yt/lent,zt/lent)# edge tangential vector
134   n1=(-1*t[2],0.0,t[0])#cross product of t x y #hard coded here
135   region=springPart.edges.findAt(((spring_coord[i+1][0]+spring_coord[i
          ][0])/2,
136   (spring_coord[i+1][1]+spring_coord[i][1])/2,(spring_coord[i+1][2]+
          spring_coord[i][2])/2)) #finding cooresponding edge using mid
          point of edge
137   springPart.assignBeamSectionOrientation(method=N1_COSINES,region=(
          region,),n1=n1)
138 #----Sections and orientation created-------#
139
140 #-----Assembly-------#
141 """
142 Assemble instances of the parts into a global coordinate system
143 """
144 import assembly
145 import regionToolset
146 #create part instances
147 springAssembly =springModel.rootAssembly
148 springInstance=springAssembly.Instance(name='Spring Instance',part=
        springPart,dependent=OFF)
```

```
149 #-----Mesh---------#
150 """
151 Create Mesh for the instances in the assembly and create sets of nodes
        for application of BC
152 """
153 import mesh
154
155 ##code for meshing the spring
156 #select type of element and assign to the instance
157 elemType = mesh.ElemType(elemCode=B32, elemLibrary=STANDARD)
158 springAssembly.setElementType(regions=(springInstance.edges,),elemTypes
        =(elemType,))
159 #seed the instance
160 springAssembly.seedPartInstance(regions=(springInstance,),size=
        springmeshSize)
161 #mesh the instance
162 springAssembly.generateMesh(regions=(springInstance,))
163 #---------Mesh created---------#
164
165 #assign variables to coordinates of reference points
166 ref_pt_bottom_center=(bottom_center[0],bottom_center[1]+plate_thickness+
        wire_dia/2,bottom_center[2])
167 ref_pt_top_center=(top_center[0],top_center[1]-plate_thickness-wire_dia
        /2,top_center[2])
168 ref_bottom_loading_pt=(bottom_center[0],bottom_center[1]+plate_thickness
        +wire_dia/2-spring_height/4,bottom_center[2]) #taking a distance of
        spring_height/4 of the loading pt
169 ref_top_loading_pt=(top_center[0],top_center[1]-plate_thickness-wire_dia
        /2+spring_height/4,top_center[2]) #taking a distance of
        spring_height/4 of the loading pt
170 #create reference points using above coordinates
171 springAssembly.ReferencePoint(ref_pt_bottom_center)
172 springAssembly.ReferencePoint(ref_pt_top_center)
173 springAssembly.ReferencePoint(ref_bottom_loading_pt)
174 springAssembly.ReferencePoint(ref_top_loading_pt)
```

```
175 #springAssembly.ReferencePoint(ref_rotation_pt)
176
177 rp1=springAssembly.referencePoints.findAt(ref_pt_bottom_center)
178 bottomcenterSet=springAssembly.Set(name='Bottom Center Ref pt',
        referencePoints = (rp1,))
179
180 rp2=springAssembly.referencePoints.findAt(ref_pt_top_center)
181 topcenterSet=springAssembly.Set(name='Top Center Ref pt',referencePoints
        = (rp2,))
182
183 #This loop creates springs on the lower and upper halves of the spring
        using spring_identifier[i] to detect whether to make or not
184 #also the loop is making connectors with non-linear behavior to simulate
         the contacts. These connectors are connected to a reference point
        on either of the plates and corresponding spring point
185
186 import connectorBehavior   # required for connectors
187 #datum Csys required for connector orientation
188 connector_datum1=springAssembly.DatumCsysByThreePoints(name='Connector
        Datum1',coordSysType= CARTESIAN, origin=bottom_center,point1=(
        bottom_center[0],bottom_center[1]+plate_thickness,bottom_center[2]),
189 point2=(bottom_center[0],bottom_center[1], bottom_center[2]+plate_radius
        ));
190 connector_datum2=springAssembly.DatumCsysByThreePoints(name='Connector
        Datum2',coordSysType= CARTESIAN, origin=bottom_center,point1=(
        bottom_center[0],bottom_center[1]-plate_thickness,bottom_center[2]),
191 point2=(bottom_center[0],bottom_center[1], bottom_center[2]+plate_radius
        ));
192 connector_datum1=springAssembly.datums[connector_datum1.id]#datum object
193 connector_datum2=springAssembly.datums[connector_datum2.id]
194
195 bottomplateSet=[]
196 topplateSet=[]
197 bottomturnSet=[]
198 topturnSet=[]
```

```
199 bottomcollector=[]
200
201 for i in range(len(spring_identifier)):
202
203   if  spring_identifier[i]==1:
204     rpi=springInstance.vertices.findAt(spring_coord[i])
205     temp=springInstance.vertices.getByBoundingSphere(center=spring_coord
          [i],radius=eps)
206     bottomturnSet.append(temp)
207     #creating radial connector connecting the center pt with the dead
          turn spring pts (radial connector)
208     #create datum csys for connector
209     temp_datum=springAssembly.DatumCsysByThreePoints(name='Connector
          Datum#'+str(i),coordSysType= CARTESIAN, origin=
          ref_pt_bottom_center,point1=spring_coord[i],
210     point2=bottom_center);
211     temp_datum=springAssembly.datums[temp_datum.id]
212     connector_len=((spring_coord[i][0]-ref_pt_bottom_center[0])**2+(
          spring_coord[i][1]-ref_pt_bottom_center[1])**2+(spring_coord[i
          ][2]-ref_pt_bottom_center[2])**2)**0.5
213     #create elasticity for connector
214     temp_behavior=connectorBehavior.ConnectorElasticity(behavior=
          NONLINEAR,table=((-radialspring_stiffness*(10*
          spring_initial_radius),
215     -(10*spring_initial_radius)),(0,-(connector_len-eps-
          spring_initial_radius)),(0,0),(radialspring_tension*(10*
          spring_initial_radius),(10*spring_initial_radius))),components
          =(1, ))
216     #connector section definition
217     temp_sec1=springModel.ConnectorSection(name='non-linear-radial '+str
          (i),translationalType=AXIAL,behaviorOptions=(temp_behavior,))
218     #create wire for connector
219     tempwire=springAssembly.WirePolyLine(points=((rp1),(rpi)),mergeType=
          IMPRINT, meshable=OFF)
```

```
220     edge=springAssembly.edges.findAt(((spring_coord[i][0]+
            ref_pt_bottom_center[0])/2,(spring_coord[i][1]+
            ref_pt_bottom_center[1])/2,(spring_coord[i][2]+
            ref_pt_bottom_center[2])/2))
221     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec1.name)
222     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            temp_datum)
223   elif  spring_identifier[i]==2:
224     rpi=springInstance.vertices.findAt(spring_coord[i])
225     temp=springInstance.vertices.getByBoundingSphere(center=spring_coord
            [i],radius=eps)
226     #bottomturnSet.append(temp)
227     #creating radial connector connecting the center pt with the dead
            turn spring pts (radial connector)
228     #create datum csys for connector
229     temp_datum=springAssembly.DatumCsysByThreePoints(name='Connector
            Datum#'+str(i),coordSysType= CARTESIAN, origin=
            ref_pt_bottom_center,point1=spring_coord[i],
230     point2=bottom_center);
231     temp_datum=springAssembly.datums[temp_datum.id]
232     connector_len=((spring_coord[i][0]-ref_pt_bottom_center[0])**2+(
            spring_coord[i][1]-ref_pt_bottom_center[1])**2+(spring_coord[i
            ][2]-ref_pt_bottom_center[2])**2)**0.5
233     #create elasticity for connector
234     temp_behavior=connectorBehavior.ConnectorElasticity(behavior=
            NONLINEAR,table=((-radialspring_stiffness*(10*
            spring_initial_radius),
235     -(10*spring_initial_radius)),(0,-(connector_len-eps-
            spring_initial_radius)),(0,0),(radialspring_tension*(10*
            spring_initial_radius),(10*spring_initial_radius))),components
            =(1, ))
236     #connector section definition
237     temp_sec1=springModel.ConnectorSection(name='non-linear-radial '+str
            (i),translationalType=AXIAL,behaviorOptions=(temp_behavior,))
```

```
238     #create wire for connector
239     tempwire=springAssembly.WirePolyLine(points=((rp1),(rpi)),mergeType=
            IMPRINT, meshable=OFF)
240     edge=springAssembly.edges.findAt(((spring_coord[i][0]+
            ref_pt_bottom_center[0])/2,(spring_coord[i][1]+
            ref_pt_bottom_center[1])/2,(spring_coord[i][2]+
            ref_pt_bottom_center[2])/2))
241     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec1.name)
242     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            temp_datum)
243     h=spring_coord[0][1]-wire_dia/2
244     springAssembly.ReferencePoint((spring_coord[i][0],h,spring_coord[i
            ][2]));#ref point on bottom plate will be same as spring coord
            only that Y=plate y
245     temp_refpt=springAssembly.referencePoints.findAt((spring_coord[i
            ][0],h,spring_coord[i][2]))
246     temp_spring_pt=springInstance.vertices.findAt(spring_coord[i])
247     springAssembly.WirePolyLine(points=((temp_refpt),(temp_spring_pt)),
            mergeType=IMPRINT, meshable=OFF)
248
249     temp=connectorBehavior.ConnectorElasticity(behavior=NONLINEAR,table
            =((-axialspring_stiffness*(spring_height+spring_coord[i][1]-
            spring_coord[0][1]),
250     -(spring_height+spring_coord[i][1]-spring_coord[0][1])),(-eps*(
            spring_coord[i][1]-spring_coord[0][1]),-(spring_coord[i][1]-
            spring_coord[0][1]+eps)),(0,0),
251     (eps*(10*spring_initial_radius),(10*spring_initial_radius))),
            components=(1,))
252     temp_sec=springModel.ConnectorSection(name='non-linear-axial-bottom
            '+str(i),translationalType=SLOT,behaviorOptions=(temp,))
253     #connector assignment
254     edge=springAssembly.edges.findAt((spring_coord[i][0],h,spring_coord[
            i][2]))
```

```
255     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
256     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            connector_datum1)
257     bottomplateSet.append(temp_refpt)
258     h=spring_coord[-1][1]+wire_dia/2
259     springAssembly.ReferencePoint((spring_coord[i][0],h,spring_coord[i
            ][2]));#ref pointon bottom plate wil be same as spring coord
            only that Y=0
260     temp_refpt=springAssembly.referencePoints.findAt((spring_coord[i
            ][0],h,spring_coord[i][2]))
261     temp_spring_pt=springInstance.vertices.findAt(spring_coord[i])
262     springAssembly.WirePolyLine(points=((temp_refpt),(temp_spring_pt)),
            mergeType=IMPRINT, meshable=OFF)
263     temp=connectorBehavior.ConnectorElasticity(behavior=NONLINEAR,table
            =((-axialspring_stiffness*(spring_height+(spring_coord[-1][1]-
            spring_coord[i][1])),
264     -(spring_height+(spring_coord[-1][1]-spring_coord[i][1]))),(-eps*(
            spring_coord[-1][1]-spring_coord[i][1]),-(spring_coord[-1][1]-
            spring_coord[i][1]+eps)),(0,0),
265     (eps*(10*spring_initial_radius),(10*spring_initial_radius))),
            components=(1,))
266     temp_sec=springModel.ConnectorSection(name='non-linear-axial-top '+
            str(i),translationalType=SLOT,behaviorOptions=(temp,))
267     edge=springAssembly.edges.findAt((spring_coord[i][0],h,spring_coord[
            i][2]))
268     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
269     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            connector_datum2)
270     topplateSet.append(temp_refpt)
271   elif spring_identifier[i]==3:
272     h=spring_coord[0][1]-wire_dia/2
```

```
273     springAssembly.ReferencePoint((spring_coord[i][0],h,spring_coord[i
            ][2]));#ref point on bottom plate will be same as spring coord
            only that Y=plate y
274     temp_refpt=springAssembly.referencePoints.findAt((spring_coord[i
            ][0],h,spring_coord[i][2]))
275     temp_spring_pt=springInstance.vertices.findAt(spring_coord[i])
276     springAssembly.WirePolyLine(points=((temp_refpt),(temp_spring_pt)),
            mergeType=IMPRINT, meshable=OFF)
277     temp=connectorBehavior.ConnectorElasticity(behavior=NONLINEAR,table
            =((-axialspring_stiffness*(spring_height+spring_coord[i][1]-
            spring_coord[0][1]),
278     -(spring_height+spring_coord[i][1]-spring_coord[0][1])),(-eps*(
            spring_coord[i][1]-spring_coord[0][1]),-(spring_coord[i][1]-
            spring_coord[0][1]+eps)),(0,0),
279     (eps*(10*spring_initial_radius),(10*spring_initial_radius))),
            components=(1,))
280     temp_sec=springModel.ConnectorSection(name='non-linear-axial-bottom
            '+str(i),translationalType=SLOT,behaviorOptions=(temp,))
281     #connector assignment
282     edge=springAssembly.edges.findAt((spring_coord[i][0],h,spring_coord[
            i][2]))
283     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
284     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            connector_datum1)
285     bottomplateSet.append(temp_refpt)
286     h=spring_coord[-1][1]+wire_dia/2
287     springAssembly.ReferencePoint((spring_coord[i][0],h,spring_coord[i
            ][2]));#ref pointon bottom plate wil be same as spring coord
            only that Y=0
288     temp_refpt=springAssembly.referencePoints.findAt((spring_coord[i
            ][0],h,spring_coord[i][2]))
289     temp_spring_pt=springInstance.vertices.findAt(spring_coord[i])
290     springAssembly.WirePolyLine(points=((temp_refpt),(temp_spring_pt)),
            mergeType=IMPRINT, meshable=OFF)
```

```
291     temp=connectorBehavior.ConnectorElasticity(behavior=NONLINEAR, table
            =((-axialspring_stiffness*(spring_height+(spring_coord[-1][1]-
            spring_coord[i][1]))),
292     -(spring_height+(spring_coord[-1][1]-spring_coord[i][1]))),(-eps*(
            spring_coord[-1][1]-spring_coord[i][1]),-(spring_coord[-1][1]-
            spring_coord[i][1]+eps)),(0,0),
293     (eps*(10*spring_initial_radius),(10*spring_initial_radius))),
            components=(1,))
294     temp_sec=springModel.ConnectorSection(name='non-linear-axial-top '+
            str(i),translationalType=SLOT,behaviorOptions=(temp,))
295     edge=springAssembly.edges.findAt((spring_coord[i][0],h,spring_coord[
            i][2]))
296     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
297     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            connector_datum2)
298     topplateSet.append(temp_refpt)
299   elif spring_identifier[i]==4:
300     rpi=springInstance.vertices.findAt(spring_coord[i])
301     temp=springInstance.vertices.getByBoundingSphere(center=spring_coord
            [i],radius=eps)
302     temp_datum=springAssembly.DatumCsysByThreePoints(name='Connector
            Datum#'+str(i),coordSysType= CARTESIAN, origin=ref_pt_top_center
            ,point1=spring_coord[i],
303     point2=top_center);
304     temp_datum=springAssembly.datums[temp_datum.id]
305     connector_len=((spring_coord[i][0]-ref_pt_top_center[0])**2+(
            spring_coord[i][1]-ref_pt_top_center[1])**2+(spring_coord[i][2]-
            ref_pt_top_center[2])**2)**0.5
306     #create elasticity for connector
307     temp_behavior=connectorBehavior.ConnectorElasticity(behavior=
            NONLINEAR,table=((-radialspring_stiffness*(10*
            spring_initial_radius),
308     -(10*spring_initial_radius)),(0,-(connector_len-eps-
            spring_initial_radius)),(0,0),(radialspring_tension*(10*
```

```
        spring_initial_radius),(10*spring_initial_radius))),components
        =(1, ))
309     #connector section definition
310     temp_sec=springModel.ConnectorSection(name='non-linear-radial '+str(
        i),translationalType=AXIAL, behaviorOptions=(temp_behavior,))
311     #create wire for connector
312     springAssembly.WirePolyLine(points=((rp2),(rpi)),mergeType=IMPRINT,
        meshable=OFF)
313     edge=springAssembly.edges.findAt(((spring_coord[i][0]+
        ref_pt_top_center[0])/2,(spring_coord[i][1]+ref_pt_top_center
        [1])/2,(spring_coord[i][2]+ref_pt_top_center[2])/2))
314     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
        temp_sec.name)
315     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
        temp_datum)
316     h=spring_coord[0][1]-wire_dia/2
317     springAssembly.ReferencePoint((spring_coord[i][0],h,spring_coord[i
        ][2]));#ref point on bottom plate will be same as spring coord
        only that Y=plate y
318     temp_refpt=springAssembly.referencePoints.findAt((spring_coord[i
        ][0],h,spring_coord[i][2]))
319     temp_spring_pt=springInstance.vertices.findAt(spring_coord[i])
320     springAssembly.WirePolyLine(points=((temp_refpt),(temp_spring_pt)),
        mergeType=IMPRINT, meshable=OFF)
321     temp=connectorBehavior.ConnectorElasticity(behavior=NONLINEAR, table
        =((-axialspring_stiffness*(spring_height+spring_coord[i][1]-
        spring_coord[0][1]),
322     -(spring_height+spring_coord[i][1]-spring_coord[0][1])),(-eps*(
        spring_coord[i][1]-spring_coord[0][1]),-(spring_coord[i][1]-
        spring_coord[0][1]+eps)),(0,0),
323     (eps*(10*spring_initial_radius),(10*spring_initial_radius))),
        components=(1,))
324     temp_sec=springModel.ConnectorSection(name='non-linear-axial-bottom
        '+str(i),translationalType=SLOT, behaviorOptions=(temp,))
325     #connector assignment
```

```
326     edge=springAssembly.edges.findAt((spring_coord[i][0],h,spring_coord[
            i][2]))
327     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
328     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            connector_datum1)
329     bottomplateSet.append(temp_refpt)
330     h=spring_coord[-1][1]+wire_dia/2
331     springAssembly.ReferencePoint((spring_coord[i][0],h,spring_coord[i
            ][2]));#ref pointon bottom plate wil be same as spring coord
            only that Y=0
332     temp_refpt=springAssembly.referencePoints.findAt((spring_coord[i
            ][0],h,spring_coord[i][2]))
333     temp_spring_pt=springInstance.vertices.findAt(spring_coord[i])
334     springAssembly.WirePolyLine(points=((temp_refpt),(temp_spring_pt)),
            mergeType=IMPRINT, meshable=OFF)
335     temp=connectorBehavior.ConnectorElasticity(behavior=NONLINEAR,table
            =((-axialspring_stiffness*(spring_height+(spring_coord[-1][1]-
            spring_coord[i][1])),
336     -(spring_height+(spring_coord[-1][1]-spring_coord[i][1]))),(-eps*(
            spring_coord[-1][1]-spring_coord[i][1]),-(spring_coord[-1][1]-
            spring_coord[i][1]+eps)),(0,0),
337     (eps*(10*spring_initial_radius),(10*spring_initial_radius))),
            components=(1,))
338     temp_sec=springModel.ConnectorSection(name='non-linear-axial-top '+
            str(i),translationalType=SLOT,behaviorOptions=(temp,))
339     edge=springAssembly.edges.findAt((spring_coord[i][0],h,spring_coord[
            i][2]))
340     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
341     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            connector_datum2)
342     topplateSet.append(temp_refpt)
343   elif spring_identifier[i]==5:
344     rpi=springInstance.vertices.findAt(spring_coord[i])
```

```
345     temp=springInstance.vertices.getByBoundingSphere(center=spring_coord
            [i],radius=eps)
346     topturnSet.append(temp)
347     temp_datum=springAssembly.DatumCsysByThreePoints(name='Connector
            Datum#'+str(i),coordSysType= CARTESIAN, origin=ref_pt_top_center
            ,point1=spring_coord[i],
348     point2=top_center);
349     temp_datum=springAssembly.datums[temp_datum.id]
350     connector_len=((spring_coord[i][0]-ref_pt_top_center[0])**2+(
            spring_coord[i][1]-ref_pt_top_center[1])**2+(spring_coord[i][2]-
            ref_pt_top_center[2])**2)**0.5
351     #create elasticity for connector
352     temp_behavior=connectorBehavior.ConnectorElasticity(behavior=
            NONLINEAR,table=((-radialspring_stiffness*(10*
            spring_initial_radius),
353     -(10*spring_initial_radius)),(0,-(connector_len-eps-
            spring_initial_radius)),(0,0),(radialspring_tension*(10*
            spring_initial_radius),(10*spring_initial_radius))),components
            =(1, ))
354     #connector section definition
355     temp_sec=springModel.ConnectorSection(name='non-linear-radial '+str(
            i),translationalType=AXIAL,behaviorOptions=(temp_behavior,))
356     #create wire for connector
357     springAssembly.WirePolyLine(points=((rp2),(rpi)),mergeType=IMPRINT,
            meshable=OFF)
358     edge=springAssembly.edges.findAt(((spring_coord[i][0]+
            ref_pt_top_center[0])/2,(spring_coord[i][1]+ref_pt_top_center
            [1])/2,(spring_coord[i][2]+ref_pt_top_center[2])/2))
359     csa =springAssembly.SectionAssignment(region=(edge,),sectionName=
            temp_sec.name)
360     springAssembly.ConnectorOrientation(region=csa.getSet(),localCsys1=
            temp_datum)
361
362 rpi = springInstance.vertices.getByBoundingSphere(center=spring_coord
    [0],radius=eps)
```

```
363 rpf = springInstance.vertices.getByBoundingSphere(center=spring_coord
        [-1],radius=eps)
364
365 #now we need to create sets of bottomplate reference points and Top
        plate reference points
366 bottomplateSet=springAssembly.Set(name='Bottom Plate Ref pts',
        referencePoints = tuple(bottomplateSet))
367 topplateSet=springAssembly.Set(name='Top Plate Ref pts',referencePoints
       =tuple(topplateSet))
368 bottomturnfirstSet=springAssembly.Set(name='Bottom Turn 1st pt',vertices
        = rpi)
369 topturnlastSet=springAssembly.Set(name='Top Turn last pt',vertices = rpf
        )
370
371 if not(not(bottomturnSet)):
372   springAssembly.Set(name='All Bottom Turn pts',vertices = tuple(
          bottomturnSet))
373   bottomturnSet=springAssembly.SetByBoolean(name='Bottom Turn pts',
          operation=DIFFERENCE, sets=(springAssembly.sets['All Bottom Turn
          pts'], springAssembly.sets['Bottom Turn 1st pt'], ))
374   del springAssembly.sets['All Bottom Turn pts']
375 else: no_bottomturnSet = 1# ie True
376
377 if not(not(topturnSet)):
378   springAssembly.Set(name='All Top Turn pts',vertices = tuple(topturnSet
          ))
379   topturnSet=springAssembly.SetByBoolean(name='Top Turn pts', operation=
          DIFFERENCE, sets=(springAssembly.sets['All Top Turn pts'],
          springAssembly.sets['Top Turn last pt'], ))
380   del springAssembly.sets['All Top Turn pts']
381 else: no_topturnSet = 1# ie True
382
383 connectorSet=springAssembly.Set(edges=springAssembly.edges,name='
        Conectors')
384 rpf = regionToolset.Region(vertices=rpf)
```

```
385 #create a spring Dashpot connecting top turn last pt to a reference pt
        to simulate friction on plate
386 #create reference pt for spring
387 last_pt=np.array(spring_coord[-1])
388 second_last_pt=np.array(spring_coord[-2])
389 v_hat= (last_pt-second_last_pt)
390 v_hat= v_hat/np.linalg.norm(v_hat)
391 refpt_top_turn_spring=last_pt+pos_refpt_spring_topturn*v_hat
392 refpt_top_turn_spring=tuple(refpt_top_turn_spring)
393 springAssembly.ReferencePoint(refpt_top_turn_spring)
394 rp5=springAssembly.referencePoints.findAt(refpt_top_turn_spring)
395 topfrictionspringSet=springAssembly.Set(name='top turn friction spring
        pt',referencePoints = (rp5,))
396 rp5=regionToolset.Region(referencePoints=(rp5,))
397 #create connecting spring
398 springAssembly.engineeringFeatures.TwoPointSpringDashpot(regionPairs=((
        rpf,rp5),),name='friction spring top turn',axis=NODAL_LINE,
399                         springBehavior=ON,springStiffness=
                                friction_spring_stiff_top)
400 rpi = regionToolset.Region(vertices=rpi)
401 #create a spring Dashpot connecting top turn last pt to a reference pt
        to simulate friction on plate
402 #create reference pt for spring
403 refpt_bottom_turn_spring=((spring_coord[0][0]),(spring_coord[0][1]),(
        spring_coord[0][2]+pos_refpt_spring_topturn))
404 springAssembly.ReferencePoint(refpt_bottom_turn_spring)
405 rp6=springAssembly.referencePoints.findAt(refpt_bottom_turn_spring)
406 bottomfrictionspringSet=springAssembly.Set(name='bottom turn friction
        spring pt',referencePoints = (rp6,))
407 rp6=regionToolset.Region(referencePoints=(rp6,))
408 #create connecting spring
409 springAssembly.engineeringFeatures.TwoPointSpringDashpot(regionPairs=((
        rpi,rp6),),name='friction spring bottom turn',axis=NODAL_LINE,
410                         springBehavior=ON,springStiffness=
                                friction_spring_stiff_bottom)
```

```
411 #----Assembly Created-----#
412
413 #------Step----#
414 """
415 Create Load Step ie type of analysis= Static + Non-Linear Geometry
        effects
416 """
417 import step
418 #creating step for  static analysis with NonLinear Geometry option as ON
419 springStep=springModel.StaticStep(name='Static',previous='Initial',
        nlgeom=ON,maxNumInc=1000,initialInc=0.001, maxInc=0.05,minInc=1E-10,
        stabilizationMagnitude=0.0002,
420      stabilizationMethod=DAMPING_FACTOR, continueDampingFactors=False,
            adaptiveDampingRatio=0.05)
421 #---Steps created---#
422
423 #-----Load and BC-----#
424 """
425 Create Boundary conditions
426 """
427 import load
428 springModel.DisplacementBC(name='bottom plate BC',createStepName=
        springStep.name,region=bottomplateSet,u2=0.0,ur1=0.0,ur2=0.0,ur3
        =0.0)
429 springModel.DisplacementBC(name='top plate BC',createStepName=springStep
        .name,region=topplateSet,u2=springDisplacement,ur1=0.0,ur2=0.0,ur3
        =0.0)
430 springModel.DisplacementBC(name='bottom center BC',createStepName=
        springStep.name,region=bottomcenterSet,u1=0.0,u2=0.0,u3=0.0)
431 springModel.DisplacementBC(name='top center BC',createStepName=
        springStep.name,region=topcenterSet,u1=0.0,u2=springDisplacement,u3
        =0.0)
432 springModel.DisplacementBC(name='bottom turn 1st pt BC',createStepName=
        springStep.name,region=bottomturnfirstSet,u2=0.0)
```

```
433 springModel.DisplacementBC(name='top turn last pt BC',createStepName=
        springStep.name,region=topturnlastSet,u2=springDisplacement)
434 if not(no_bottomturnSet):
435   springModel.DisplacementBC(name='bottom turn BC',createStepName=
          springStep.name,region=bottomturnSet,u2=0.0)
436 if not(no_topturnSet):
437   springModel.DisplacementBC(name='top turn BC',createStepName=
          springStep.name,region=topturnSet,u2=springDisplacement)
438 springModel.DisplacementBC(name='top friction spring ref pt BC',
        createStepName=springStep.name,region=topfrictionspringSet,u1=0.0,u2
        =springDisplacement,u3=0.0)
439 springModel.DisplacementBC(name='bottom friction spring ref pt BC',
        createStepName=springStep.name,region=bottomfrictionspringSet,u3
        =0.0)
440 #----Load and BC created----#
441
442 #--------Job--------#
443 """
444 Create Job
445 """
446 import job
447
448 springJob=mdb.Job(name='springJob', model= springModel.name, description
        ='', type=ANALYSIS,
449     atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
450     memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
451     explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF
          ,
452     modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='
          ',
453     scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus
          =32,numDomains=32,
454     numGPUs=0)
455 springModel.FieldOutputRequest(name='ReactionForces', createStepName=
        springStep.name, variables=('RT','U', ))
```

```
456 springModel.FieldOutputRequest(name='Connector forces',
457     createStepName=springStep.name, variables=('CTF', 'CEF', 'CU', 'CUE'
        , 'CUP'),
458     region=connectorSet, sectionPoints=DEFAULT, rebar=EXCLUDE)
459 #submit job
460 springJob.submit(consistencyChecking=OFF)
461 springJob.waitForCompletion()
462 #---Job created-----#
463 #save file
464 mdb.saveAs(pathName= mainfilepath+'/My_Model'+'.cae')
465 #-----odb---------#
466 """
467 Access the output database generated from the job and print out
    quantities of interest to txt files at current directory
468 """
469 import odb
470 from odbAccess import *
471 odb  = openOdb(path = mainfilepath+'/'+springJob.name+'.odb')
472 #define regions for bottom center and bottom plate pts
473 springOdb=odb.rootAssembly
474 odb_bottomcenter_region= springOdb.nodeSets['Bottom Center Ref pt']
475 odb_bottomplate_region= springOdb.nodeSets['Bottom Plate Ref pts']
476 odb_bottomturnfirst_region= springOdb.nodeSets['Bottom Turn 1st pt']
477 if not(no_bottomturnSet):
478   odb_bottomturn_region= springOdb.nodeSets['Bottom Turn pts']
479 odb_topcenter_region= springOdb.nodeSets['Top Center Ref pt']
480 #loop over frames to populate the values of reaction force and
    displacement
481 sideforce = []
482 axialforce = []
483 displacement =[]
484 odbStep=odb.steps[springStep.name]
485 for i in range(len(odbStep.frames)):
486   displacement.append(abs(odbStep.frames[i].fieldOutputs['U'].getSubset(
        region =odb_topcenter_region ).values[0].data[1]))#appending abs
```

```
        value of Uy of the top node ## note vlaues[0] says that we have
        only one node in this subset
487     sideforce.append(tuple(odbStep.frames[i].fieldOutputs['RT'].getSubset(
        region =odb_bottomcenter_region).values[0].data))# appending tuple
         as reaction value to side force
488     bottomplate_reactions= odbStep.frames[i].fieldOutputs['RT'].getSubset(
        region =odb_bottomplate_region)
489     tempbp=0
490     for ibp in range(len(bottomplate_reactions.values)):
491       tempbp=tempbp+bottomplate_reactions.values[ibp].data[1]# index 1 for
            Fy
492     bottomturnfirst_reactions= odbStep.frames[i].fieldOutputs['RT'].
        getSubset(region =odb_bottomturnfirst_region).values[0].data[1]#
        again only one node in the subset
493     if not(no_bottomturnSet):
494       bottomturn_reactions= odbStep.frames[i].fieldOutputs['RT'].getSubset
          (region =odb_bottomturn_region)
495       tempbt=0
496       for ibt in range(len(bottomturn_reactions.values)):
497         tempbp=tempbp+bottomturn_reactions.values[ibt].data[1]# index 1
              for Fy
498     else:tempbt=0
499     axialforce.append(tempbp+tempbt+bottomturnfirst_reactions)
500 #print value of the above variables to a text file
501 fsideforce = open("sideforce.txt","w")
502 fsideforce.write("Side forces [Fx Fy Fz] at bottom center node\n")
503 for i in sideforce:
504     fsideforce.write(str(i[0])+'  '+str(i[1])+' '+str(i[2])+'\n')
505 fsideforce.close()
506
507 fdisplacement  = open("displacement.txt","w")
508 fdisplacement.write("displacement abs(Uy) of Top center node\n")
509 for i in displacement:
510     fdisplacement.write(str(i)+'\n')
511 fdisplacement.close()
```

```
512
513 faxialforce  = open("axialforce.txt","w")
514 faxialforce.write("sum of axial force Fy due to bottomplate+ bottomturn+
        bottomturnfirst nodes\n")
515 for i in axialforce:
516   faxialforce.write(str(i)+'\n')
517 faxialforce.close()
518 #-----odb created--------#
519 statusfile  = open(mainfilepath+'/status.txt','w')
520 statusfile.write("0")
521 statusfile.close()
522 import sys
523 sys.exit()
```